List of Experiments

S.No.	Experiment
1	Implement Affine Cipher
2	Implement Hill Cipher
3	Implement Vigener Cipher
4	Implement Playfair Cipher
5	Implement Keyless and Keyed Transposition Cipher
6	Implement Data Encryption Standards
7	Implement Advanced Encryption Standards
8	Implement Public Cryptography “RSA”
9	Implement Public Key Cryptography ElGamal
10	Implement Public Key Cryptography Rabin Algorithm
11	Implement Public Key Cryptography Elliptic Curve Cryptography
12	Implement Secure Hash Algorithm“SHA-512”

Q1. Implement Affine Cipher.

Theory: The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing whichletter goes to which.
The whole process relies on working modulo m (the length of the alphabet used). In the affine cipher, the letters of an alphabet of sizem are first mapped to the integers in the range 0 … m-1.

The 'key' for the Affine cipher consists of 2 numbers, we'll call thema and b. The following discussion assumes the use of a 26 characteralphabet (m = 26). a should be chosen to be relatively prime to m (i.e. a should have no factors in common with m)

It uses modular arithmetic to transform the integer that each plaintext letter corresponds to into another integer that correspondto a ciphertext letter.

Encryption

E (x) = (a x + b) mod m modulus m:

size of the alphabeta and b: key of the

cipher.

a must be chosen such that a and m are coprime.

Decryption

In deciphering the ciphertext, we must perform the opposite (or inverse) functions on the ciphertext to retrieve the plaintext. Onceagain, the first step is to convert each of the ciphertext letters intotheir integer values. The decryption function is

D (x) = a^-1 (x - b) mod m

a^-1 : modular multiplicative inverse of a modulo m. i.e., it satisfiesthe equation

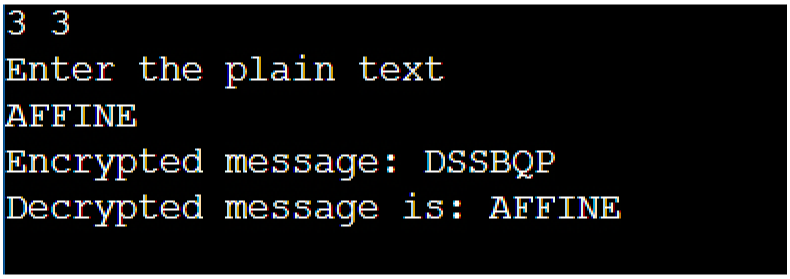
1 = a a^-1 mod m

Code;

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


int mul(int b,int n) {

   if(_gcd(b,n)!=1) {
      cout<<"\n GCD DNE Hence No Multiplicative Inverse exists ";
      return -1;
   }

   int q,r,t,t1=0,t2=1;
   int r1=max(n,b);
   int r2=min(n,b);


   q=r1/r2;
   r=r1-q*r2;

   while(r2!=0) {

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

   if(r2!=0)q=r1/r2;
```

```
    r=r1-q*r2;
}

return t1<0?n+t1:t1;

}



int main()
{
   //Affine Cipher
   int k1,k2;
   cin>>k1>>k2;


   string msg;
   cout<<"Enter the plaint text"<<endl;
   cin>>msg;

   string temp_cipher;

   for(auto it:msg) {
      char ch=it;

      int temp=it-'a';
      temp*=k1;
      temp%=26;
      int temp2;
      temp2=(temp+k2)%26;
      temp2+='a';

      temp_cipher+=(char)(temp2);
   }

    cout<<"Encrypted message: ";
    cout<<temp_cipher<<endl;
    int mulinv=mul(k1,26);

    cout<<mulinv;
    string dec;
    for(auto it:temp_cipher) {
      char ch=it;

      int temp=ch-'a';
      temp-=k2;
      temp%=26;
      if(temp<0) temp+=26;
      int temp2;

      temp2=temp*mulinv;
      temp2%=26;

      temp2+='a';

      dec+=temp2;
  }


    cout<<"Decrypted message is: "<<dec;
    return 0;
}
```

OUTPUT:



```
3 3
Enter the plain text
AFFINE
Encrypted message: DSSBQP
Decrypted message is: AFFINE
```

Q2. Implement Hill Cipher.

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often thesimple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of nletters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo26)

Exanple:

Input : Plaintext: ACTKey:

GYBNQKURP

Output : Ciphertext: POH

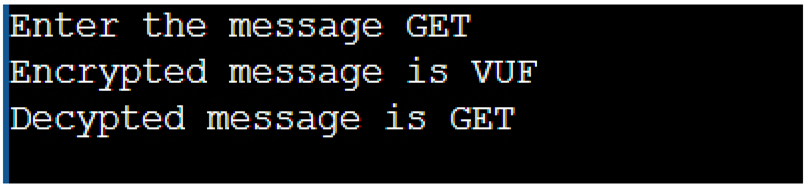
Decryption

To decrypt the message, we turn the ciphertext back into a vector,then simply multiply by the inverse matrix of the key matrix

Code;

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

int mul(int a, int m) {

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

}

void cofactor(int matrix[3][3], int temp[3][3], int p, int q, int n) {
    int i = 0, j = 0;

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

                if (j == n - 1) {
                    j = 0;
                    i++;
                }
            }
        }
    }
}

int deter(int matrix[3][3], int n) {
    if (n == 2) {
        return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]);
    }

    int det = 0;

    for (int f = 0; f < n; f++) {
        int temp[3][3];
        cofactor(matrix, temp, 0, f, n);
        det += (f % 2 == 0 ? 1 : -1) * matrix[0][f] * deter(temp, n - 1);
    }

    return det;
}
int adjoint[3][3];


void findAdjoint(int matrix[3][3]) {

    int sign = 1;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            int temp[3][3];
            cofactor(matrix, temp, i, j, 3);
            sign = ((i + j) % 2 == 0) ? 1 : -1;
            adjoint[j][i] = (sign * deter(temp, 2))%26;
        }
    }
}

int main()
{
    int key[3][3];
    key[0][0]=6;
    key[0][1]=24;
    key[0][2]=1;
    key[1][0]=13;
    key[1][1]=16;
    key[1][2]=10;
    key[2][0]=20;
    key[2][1]=17;
    key[2][2]=15;


    cout<<"Enter the message ";
     string plain_text;
     cin>>plain_text;
     int pt[3][1];
     pt[0][0]=plain_text[0]-'A';
     pt[1][0]=plain_text[1]-'A';
     pt[2][0]=plain_text[2]-'A';

     int cipher[3][1];

     for(int i=0;i<3;i++) {
         cipher[i][0]=(key[i][0]*pt[0][0]+key[i][1]*pt[1][0]+key[i][2]*pt[2][0])%26;
     }
    cout<<"Encrypted message is ";
    cout<<(char)(cipher[0][0]+'A')<<(char)(cipher[1][0]+'A')<<(char)(cipher[2][0]+'A');

    int det_key=deter(key,3)%26;
    int key_inverse=mul(det_key,26);
    findAdjoint(key);
    int key_inverse_mat[3][3];
    for(int i=0;i<3;i++){
     for(int j=0;j<3;j++){
         key_inverse_mat[i][j]=(adjoint[i][j]*key_inverse)%26;

     if(key_inverse_mat[i][j]<0) key_inverse_mat[i][j]+=26;
     }
    }

    int decry[3][1];
    for(int i=0;i<3;i++)

decry[i][0]=(key_inverse_mat[i][0]*cipher[0][0]+key_inverse_mat[i][1]*cipher[1][0]+key_invers
e_mat[i][2]*cipher[2][0])%26;

   cout<<endl<<"Decrypted message is ";
    cout<<(char)(decry[0][0]+'A')<<(char)(decry[1][0]+'A')<<(char)(decry[2][0]+'A');
    return 0;
}
```

OUTPUT:



```
Enter the message GET
Encrypted message is VUF
Decypted message is GET
```

Q3. Implement Vigenere Cipher.

Theory:

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

The table consists of the alphabets written out 26 times in differentrows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.

At different points in the encryption process, the cipher uses adifferent alphabet from one of the rows.

The alphabet used at each point depends on a repeating keyword.

Example:

Input : Plaintext : GEEKSFORGEEKS

Keyword : AYUSH

Output : Ciphertext : GCYCZFMLYLEIM

For generating key, the given keyword is repeated in a circular manner until it matches the length of the plain text.

The keyword "AYUSH" generates the key "AYUSHAYUSHAYU"

The plain text is then encrypted using the processexplained below.

Encryption:

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

Decryption:

Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letterin this row, and then using the column's label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in columnG, which is the first plaintext letter. Next, we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.

Encryption Forumla

The plaintext(P) and key(K) are added modulo 26.$E_i = (P_i + K_i)$ mod 26

Decryption Formula

$D_i = (E_i - K_i + 26)$ mod 26

Note: D_i denotes the offset of the i-th character of the plaintext. Likeoffset of **A** is 0 and of **B** is 1 and so on.

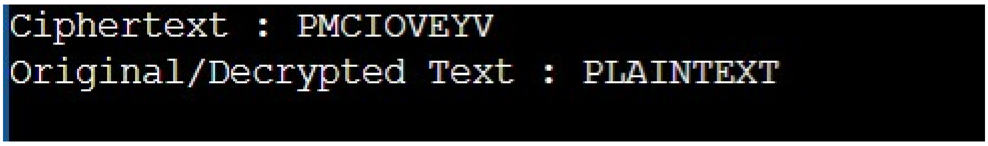
Code;

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

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

string cipherText(string str, string key)
{
        string cipher_text;
```

```
    for (int i = 0; i < str.size(); i++)
    {
        char x = (str[i] + key[i]) %26;
        x += 'A';
        cipher_text.push_back(x);
    }
    return cipher_text;
}

string originalText(string cipher_text, string key)
{
    string orig_text;

    for (int i = 0 ; i < cipher_text.size(); i++)
    {

        char x = (cipher_text[i] - key[i] + 26) %26;
        x += 'A';
        orig_text.push_back(x);
```

```
          }
          return orig_text;
}

int main()
{
          string str = "PLAINTEXT";string
          keyword = "ABC";

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

          cout << "Ciphertext : "<< cipher_text << endl;

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

OUTPUT:



```
Ciphertext : PMCIOVEYV
Original/Decrypted Text : PLAINTEXT
```

Q4. Implement playfair cipher algorithm.

Theory:

The Playfair cipher was the first practical digraph substitutioncipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II. This wasbecause Playfair is reasonably fast to use and requires nospecial equipment.

Encryption Technique

For the encryption process let us consider the followingexample:

Keytext: Monarchy Plaintext:

instruments

1. **Generate the key Square(5×5):**

   - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the25 alphabets must be unique and one letter of thealphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If theplaintext contains J, then it is replaced by I.

- The initial alphabets in the key square are the unique alphabets of the key in the order in whichthey appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to encrypt the plain text:** The plaintext is splitinto pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

   For example:

PlainText: "instruments"

After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'

1. Pair cannot be made with same letter. Break the letter insingle and add a bogus letter to the previous letter.

Plain Text: “hello”

After Split: ‘he’ ‘lx’ ‘lo’

Here **‘x’** is the bogus letter.

2. If the letter is standing alone in the process of pairing, thenadd an extra bogus letter with the alone letter

Plain Text: “helloe” **AfterSplit:** ‘he’ ‘lx’
‘lo’ ‘ez’ Here **‘z’** is the bogus letter.

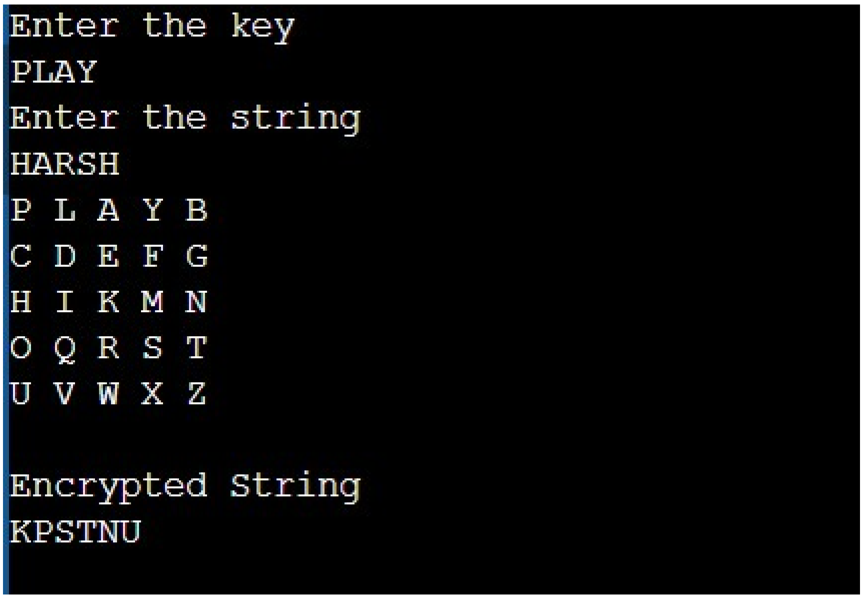
Code:

```
#include<iostream>
#include<unordered_map>
using namespace std;

int main(){
    char mat[5][5];
    unordered_map<char,pair<int,int>>m;
    string key;
    cout<<"Enter the key"<<endl;
    cin>>key;

    for(int i=0;i<key.length();i++){
        if(key[i]=='J')
            key[i]='I';
    }

    string acplain;
    cout<<"Enter the string"<<endl;
    cin>>acplain;
    string plain="";
    int ind;
```

```
for(int i=0;i<acplain.length()-1;i+=2){
    if(acplain[i]==acplain[i+1]){
        plain+=acplain[i];
        plain+='X';
        i--;
    }
    else{
        plain+=acplain[i];
        plain+=acplain[i+1];
    }
    ind=i;
}
ind+=2;
if(ind==acplain.length()-1)
plain+=acplain[acplain.length()-1];
if(plain.length()%2){
    plain+='Z';
}
int a=0,i=0,j=0;
while(a<key.length()){

    if(m.find(key[a])!=m.end())
    {
        a++;
        continue;
    }
    mat[i][j]=key[a];
    m[key[a]]={i,j};
    if(j+1>4){
        j=0;
        i++;
    }
    else{
        j++;
    }
    a++;
}

char ch='A';
while(ch<='Z'){
    if(ch=='I' || ch=='J'){
        if(m.find('J')!=m.end() || m.find('I')!=m.end())
        {
            ch++;
            continue;
        }
    }

    if(m.find(ch)!=m.end())
    {
        ch++;
        continue;
    }
    mat[i][j]=ch;
    m[ch]={i,j};
    if(j+1>4){
        j=0;
        i++;
    }
    else{
        j++;
    }
}

string cipher="";
```

```
   for(int i=0;i<plain.length()-1;i+=2){
      pair<int,int>c1=m[plain[i]];
      pair<int,int>c2=m[plain[i+1]];

      if(c1.second==c2.second){
         int row1=(c1.first+1)%5;
         int row2=(c2.first+1)%5;
         cipher+=mat[row1][c1.second];
         cipher+=mat[row2][c2.second];
      }

      else if(c1.first==c2.first){
         int col1=(c1.second+1)%5;
         int col2=(c2.second+1)%5;
         cipher+=mat[c1.first][col1];
         cipher+=mat[c1.first][col2];
      }
      else{
         cipher+=mat[c1.first][c2.second];
         cipher+=mat[c2.first][c1.second];
      }
   }
   for(int i=0;i<5;i++){
      for(int j=0;j<5;j++){
         cout<<mat[i][j]<<" ";
      }
      cout<<endl;

   }
   cout<<endl<<"Encrypted String"<<endl<<cipher;
return 0;}
```

OUTPUT:



```
Enter the key
PLAY
Enter the string
HARSH
P L A Y B
C D E F G
H I K M N
O Q R S T
U V W X Z

Encrypted String
KPSTNU
```

Q5. Implement Keyless and Keyed Cipher.

Theory: Transposition Cipher:

- A transposition cipher does not substitute one symbol for another (as in substitution cipher), but changes the location ofthese symbols.
- It reorders (jumbles) the given plain-text to give the cipher-text.
- They are of two types: Keyed and Keyless Transposition Cipher.

Keyless Transposition Cipher:

- In this cipher technique, the message is converted to ciphertextby either of two permutation techniques:

a. Text is written into a table column-by-column and is thentransmitted row-by-row.

b. Text is written into a table row-by-row and is then transmittedcolumn-by-column

Keyed Transposition cipher:

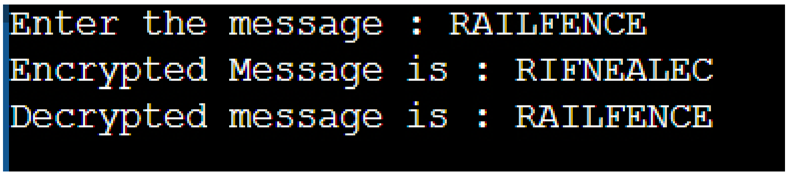
- In this approach, rather than permuting all the symbolstogether, we divide the entire plaintext into blocks of predetermined size and then permute each block independently.
- Suppose A wants to send a message to B "WE HAVE AN ATTACK". Both A and B agreed to had previously agreed ovedthe blocks size as 5.

Code:

```
#include <iostream>

using namespace std;

int main(){


        string message ;
        cout<<"Enter the message : ";
        cin>>message;
        string row1;
        string row2;
        for(int i=0;i<message.length();i++){
                if(i%2!=0){
                        row1+=message[i];
                }                                           }
                else{

                }
```

```
                        row2+=message[i];
        cout<<"Encrypted Message is : "<<row2<<row1<<endl;

        string cipher=row2+row1;

        int i=0,j=cipher.size()/2;
        if(cipher.size()%2)
        j++;

        cout<<"Decrypted message is : ";
        while(i<cipher.size()/2 && j<cipher.size()){
           cout<<cipher[i++]<<cipher[j++];
        }
        if(cipher.size()%2)
        cout<<cipher[i];

        return 0;
}
```

OUTPUT:



```
Enter the message : RAILFENCE
Encrypted Message is : RIFNEALEC
Decrypted message is : RAILFENCE
```

Q6 - Implement DES

Theory: **Data encryption standard (DES)** has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DESis a block cipher and encrypts data in blocks of size of 64
bits each, which means 64 bits of plain text go as the input toDES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

Let us now discuss the broad-level steps in DES.

- In the first step, the 64-bit plain text block is handedover to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves ofthe permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of theencryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block

The result of this process
produces 64-bit
ciphertext

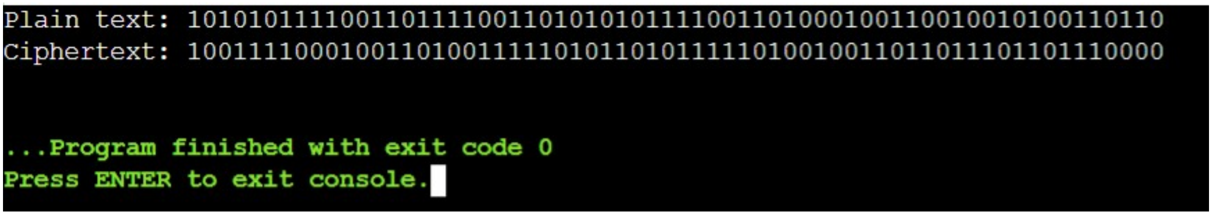
Code:

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

using namespace std;

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

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

int keyp[] = {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 keycompr[]={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 expansion[]={32, 1, 2, 3, 4, 5, 4, 5,
      6, 7, 8, 9, 8, 9, 10, 11,
      12, 13, 12, 13, 14, 15, 16, 17,
      16, 17, 18, 19, 20, 21, 20, 21,
      22, 23, 24, 25, 24, 25, 26, 27,
      28, 29, 28, 29, 30, 31, 32, 1};

int per[]={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};

int sbox[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}}};

string initialPerm(string s){
    string perm;
    for(int i=0;i<64;i++){
        perm+=s[arr[i]-1];
    }

    return perm;
}

string compress(string key){
    string key56;

    for(int i=0;i<56;i++){

        key56+=key[keyp[i]-1];
    }

    return key56;
}

string keycompress(string key){
    string key48;

    for(int i=0;i<48;i++){
        key48+=key[keycompr[i]-1];
    }

    return key48;
}

string expand(string str){
    string str48;

    for(int i=0;i<48;i++){
        str48+=str[expansion[i]-1];
    }

    return str48;
}

string DecimalToBinary(int num)
{
    string str;
      while(num){
      if(num & 1) // 1
      str+='1';
      else // 0
        str+='0';
      num>>=1; // Right Shift by 1
    }

    for(int i=str.length();i<4;i++){
        str+='0';
    }
      return str;
}

string bin_to_hex(string binary) {
    binary = string(binary.length() % 4 ? 4 - binary.length() % 4 : 0, '0') + binary;
    unordered_map<string, char> hex_dict = {
        {"0000", '0'}, {"0001", '1'}, {"0010", '2'}, {"0011", '3'},
        {"0100", '4'}, {"0101", '5'}, {"0110", '6'}, {"0111", '7'},
        {"1000", '8'}, {"1001", '9'}, {"1010", 'A'}, {"1011", 'B'},
        {"1100", 'C'}, {"1101", 'D'}, {"1110", 'E'}, {"1111", 'F'}
    };
    string hexadecimal;
    for (size_t i = 0; i < binary.length(); i += 4) {
        string group = binary.substr(i, 4);
        hexadecimal += hex_dict[group];
    }
    return hexadecimal;
}
int main()
{

    string
plaintext="0001001000110100010101101010101111001101000100110010010100110110";
    string
key="1010101010111011000010010001100000100111001101101100110011011101";


    //Initial Permutation
    string afterPerm=initialPerm(plaintext);
    string key56=compress(key);

    string leftb,rightb;
    for(int i=0;i<32;i++){
        leftb+=afterPerm[i];
    }
```

```
for(int i=32;i<64;i++){
    rightb+=afterPerm[i];
}

//Calculate C and D 28 bit each

string keyl,keyr;
for(int i=0;i<28;i++){
    keyl+=key56[i];
}

for(int i=28;i<56;i++){
    keyr+=key56[i];
}

vector<string>roundkey;
string rkeyl=keyl,rkeyr=keyr;

//Calculate round key of 56 bits and compressing to 48 bits
string leftpart=leftb,rightpart=rightb;
for(int i=0;i<16;i++){
    string rkeyl,rkeyr;

    for(int j=0;j<28;j++){
        rkeyl+=keyl[(j+shift[i])%keyl.size()];
        rkeyr+=keyr[(j+shift[i])%keyr.size()];
    }

    // rkeyl=rkeyl.substr(shift[i])+rkeyl.substr(0,shift[i]);

    // rkeyr=rkeyr.substr(shift[i])+rkeyr.substr(0,shift[i]);

    string round_key=rkeyl+rkeyr;

    string round_key48=keycompress(round_key);
    roundkey.push_back(round_key48);
    keyl=rkeyl,keyr=rkeyr;

}

string temprightb=rightb;
for(int i=0;i<16;i++){
    string rightbexpanded=expand(temprightb);
    string xored;
    for(int j=0;j<48;j++){
        if(rightbexpanded[j]==roundkey[i][j])
        xored+='0';
```

```
            else
            xored+='1';
        }

        //cout<<bin_to_hex(xored)<<endl;

        //COMPRESSION
        string str="";

        int x=0,y=5;
        for(int i=0;i<8;i++){
            int row=(xored[x]-'0')*2 + (xored[y]-'0');
            int col=(xored[x+1]-'0')*8 + (xored[x+2]-'0')*4 +(xored[x+3]-'0')*2 +(xored[x+4]-'0') ;
            str+=DecimalToBinary(sbox[i][row][col]);
            x+=6;
            y+=6;
        }
        xored=str;

        str="";
        for(int k=0;k<32;k++){
            str+=xored[per[k]-1];
        }
        xored=str;

        string temp=leftpart;
        leftpart=rightpart;
        str="";
        for(int j=0;j<32;j++){
            if(temp[j]==xored[j])
            {
                str+='0';
            }
            else str+='1';
        }

        rightpart=str;
        temprightb=rightpart;
        cout<<bin_to_hex(leftpart)<<" "<<bin_to_hex(rightpart)<<"
"<<bin_to_hex(roundkey[i])<<endl;

    }

    return 0;
}
```

OUTPUT:



```
Plain text: 1010101111001101111001101010101111001101000100110010010100110110
Ciphertext: 1001111000100110100111110101101011111010010011011011101101110000


...Program finished with exit code 0
Press ENTER to exit console.
```

Q7-. Implement AES Algorithm.

Theory: The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classifiedinformation.

AES is implemented in software and hardware throughout the worldto encrypt sensitive data. It is essential for government computer security, cybersecurity and electronic data protection.

The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the Data Encryption Standard (DES), which was startingto become vulnerable to brute-force attacks.

NIST stated that the newer, advanced encryption algorithm would beunclassified and must be "capable of protecting sensitive government information well into the [21st] century." It was intended to be easy to implement in hardware and software, as well as in restricted environments -- such as a smart card -- and offer decent defenses against various attack techniques.

AES was created for the U.S. government with additional voluntary, free use in public or private, commercial or noncommercial programsthat provide encryption services. However, nongovernmental organizations choosing to use AES are subject to limitations created by U.S. export control.

How AES encryption works

AES includes three block ciphers:

AES-128 uses a 128-bit key length to encrypt and decrypt a block ofmessages.

AES-192 uses a 192-bit key length to encrypt and decrypt a block ofmessages.

AES-256 uses a 256-bit key length to encrypt and decrypt a block ofmessages.

Each cipher encrypts and decrypts data in blocks of 128 bits usingcryptographic keys of 128, 192 and 256 bits, respectively.

Symmetric, also known as secret key, ciphers use the same key for encrypting and decrypting. The sender and the receiver must bothknow -- and use -- the same secret key.

The government classifies information in three categories: Confidential, Secret or Top Secret. All key lengths can be used toprotect the Confidential and Secret level. Top Secret informationrequires either 192- or 256-bit key lengths.

There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the inputplaintext to transform it into the final output of ciphertext

Encryption :

AES considers each block as a 16 byte (4 byte x 4 byte = 128) grid ina column major arrangement.

[b0 | b4 | b8 | b12 |

| b1 | b5 | b9 | b13 |

| b2 | b6 | b10| b14 |

| b3 | b7 | b11| b15]

Each round comprises of 4 steps :

- SubBytes
- ShiftRows
- MixColumns
- Add Round Key

Code:

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <sstream>




// Encryption: Forward Rijndael S-box
unsigned char s[256] =
{
	0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
	0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
	0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
	0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
	0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
	0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
	0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
	0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
	0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
	0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
	0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
	0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86,
0xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE,
0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0,
0x54, 0xBB, 0x16
};

// Encryption: Multiply by 2 for MixColumns
unsigned char mul2[] =
{

0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,

0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,

0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,

0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
        0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
        0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
};

// Encryption: Multiply by 3 for MixColumns
unsigned char mul3[] =
{
```

```
0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
        0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
        0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
};

// Used in KeyExpansion
unsigned char rcon[256] = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
0x91, 0x39,
        0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83,
0x1d, 0x3a,
        0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8,
        0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
0xfa, 0xef,
        0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc,
        0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1b,
	0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3,
	0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94,
	0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
0x10, 0x20,
	0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35,
	0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f,
	0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04,
	0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63,
	0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
0xd3, 0xbd,
	0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb, 0x8d
};

// Decryption: Inverse Rijndael S-box
unsigned char inv_s[256] =
{
	0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81,
0xF3, 0xD7, 0xFB,
	0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4,
0xDE, 0xE9, 0xCB,
	0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42,
0xFA, 0xC3, 0x4E,
	0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D,
0x8B, 0xD1, 0x25,
	0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D,
0x65, 0xB6, 0x92,
	0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7,
0x8D, 0x9D, 0x84,
	0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8,
0xB3, 0x45, 0x06,
	0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01,
0x13, 0x8A, 0x6B,
	0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0,
0xB4, 0xE6, 0x73,
	0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C,
0x75, 0xDF, 0x6E,
	0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA,
0x18, 0xBE, 0x1B,
	0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78,
0xCD, 0x5A, 0xF4,
        0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27,
0x80, 0xEC, 0x5F,
        0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93,
0xC9, 0x9C, 0xEF,
        0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83,
0x53, 0x99, 0x61,
        0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55,
0x21, 0x0C, 0x7D
};

// Decryption: Multiply by 9 for InverseMixColumns
unsigned char mul9[256] =
{

0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65,0x7e,0x77,

0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5,0xee,0xe7,

0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e,0x45,0x4c,

0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,0xd5,0xdc,

0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,0x08,0x01,

0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,0x98,0x91,

0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,0x33,0x3a,

0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,0xa3,0xaa,

0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,0x92,0x9b,

0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,0x02,0x0b,

0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,0xa9,0xa0,

0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,0x39,0x30,

0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,0xe4,0xed,

0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,0x74,0x7d,

0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,0xdf,0xd6,

0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,0x4f,0x46
};

// Decryption: Multiply by 11 for InverseMixColumns
```

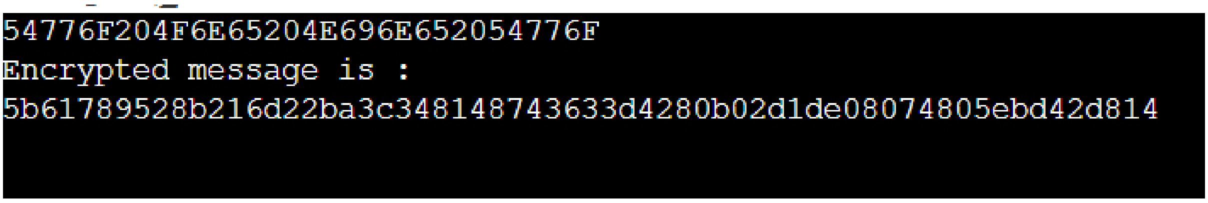
```
unsigned char mul11[256] =
{

0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04,0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05,0x18,0x13,

0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3
};

// Decryption: Multiply by 13 for InverseMixColumns
unsigned char mul13[256] =
{

0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,

0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,

0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,

0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,
0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,

0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,

0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,

0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,

0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,

0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,

0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,

0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,

0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,

0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,

0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,

0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d,0x9a,0x97
};

// Decryption: Multiply by 14 for InverseMixColumns
unsigned char mul14[256] =
{

0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46,0x54,0x5a,

0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6,0xb4,0xba,

0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d,0x8f,0x81,

0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d,0x6f,0x61,

0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb,0xf9,0xf7,

0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b,0x19,0x17,

0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30,0x22,0x2c,

0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0,0xc2,0xcc,

0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07,0x15,0x1b,
0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7,0xf5,0xfb,

0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc,0xce,0xc0,

0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c,0x2e,0x20,

0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa,0xb8,0xb6,

0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a,0x58,0x56,

0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71,0x63,0x6d,
    0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91,0x83,0x8d
};

// Auxiliary function for KeyExpansion
void KeyExpansionCore(unsigned char * in, unsigned char i) {
    // Rotate left by one byte: shift left
    unsigned char t = in[0];
    in[0] = in[1];
    in[1] = in[2];
    in[2] = in[3];
    in[3] = t;

    // S-box 4 bytes
    in[0] = s[in[0]];
    in[1] = s[in[1]];
    in[2] = s[in[2]];
    in[3] = s[in[3]];

    // RCon
    in[0] ^= rcon[i];
}

/* The main KeyExpansion function
 * Generates additional keys using the original key
 * Total of 11 128-bit keys generated, including the original
 * Keys are stored one after the other in expandedKeys
 */
void KeyExpansion(unsigned char inputKey[16], unsigned char expandedKeys[176]) {
    // The first 128 bits are the original key
    for (int i = 0; i < 16; i++) {
        expandedKeys[i] = inputKey[i];
    }

    int bytesGenerated = 16; // Bytes we've generated so far
    int rconIteration = 1; // Keeps track of rcon value
    unsigned char tmpCore[4]; // Temp storage for core
    while (bytesGenerated < 176) {
        /* Read 4 bytes for the core
        * They are the previously generated 4 bytes
        * Initially, these will be the final 4 bytes of the original key
        */
        for (int i = 0; i < 4; i++) {
            tmpCore[i] = expandedKeys[i + bytesGenerated - 4];
        }

        // Perform the core once for each 16 byte key
        if (bytesGenerated % 16 == 0) {
            KeyExpansionCore(tmpCore, rconIteration++);
        }

        for (unsigned char a = 0; a < 4; a++) {
            expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated -
16] ^ tmpCore[a];
            bytesGenerated++;
        }

    }
}
```

```
using namespace std;

/* Serves as the initial round during encryption
 * AddRoundKey is simply an XOR of a 128-bit block with the 128-bit key.
 */
void AddRoundKey(unsigned char * state, unsigned char * roundKey) {
    for (int i = 0; i < 16; i++) {
        state[i] ^= roundKey[i];
    }
}

/* Perform substitution to each of the 16 bytes
 * Uses S-box as lookup table
 */
void SubBytes(unsigned char * state) {
    for (int i = 0; i < 16; i++) {
        state[i] = s[state[i]];
    }
```

```
}

// Shift left, adds diffusion
void ShiftRows(unsigned char * state) {
        unsigned char tmp[16];

        /* Column 1 */
        tmp[0] = state[0];
        tmp[1] = state[5];
        tmp[2] = state[10];
        tmp[3] = state[15];

        /* Column 2 */
        tmp[4] = state[4];
        tmp[5] = state[9];
        tmp[6] = state[14];
        tmp[7] = state[3];

        /* Column 3 */
        tmp[8] = state[8];
        tmp[9] = state[13];
        tmp[10] = state[2];
        tmp[11] = state[7];

        /* Column 4 */
        tmp[12] = state[12];
        tmp[13] = state[1];
        tmp[14] = state[6];
        tmp[15] = state[11];

        for (int i = 0; i < 16; i++) {
                state[i] = tmp[i];
        }
}

 /* MixColumns uses mul2, mul3 look-up tables
  * Source of diffusion
  */
void MixColumns(unsigned char * state) {
        unsigned char tmp[16];

        tmp[0] = (unsigned char) mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3];
        tmp[1] = (unsigned char) state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3];
        tmp[2] = (unsigned char) state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]];
        tmp[3] = (unsigned char) mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]];

        tmp[4] = (unsigned char)mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7];
        tmp[5] = (unsigned char)state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7];
```

```
    tmp[6] = (unsigned char)state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]];
    tmp[7] = (unsigned char)mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]];

    tmp[8] = (unsigned char)mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11];
    tmp[9] = (unsigned char)state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11];
    tmp[10] = (unsigned char)state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]];
    tmp[11] = (unsigned char)mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]];

    tmp[12] = (unsigned char)mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15];
    tmp[13] = (unsigned char)state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15];
    tmp[14] = (unsigned char)state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]];
    tmp[15] = (unsigned char)mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]];

    for (int i = 0; i < 16; i++) {
        state[i] = tmp[i];
    }
}

/* Each round operates on 128 bits at a time
 * The number of rounds is defined in AESEncrypt()
 */
void Round(unsigned char * state, unsigned char * key) {
    SubBytes(state);
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(state, key);
}

 // Same as Round() except it doesn't mix columns
void FinalRound(unsigned char * state, unsigned char * key) {
    SubBytes(state);
    ShiftRows(state);
    AddRoundKey(state, key);
}

/* The AES encryption function
 * Organizes the confusion and diffusion steps into one function
 */
void AESEncrypt(unsigned char * message, unsigned char * expandedKey, unsigned char *
encryptedMessage) {
    unsigned char state[16]; // Stores the first 16 bytes of original message

    for (int i = 0; i < 16; i++) {
        state[i] = message[i];
    }

    int numberOfRounds = 9;
    AddRoundKey(state, expandedKey); // Initial round

    for (int i = 0; i < numberOfRounds; i++) {
        Round(state, expandedKey + (16 * (i+1)));
    }

    FinalRound(state, expandedKey + 160);

    // Copy encrypted state to buffer
    for (int i = 0; i < 16; i++) {
        encryptedMessage[i] = state[i];
    }
}

int main() {


    char message[1024]="54776F204F6E65204E696E652054776F";

//  cout << "Enter the message of 128 bit size: ";
    cout << message << endl;

    // Pad message to 16 bytes
    int originalLen = strlen((const char *)message);

    int paddedMessageLen = originalLen;

    if ((paddedMessageLen % 16) != 0) {
        paddedMessageLen = (paddedMessageLen / 16 + 1) * 16;
    }

    unsigned char * paddedMessage = new unsigned char[paddedMessageLen];
    for (int i = 0; i < paddedMessageLen; i++) {
        if (i >= originalLen) {
            paddedMessage[i] = 0;
        }
        else {
            paddedMessage[i] = message[i];
        }
    }

    unsigned char * encryptedMessage = new unsigned char[paddedMessageLen];

    string str="01 04 02 03 01 03 04 0A 09 0B 07 0F 0F 06 03 00";
 // this str is key


    istringstream hex_chars_stream(str);
```

```
        unsigned char key[16];
        int i = 0;
        unsigned int c;
        while (hex_chars_stream >> hex >> c)
        {
                key[i] = c;
                i++;
        }

        unsigned char expandedKey[176];

        KeyExpansion(key, expandedKey);

        for (int i = 0; i < paddedMessageLen; i += 16) { AESEncrypt(paddedMessage+i,
                expandedKey, encryptedMessage+i);
        }

        cout << "Encrypted message is :" << endl; for
        (int i = 0; i < paddedMessageLen; i++) {
                cout << hex << (int) encryptedMessage[i];
//              cout << " ";
        }

        cout << endl;


        return 0;
}
```



```
54776F204F6E65204E696E652054776F
Encrypted message is :
5b61789528b216d22ba3c348148743633d4280b02d1de08074805ebd42d814
```

Q8- Implement RSA Algorithm.

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

An example of asymmetric cryptography :

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

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

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

Let us learn the mechanism behind RSA algorithm : >>Generating Public Key :

Select two prime no's. Suppose **P = 53 and Q = 59.Now First part of the Public key : n = P*Q = 3127.**We also need a small exponent say **e :**

But e Must beAn

integer.

Not be a factor of n.

1 < e < Φ(n) [Φ(n) is discussed below],

Let us now consider it to be equal to 3.

Our Public Key is made of n and e

>> Generating Private Key : We need to calculate Φ(n) :Such that **Φ(n) = (P-1)(Q-1)**

so, Φ(n) = 3016

Now calculate Private Key, **d :**

d = (k*Φ(n) + 1) / e for some integer k

For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3)and Private Key(d = 2011)
Now we will encrypt **“HI”** :

Convert letters to numbers : H = 8 and I = 9Thus **Encrypted**

Data c = 89$^{\text{e mod n.}}$

Thus our Encrypted Data comes out to be 1394

Now we will decrypt **1394** :

Decrypted Data = c$^{\text{d mod n.}}$

Thus our Encrypted Data comes out to be 89

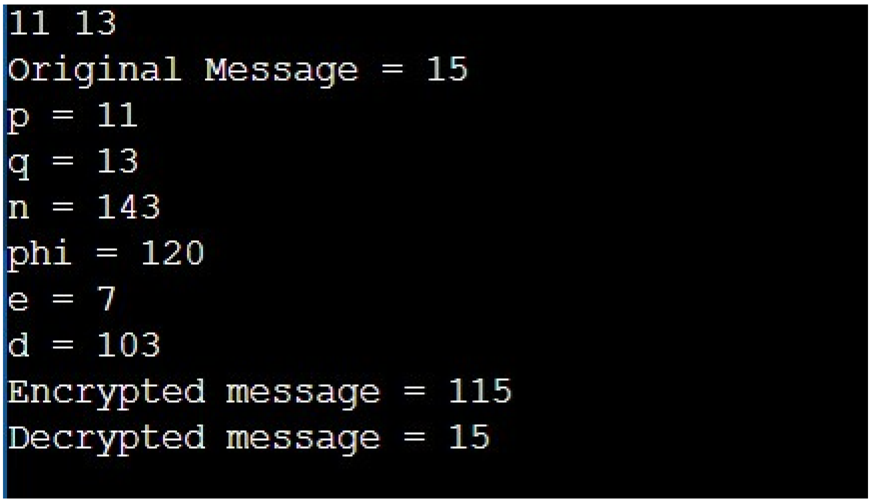
8 = H and I = 9 i.e. "HI".

Code:

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

int mul(int e, int phi){
  for(int i=1;i<phi;i++){
    if((i*e)%phi==1)
    return i;
  }
```

```
    return -1;
}

int main()
{
    int p,q;
    cin>>p>>q;
    int phi=(p-1)*(q-1),n=p*q,e=2;

    while(e<phi){
        if(_gcd(e,phi)==1)
        break;
        e++;
    }

    int d=mul(e,phi),message;
    cin>>message;
    int cipher=pow(message,e);
    cipher=cipher%n;
    int m=pow(cipher,d);
    m=m%n;
    cout<<"Cipher message "<<cipher<<endl;
    cout<<"Decrypted message "<<m;


    return 0;
}
```

OUTPUT:



```
11 13
Original Message = 15
p = 11
q = 13
n = 143
phi = 120
e = 7
d = 103
Encrypted message = 115
Decrypted message = 15
```

Q9- Implement Public-Key Cryptography El Gamal

Theory: ElGamal encryption is a public-key cryptosystem. It uses asymmetrickey encryption for communicating between two parties and encrypting the message.

This cryptosystem is based on the difficulty of finding **discrete logarithm** in a cyclic group that is even if we know g^a and g^k, it isextremely difficult to compute g^{ak}.

Idea of ElGamal cryptosystem

Suppose Alice wants to communicate with Bob.

1. Bob generates public and private keys:
   - Bob chooses a very large number **q** and a cyclic group $\mathbf{F}_q$.
   - From the cyclic group $\mathbf{F}_q$, he choose any element **g** andan element **a** such that gcd(a, q) = 1.
   - Then he computes $h = g^a$.
   - Bob publishes **F**, $\mathbf{h = g^a}$, **q**, and **g** as his public key andretains **a** as private key.
2. Alice encrypts data using Bob's public key :
   - Alice selects an element **k** from cyclic group **F** such that gcd(k, q) = 1.
   - Then she computes $p = g^k$ and $s = h^k = g^{ak.}$
   - She multiples s with M.
   - Then she sends $(p, M*s) = (g^k, M*s)$.
3. Bob decrypts the message :
   - Bob calculates $s' = p^a = g^{ak}$.

- He divides M*s by s′ to obtain M as s = s

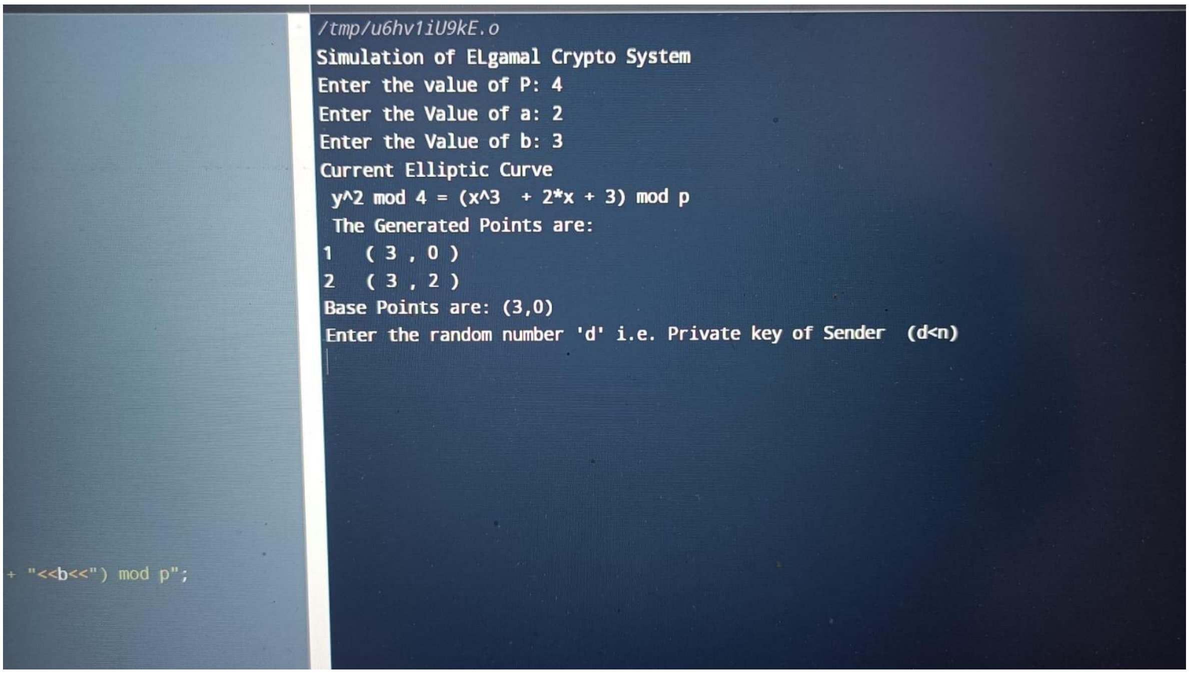
Code:

```
#include<iostream>
#include<math.h>
#include<cstdlib>
#include<vector> using
namespace std;

int main()
{
        int n,p; int LHS[2][n],RHS[2][n],a,b,i,j;vector <int>
        arr_x;
        vector <int> arr_y;
        cout<<"Simulation of ELgamal Crypto System";cout<<"\n Enter
        the value of P: ";
        cin>>p;n
        = p;
        cout<<"\n Enter the Value of a: ";cin>>a;
        cout<<"\n Enter the Value of b: ";cin>>b;
        cout<<"\n Current Elliptic Curve";
        cout<<"\n y^2 mod "<<p<<" = (x^3  + "<<a<<"*x + "<<b<<")mod p";
```

```
    for(int i=0; i<n; i++)
    {
        LHS[0][i] = i;
        RHS[0][i] = i;
        LHS[1][i] = ((i*i*i) + a*i + b) % p;
        RHS[1][i] = (i*i) % p;
    }



int in_c = 0;
    for(i = 0; i < n; i ++)
    {

        for(j = 0; j < n; j++)
        {
            if(LHS[1][i] == RHS[1][j])
            {
                        in_c++; arr_x.push_back(LHS[0][i]);
                    arr_y.push_back(RHS[0][j]);
            }
        }
    }

    cout<<"\n The Generated Points are: \n";for(i =0; i <
    in_c; i++)
    {
            cout<<i+1<<"\t( "<<arr_x[i]<<" , "<<arr_y[i]<<" )"<<"\n";
    }
    cout<<"Base Points are: ("<<arr_x[0]<<","<<arr_y[0]<<")"<<"\n";
    int k,d,M;
    cout<<"Enter the random number 'd' i.e. Private key of Sender(d<n)\n";
    cin>>d;
    int Qx=d*arr_x[0];int
    Qy=d*arr_y[0];
cout<<"Enter the random number 'k' (k<n)\n";cin>>k;

cout<<"Enter the message to be sent:\n";cin>>M;
    cout<<"The message to be sent is:\n"<<M<<"\n";int c1x=k*arr_x[0];
```

```
        int c1y=k*arr_y[0];
        cout<<"Value of C1: ("<<c1x<<","<<c1y<<")"<<"\n";

        int c2x=k*Qx+M;int
        c2y=k*Qy+M;
        cout<<"Value of C2: ("<<c2x<<","<<c2y<<")"<<"\n";


        cout<<"\nThe message received is:\n";int Mx=c2x-
        d*c1x;
        int My=c2y-d*c1y;
        cout<<Mx;
}
```

Output:



```
+ "<<b<<") mod p";
```

```
/tmp/u6hv1iU9kE.o
Simulation of ELgamal Crypto System
Enter the value of P: 4
Enter the Value of a: 2
Enter the Value of b: 3
Current Elliptic Curve
 y^2 mod 4 = (x^3  + 2*x + 3) mod p
 The Generated Points are:
1   ( 3 , 0 )
2   ( 3 , 2 )
Base Points are: (3,0)
Enter the random number 'd' i.e. Private key of Sender  (d<n)
```

Q.10 Implement Rabin Algorithm

Rabin Cryptosystem is an public-key cryptosystem invented by Michael Rabin. It uses asymmetric key encryption for communicatingbetween two parties and encrypting the message.

The security of Rabin cryptosystem is related to the difficulty of factorization. It has the advantage over the others that the problem on which it banks has proved to be hard as integer factorization. It has the disadvantage also, that each output of the Rabin function can be generated by any of four possible inputs. if each output is a ciphertext, extra complexity is required on decryption to identify which of the four possible inputs was the true plaintext.

<u>Steps in Rabin cryptosystem</u>

Key generation

1. Generate two very large prime numbers, p and q, whichsatisfies the condition
   $p \neq q \rightarrow p \equiv q \equiv 3 \pmod 4$
   For example:
   p=139 and q=191
2. Calculate the value of nn = p.q
3. Publish n as public key and save p and q as private key

Encryption

1. Get the public key n.
2. Convert the message to ASCII value. Then convert it to binary and extend the binary value with itself, and change the binaryvalue back to decimal m.

3. Encrypt with the formula:$C = m^2 \mod n$

4. Send C to recipient.

Decryption

1. Accept C from sender.

2. Specify a and b with Extended Euclidean GCD such that, $a.p + b.q = 1$

3. Compute r and s using following formula:$r = C^{(p+1)/4} \mod p$
   $s = C^{(q+1)/4} \mod q$

4. Now, calculate X and Y using following formula:$X = (a.p.r + b.q.s) \mod p$
   $Y = (a.p.r – b.q.s) \mod q$

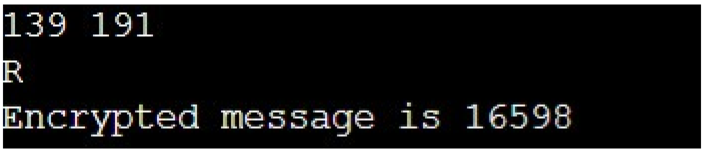
5. The four roots are, m1=X, m2=-X, m3=Y, m4=-Y
   Now, Convert them to binary and divide them all in half.

6. Determine in which the left and right half are same. Keep that binary's one half and convert it to decimal m. Get the ASCII character for the decimal value m. The resultant character givesthe correct message sent by sender.

Code:

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

using namespace std;

string DecimalToBinary(int num)
{
    string str;
      while(num){
      if(num & 1) // 1
      str+='1';
      else // 0
        str+='0';
      num>>=1; // Right Shift by 1
    }
    reverse(str.begin(),str.end());
    return str;
}

int binaryToDecimal(string n)
{
    string num = n;
    int dec_value = 0;
    int base = 1;

    int len = num.length();
    for (int i = len - 1; i >= 0; i--) {
        if (num[i] == '1')
            dec_value += base;
        base = base * 2;
    }

    return dec_value;
}

pair<int,int> mul(int a,int b){
    int q;
    int r1=max(a,b);
    int  r2=min(a,b);
    int r;
    int    t,t1=0,t2=1;
    int s1=1,s2=0,s;

    q=r1/r2;
    r=r1-q*r2;
    while(r2!=0) {

    t=t1-q*t2;
    s=s1-q*s2;
    r1=r2;
    r2=r;
    r=t1;
    t1=t2;
    t2=t;
    s1=s2;
    s2=s;
    if(r2!=0)q=r1/r2;

    r=r1-q*r2;
    }

    return {s1,t1};

}

int main()
{
    int p,q;
    cin>>p>>q;

    char message;
    cin>>message;

    int n=p*q;
    int ascii=message;
    string bin=DecimalToBinary(ascii);

    bin=bin+bin;
    int redundant=binaryToDecimal(bin);
    int encrypt=pow(redundant,2);
    encrypt=encrypt%n;
    cout<<"Encrypted message is " <<encrypt<<endl;


    int r,s;
    r=pow(encrypt,(p+1)/4);
    r=r%p;
    s=pow(encrypt,(q+1)/4);
    s=s%q;
    cout<<r<<" "<<s<<endl;

    pair<int,int>pr=mul(p,q);
    // cout<<pr.first<<" "<<pr.second<<endl;
    int x=p>q?pr.first:pr.second;
    int y=p<q?pr.first:pr.second;

    cout<<x*p+y*q;

    int X=(x*p*r+y*q*s)%p;
    int Y=(x*p*r-y*q*s)%q;

    int m1=abs(X),m2=abs(Y);

    string first=DecimalToBinary(m1); string
    second=DecimalToBinary(m2);

    if(first.substr(0,first.length()/2-1)==first.substr(first.length()/2))
    cout<<binaryToDecimal(first.substr(first.length()/2));

    else
    cout<<binaryToDecimal(second.substr(second.length()/2));

    return 0;
}
```

OUTPUT:



```
139 191
R
Encrypted message is 16598
```

Q11- Implement Public-Key Cryptography Elliptic Curve Cryptography “ECC”

Elliptic Curve Cryptography (ECC) is a key-based technique for encrypting data. ECC focuses on pairs of public and private keys fordecryption and encryption of web traffic.

ECC is frequently discussed in the context of the Rivest–Shamir–Adleman (RSA) cryptographic algorithm. RSA achieves one-way encryption of things like emails, data, and software using prime factorization.

What is Elliptic Curve Cryptography?

ECC, an alternative technique to RSA, is a powerful cryptography approach. It generates security between key pairs for public key encryption by using the mathematics of elliptic curves.

RSA does something similar with prime numbers instead of elliptic curves, but ECC has gradually been growing in popularity recently due to its smaller key size and ability to maintain security. This trendwill probably continue as the demand on devices to remain secure increases due to the size of keys growing, drawing on scarce mobile resources. This is why it is so important to understand elliptic curve cryptography in context.

In contrast to RSA, ECC bases its approach to public key cryptographic systems on how elliptic curves are structured algebraically over finite fields. Therefore, ECC creates keys that aremore difficult, mathematically, to crack. For this reason, ECC is considered to be the next generation implementation of public keycryptography and more secure than RSA.

It also makes sense to adopt ECC to maintain high levels of both performance and security. That’s because ECC is increasingly in wideruse as websites strive for greater online security in customer data and greater mobile optimization, simultaneously. More sites using ECC to secure data means a greater need for this kind of quick guideto elliptic curve cryptography.

An elliptic curve for current ECC purposes is a plane curve over a finite field which is made up of the points satisfying the equation:$y^2=x^3 + ax + b$.

In this elliptic curve cryptography example, any point on the curve can be mirrored over the x-axis and the curve will stay the same. Anynon-vertical line will intersect the curve in three places or fewer.

Elliptic Curve Cryptography vs RSA

The difference in size to security yield between RSA and ECC encryption keys is notable. The table below shows the sizes of keys needed to provide the same level of security. In other words, an elliptic curve cryptography key of 384 bit achieves the same level ofsecurity as an RSA of 7680 bit.

RSA Key Length (bit)1024

2048
3072
7680
15360

ECC Key Length (bit)160

224
256
384
521

There is no linear relationship between the sizes of ECC keys and RSAkeys. That is, an RSA key size that is twice as big does not translate into an ECC key size that’s doubled. This compelling difference shows that ECC key generation and signing are substantially quicker than forRSA, and also that ECC uses less memory than does RSA.

Also, unlike in RSA, where both are integers, in ECC the private andpublic keys are not equally exchangeable. Instead, in ECC the publickey is a point on the curve, while the private key is still an integer.

A quick comparison of the advantages and disadvantages of ECC andRSA algorithms looks like this:

ECC features smaller ciphertexts, keys, and signatures, and faster generation of keys and signatures. Its decryption and encryption speeds are moderately fast. ECC enables lower latency than inverse throughout by computing signatures in two stages. ECC features strong protocols for authenticated key exchange and support for thetech is strong.

The main disadvantage of ECC is that it isn't easy to securely implement. Compared to RSA, which is much simpler on both the verification and encryption sides, ECC is a steeper learning curve anda bit slower for accumulating actionable results.

However, the disadvantages of RSA catch up with you soon. Key generation is slow with RSA, and so is decryption and signing, whicharen't always that easy to implement securely.

Code:

```
#include
<stdio.h>
#include
<stdlib.h>
#include
<math.h>

int Enc[4]={0,0,0,0};
int a=3;
int
b=20;int
p=19;
int
points[1000][2];int
PrivKey=11;
int
PubKey[2]={0,0};
int Rm = 11;
int Pbase[2]={0,0};

int * sclr_mult(int k,int
point[2]);int * add(int A[2],int
B[2]);
int inverse(int num);
int * encode(int m,int Pb[2],int Rm,int Pbase[2]);
int * genKey(int X,int P[2]);
int decode(int Enc[4],int
PrivKey);void generate();

int main()
{
   int *temp;
   printf("Simulation of ECC Cryptography
   System");generate();
   Pbase[0]=points[5][0];
   Pbase[1]=points[5][1];
   temp=genKey(PrivKey,Pba
   se);PubKey[0]=*temp;
   PubKey[1]=*(temp+1);
   printf("\n\n The Public Keys are (%d,%d)\n",PubKey[0],PubKey[1]);

   int message[2];
   message[0]=points[5][
   0];
   message[1]=points[5][
   1];
   printf("The Message Points are (%d,%d)\n",message[0],message[1]);

   int P[2];
   temp=sclr_mult(Rm,Pbas
   e);P[0]=*temp;
   P[1]=*(temp+1);
   int Q[2];
   temp=sclr_mult(Rm,PubKe
   y);Q[0]=*temp;
   Q[1]=*(temp+1);
```

```
    int R[2];
    temp=add(message,Q)
    ;R[0]=*temp;
    R[1]=*(temp+1);

    printf("The Encryption Points are [(%d,%d),(%d,%d)]\n",P[0],P[1],R[0],R[1]);

    temp=sclr_mult(PrivKey,P
    );int O[2];
    O[0]=*temp;
    O[1]=p-
    *(temp+1);

    temp=add(R,O
    ); O[0]=*temp;
    O[1]=*(temp+1)
    ;
    printf("The Message Points are
    (%d,%d)\n",O[0],O[1]);return 0;
}

int * sclr_mult(int k,int P[2])
{
    int *temp,i;
    int *Q =
    calloc(2,sizeof(int));
    Q[0]=0;
    Q[1]=0;
    for(i=31;i>=0;i--)
    {
        if((k>>i)&1
            )break;
```

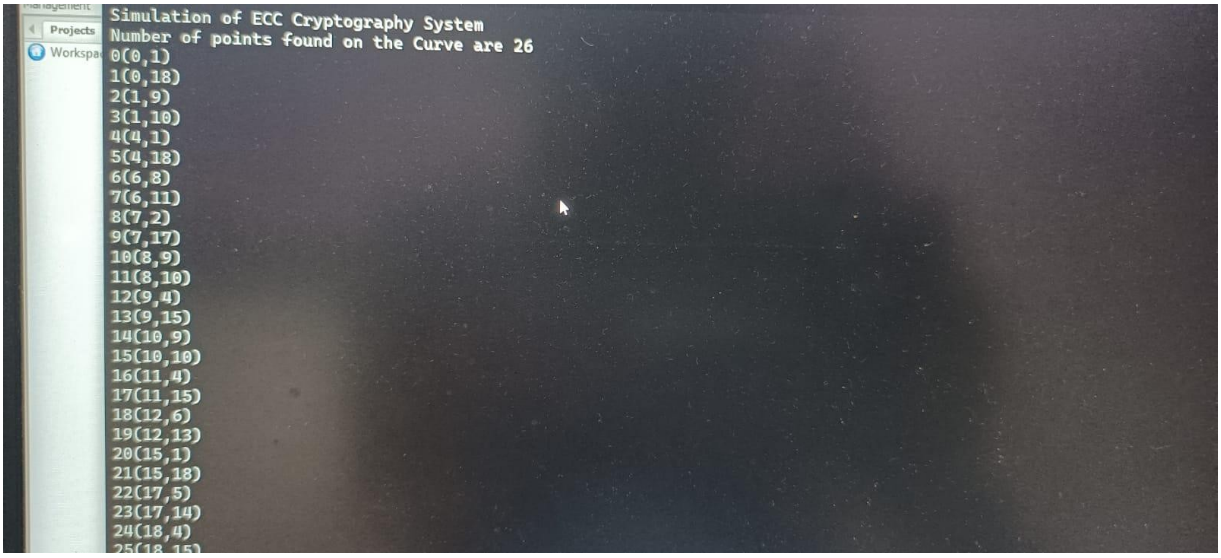
```
  }
  for(int j=0;j<=i;j++)
  {
    if((k>>j)&1)
    {
      temp=add(Q,P)
      ; Q[0]=*temp;
      Q[1]=*(temp+1)
      ;
    }
    temp=add(P,P);
    P[0]=*temp;
    P[1]=*(temp+1);
  }
  return Q;
}

int * add(int A[2],int B[2])
{
  int *C =
  calloc(2,sizeof(int));int
  x=0;
  if (A[0]==0 || A[1]==0)
  {
    return B;
  }
  if (B[0]==0 || B[1]==0)
  {
    return A;
  }
  if (A[1]==(p-B[1]))
```

```
   {
      return C;
   }
   if ((A[0]==B[0]) && (A[1]==B[1]))
   {
      x=((3*(A[0]*A[0]))+a)*inverse(2*A[1]);
      C[0]=((x*x)-(2*A[0]))%p;
      C[1]=((x*(A[0]-C[0]))-A[1])%p;
      }
   else
   {
      x=(B[1]-A[1])*inverse(B[0]-A[0]);
      C[0]=((x*x)-(A[0]+B[0]))%p;
      C[1]=((x*(A[0]-C[0]))-
   A[1])%p;}if (C[0]<0)
      C[0]=p+C[0];
   if (C[1]<0)
      C[1]=p+C[1];
   return C;
}
int inverse(int num)
{
   int i=1;
   if (num<0)
      num=p+nu
      m;
   for (i=1;i<p;i++)
   {
      if(((num*i)%p)==1)
         break;
```

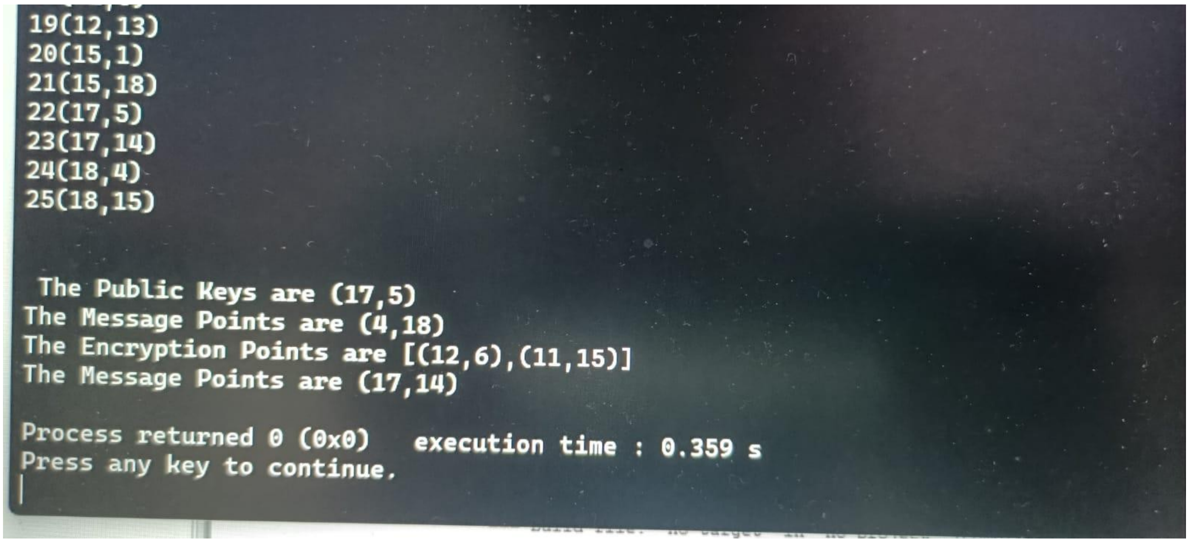
```
 }
 return i;
}

void generate()
{
 int rhs,lhs,i=0;
 for(int
 x=0;x<p;x++)
 {
  rhs=((x*x*x)+(a*x)+b)%p;
  for(int y=0;y<p;y++)
  {
   lhs=(y*y)%
   p;if
   (lhs==rhs)
   {
    points[i][0]=x;
    points[i][1]=y
    ;i+=1;
   }
  }
 }
 printf("\nNumber of points found on the Curve are %d
 \n",i);for(int k=0;k<i;k++)
 {
  printf("%d(%d,%d)\n",(k),points[k][0],points[k][1]);
 }
}

int * genKey(int X,int P[2])
```

```
{
   int *temp;
   int *Q =
   calloc(2,sizeof(int));
   temp=sclr_mult(X,P);
   Q[0]=*temp;
   Q[1]=*(temp+1);
   return Q;
}
```

Output:



```
Simulation of ECC Cryptography System
Number of points found on the Curve are 26
0(0,1)
1(0,18)
2(1,9)
3(1,10)
4(4,1)
5(4,18)
6(6,8)
7(6,11)
8(7,2)
9(7,17)
10(8,9)
11(8,10)
12(9,4)
13(9,15)
14(10,9)
15(10,10)
16(11,4)
17(11,15)
18(12,6)
19(12,13)
20(15,1)
21(15,18)
22(17,5)
23(17,14)
24(18,4)
25(18,15)
```



```
19(12,13)
20(15,1)
21(15,18)
22(17,5)
23(17,14)
24(18,4)
25(18,15)


 The Public Keys are (17,5)
The Message Points are (4,18)
The Encryption Points are [(12,6),(11,15)]
The Message Points are (17,14)

Process returned 0 (0x0)   execution time : 0.359 s
Press any key to continue.
```

Q12- Implement Secure Hash Algorithm “SHA-512”

SHA-512 is a hashing algorithm that performs a hashing function onsome data given to it.

Hashing algorithms are used in many things such as internet security,digital certificates and even blockchains. Since hashing algorithms play such a vital role in digital security and cryptography, this is an easy-to-understand walkthrough, with some basic and simple maths along with some diagrams, for a hashing algorithm called SHA-512.
It’s part of a group of hashing algorithms called SHA-2 which includesSHA-256 as well which is used in the bitcoin blockchain for hashing.

Before starting with an explanation of SHA-512, I think it would be
useful to have a basic idea of what a hashing function’s features are.

Hashing Functions

Hashing functions take some data as input and produce an output(called hash digest) of fixed length for that input data. This outputshould, however, satisfy some conditions to be useful.

1. Uniform distribution: Since the length of the output hash digestis of a fixed length and the input size may vary, it is apparent that there are going to be some output values that can be obtained for different input values. Even though this is the case, the hash function should be such that for any input value,each possible output value should be equally likely. That is to say that every possible output has the same likelihood to be produced for any given input value.

2. Fixed Length: This is should be quite self-explanatory. The output values should all be of a fixed length. So, for example, ahashing function could have an output size of 20 characters or 12 characters, etc. SHA-512 has an output size of 512 bits.

3. Collision resistance: Simply speaking, this means that there aren't any or rather it is not feasible to find two distinct inputsto the hash function that result in the same output (hash digest).

That's a simple introduction about hash functions. Now let's look at
SHA-512.

Hashing Algorithm — SHA-512

So, SHA-512 does its work in a few stages. These stages go as follows:

1. Input formatting
2. Hash buffer initialization
3. Message Processing
4. Output

Let's look at these one-by-one.

1. Input Formatting:

SHA-512 can't actually hash a message input of any size, i.e. it has aninput size limit. This limit is imposed by its very structure as you maysee further on. The entire formatted mesage has basically three parts: the original message, padding bits, size of original message.
And this should all have a combined size of a whole multiple of 1024bits. This is because the formatted message will be processed as blocks of 1024 bits each, so each bock should have 1024 bits to workwith.

<pic: original message>

L bits
Message

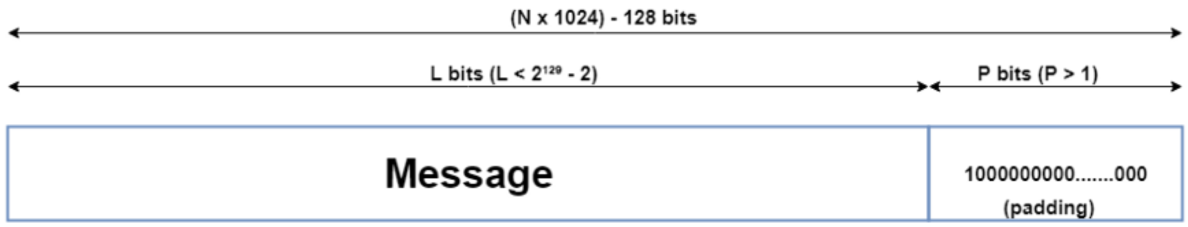
Original message

Padding bits

The input message is taken and some padding bits are appended to itin order to get it to the desired length. The bits that are used for padding are simply ‘0’ bits with a leading ‘1’ (100000…000). Also, according to the algorithm, padding *needs* to be done, even if it is by one bit. So a single padding bit would only be a ‘1’.

The total size should be equal to 128 bits short of a multiple of 1024since the goal is to have the formatted message size as a multiple of1024 bits (N x 1024).

<pic: msg + pad>



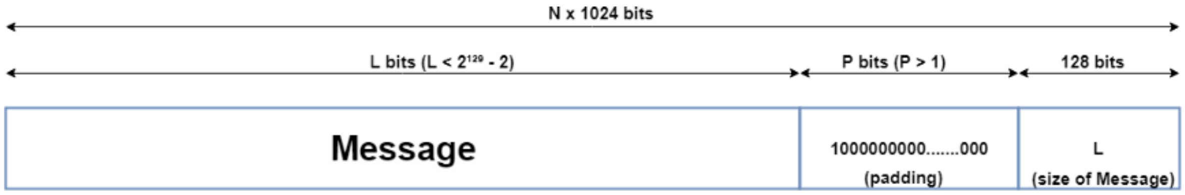
Message with padding

Padding size

After this, the size of the original message given to the algorithm is appended. This size value needs to be represented in 128 bits and isthe only reason that the SHA-512 has a limitation for its input message.

Since the size of the original message needs to be represented in 128bits and the largest number that can be represented using 128 bits is(2^{12}□-1), the message size can be at most (2^{12}□-1) bits; and also taking into consideration the necessary single padding bit, the maximum size for the original message would then be (2^{12}□-2). Even though thislimit exists, it doesn’t actually cause a problem since the actual limit is so high (2^{12}□-2 = 340,282,366,920,938,463,463,374,607,431,768,211,454 bits).

<pic: msg + pad +size>



Message with padding and size

Now that the padding bits and the size of the message have been appended, we are left with the completely formatted input for theSHA-512 algorithm.

Formatted Message

2. Hash buffer initialization:

The algorithm works in a way where it processes each block of 1024 bits from the message using the result from the previous block. Now,this poses a problem for the first 1024 bit block which can't use the result from any previous processing. This problem can be solved by using a default value to be used for the first block in order to start offthe process. (Have a look at the second-last diagram).

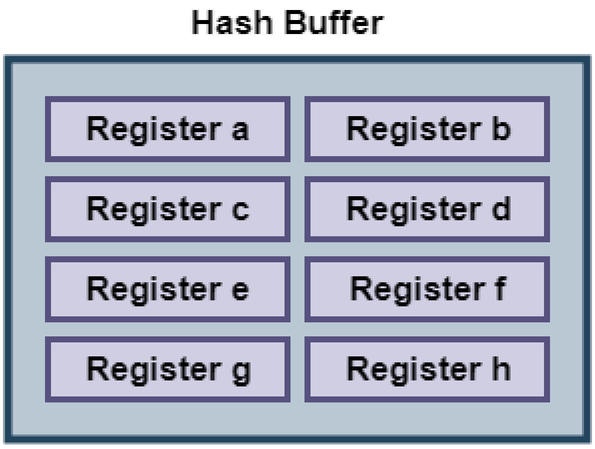
Since each intermediate result needs to be used in processing the next block, it needs to be stored somewhere for later use. This wouldbe done by the *hash buffer*, this would also then hold the final hash digest of the entire processing phase of SHA-512 as the last of these

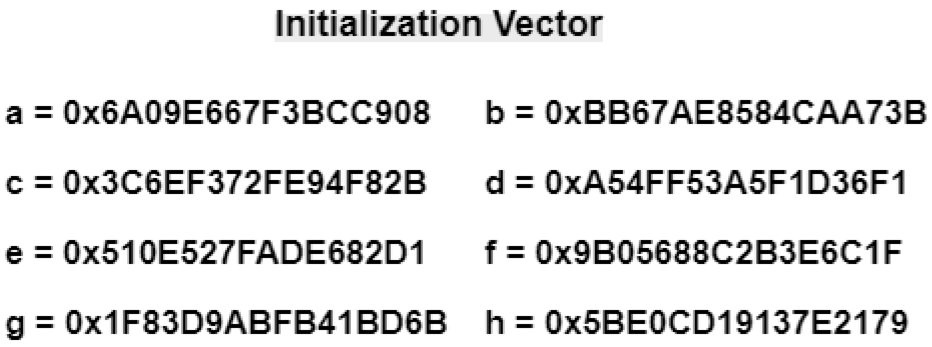
'intermediate' results.

So, the default values used for starting off the chain processing of each 1024 bit block are also stored into the hash buffer at the start of processing. The actual value used is of little consequence, but forthose interested, the values used are obtained by taking the first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19). These values are called the Initial Vectors (IV).

Why 8 prime numbers instead of 9? Because the hash buffer actuallyconsists of 8 subparts (registers) for storing them.

<pic: IV>





Initialization Vector

a = 0x6A09E667F3BCC908 b = 0xBB67AE8584CAA73B

c = 0x3C6EF372FE94F82B d = 0xA54FF53A5F1D36F1

e = 0x510E527FADE682D1 f = 0x9B05688C2B3E6C1F

g = 0x1F83D9ABFB41BD6B h = 0x5BE0CD19137E2179

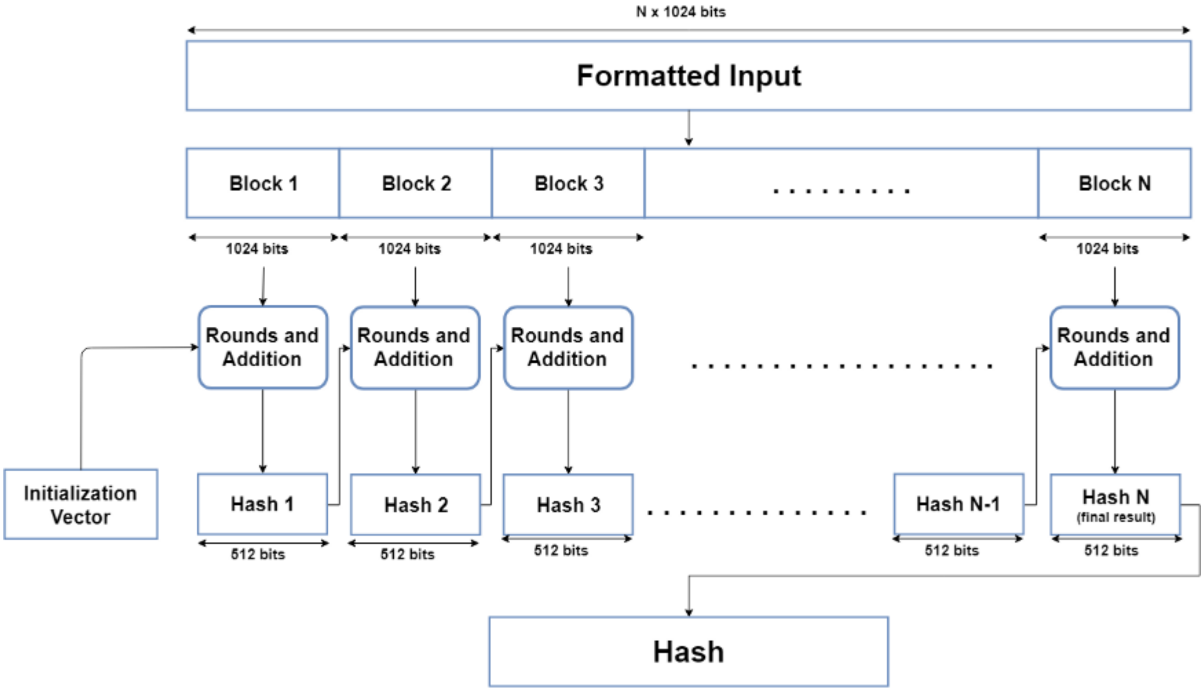
Hash buffer and Initialization Vector values

3. Message Processing:

Message processing is done upon the formatted input by taking oneblock of 1024 bits at a time. The actual processing takes place by using two things: The 1024 bit block, and the result from the previous processing.

This part of the SHA-512 algorithm consists of several ‘Rounds’ and an addition operation.

<pic: Formatted input 1024 bit blocks;F(M.n ,H.n-1)=H.n>



William Stallings, Cryptography and Network Security — Principlesand Practise (Seventh Edition) referred for diagram

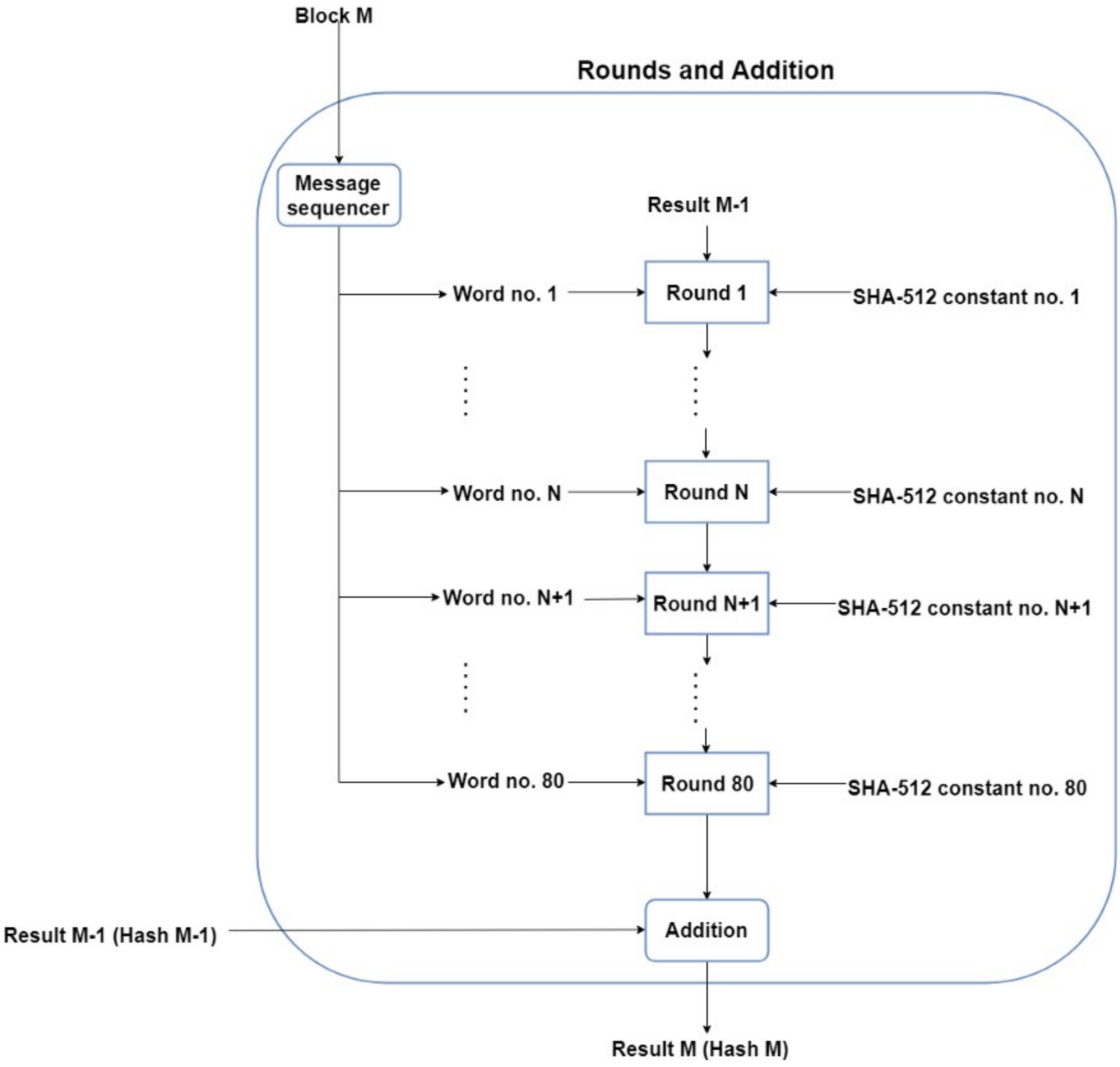
So, the Message block (1024 bit) is expanded out into ‘Words’ usinga ‘message sequencer’. Eighty Words to be precise, each of them having a size of 64 bits.

Rounds

The main part of the message processing phase may be considered to be the Rounds. Each round takes 3 things: one Word, the output of the previous Round, and a SHA-512 constant. The first Round doesn’t have a previous Round whose output it can use, so it uses the final output from the previous message processing phase for theprevious block of 1024 bits. For the first Round of the first block (1024 bits) of the formatted input, the Initial Vector (IV) is used.

SHA-512 constants are predetermined values, each of whom is usedfor each Round in the message processing phase. Again, these aren’tvery important, but for those interested, they are the first 64 bits from the fractional part of the cube roots of the first 80 prime numbers.Why 80? Because there are 80 Rounds and each of themneeds one of these constants.

Once the Round function takes these 3 things, it processes them andgives an output of 512 bits. This is repeated for 80 Rounds. After the80th Round, its output is simply added to the result of the previous message processing phase to get the final result for this iteration of message processing.



4. Output:

After every block of 1024 bits goes through the message processingphase, i.e. the last iteration of the phase, we get the final 512 bit Hash value of our original message. So, the intermediate results are all used from each block for processing the next block. And when thefinal 1024 bit block has finished being processed, we have with us the final result of the SHA-512 algorithm for our original message.

Thus, we obtain the final hash value from our original message. The SHA-512 is part of a group of hashing algorithms that are very similarin how they work, called SHA-2. Algorithms such as SHA-256 and SHA-384 are a part of this group alongside SHA-512. SHA-256 is also used in the Bitcoin blockchain as the designated hash function.

That’s a brief overview of how the SHA-512 hashing algorithm works.I intend to go into further detail about what makes the hash functions practically irreversible (one-way) and how this is helpful fordigital security.

Code:

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

typedef unsigned long long int int64;


int64 Message[80];


// Stores the hexadecimal values for

// calculating hash

valuesconst int64

Constants[80]

        = { 0x428a2f98d728ae22, 0x7137449123ef65cd,

                0xb5c0fbcfec4d3b2f,

                0xe9b5dba58189dbbc,

                0x3956c25bf348b538,

                0x59f111f1b605d019,

                0x923f82a4af194f9b,

                0xab1c5ed5da6d8118,

                0xd807aa98a3030242,

                0x12835b0145706fbe,

                0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,
                0x72be5d74f27b896f,
                0x80deb1fe3b1696b1,
                0x9bdc06a725c71235,
                0xc19bf174cf692694,
                0xe49b69c19ef14ad2,
                0xefbe4786384f25e3,
                0x0fc19dc68b8cd5b5,
                0x240ca1cc77ac9c65,
                0x2de92c6f592b0275,
                0x4a7484aa6ea6e483,
                0x5cb0a9dcbd41fbd4,
                0x76f988da831153b5,
                0x983e5152ee66dfab,
                0xa831c66d2db43210,
                0xb00327c898fb213f,
                0xbf597fc7beef0ee4,
                0xc6e00bf33da88fc2,
                0xd5a79147930aa725,
                0x06ca6351e003826f,
                0x142929670a0e6e70,
                0x27b70a8546d22ffc,
                0x2e1b21385c26c926,
                0x4d2c6dfc5ac42aed,
                0x53380d139d95b3df,
                0x650a73548baf63de,
                0x766a0abb3c77b2a8,
                0x81c2c92e47edaee6,
                0x92722c851482353b,
                0xa2bfe8a14cf10364,
                0xa81a664bbc423001,
                0xc24b8b70d0f89791,
                0xc76c51a30654be30,
                0xd192e819d6ef5218,
                0xd69906245565a910,
                0xf40e35855771202a, 0x106aa07032bbd1b8,
                0x19a4c116b8d2d0c8,
                0x1e376c085141ab53,
                0x2748774cdf8eeb99,
                0x34b0bcb5e19b48a8,
                0x391c0cb3c5c95a63,
                0x4ed8aa4ae3418acb,
                0x5b9cca4f7763e373,
                0x682e6ff3d6b2b8a3,
                0x748f82ee5defb2fc,
                0x78a5636f43172f60,
                0x84c87814a1f0ab72,
                0x8cc702081a6439ec,
                0x90befffa23631e28,
                0xa4506cebde82bde9,
                0xbef9a3f7b2c67915,
                0xc67178f2e372532b,
                0xca273eceea26619c,
                0xd186b8c721c0c207,
                0xeada7dd6cde0eb1e,
                0xf57d4f7fee6ed178,
                0x06f067aa72176fba,
                0x0a637dc5a2c898a6,
                0x113f9804bef90dae,
                0x1b710b35131c471b,
                0x28db77f523047d84,
                0x32caab7b40c72493,
                0x3c9ebe0a15c9bebc,
                0x431d67c49c100d4c,
                0x4cc5d4becb3e42b6,
                0x597f299cfc657e2a,
                0x5fcb6fab3ad6faec,
                0x6c44198c4a475817 };

// to hexa-decimal
valuestring
gethex(string bin)
{
        if (bin == "0000")
                return "0";
        if (bin == "0001")
                return "1";
        if (bin == "0010")
                return "2";
        if (bin == "0011")
                return "3";
        if (bin == "0100")
                return "4";
        if (bin == "0101")
                return "5";
        if (bin == "0110")
                return "6";
        if (bin == "0111")
                return "7";
        if (bin == "1000")
                return "8";
        if (bin == "1001")
```

```
                return "9";
        if (bin == "1010")
                return "a";
        if (bin == "1011")
                return "b";
        if (bin == "1100")
                return "c";
        if (bin ==
        "1101")
                return "d";
        if (bin == "1110")
                return "e";
        if (bin == "1111")
                return "f";
}

// Function to convert a decimal value
// to hexa decimal value
string decimaltohex(int64 deci)
{
        // Stores the value as string
        string EQBIN = bitset<64>(deci).to_string();

        // Stores the equivalent hexa
        decimalstring hexstring = "";
        string temp;

        // Traverse the string
        EQBINfor (unsigned int i
        = 0;
                i < EQBIN.length(); i += 4) {
```

```
                temp = EQBIN.substr(i,
                4); hexstring +=
                gethex(temp);
        }

        // Return the
        hexstringreturn
        hexstring;
}

// string to decimal
value int64
BintoDec(string bin)
{

        int64 value =
        bitset<64>(bin)
                                .to_ullong();

        return value;
}

// Function to right rotate x by n
bitsint64 rotate_right(int64 x, int
n)
{
        return (x >> n) | (x << (64 - n));
}

// Function to right shift x by n
bitsint64 shift_right(int64 x, int
n)
{
        return (x >> n);
}
```

```
void separator(string getBlock)
{
        // Stores the size of
        chunksint chunknum =
        0;

        // Traverse the string
        Sfor (unsigned int i =
        0;
                i < getBlock.length();
                i += 64, ++chunknum) {

                // Update the
                Message[chunknum]
                Message[chunknum]
                        = BintoDec(getBlock.substr(i, 64));
        }

        // Iterate over the range [16,
        80]for (int g = 16; g < 80; ++g)
        {

                // Find the WordA
                int64 WordA = rotate_right(Message[g - 2], 19)
                                        ^ rotate_right(Message[g - 2], 61)
                                        ^ shift_right(Message[g - 2], 6);

                // Find the WordB
                int64 WordB = Message[g - 7];
        // Find the WordC
        int64 WordC = rotate_right(Message[g - 15], 1)
                    ^ rotate_right(Message[g - 15], 8)
                    ^ shift_right(Message[g - 15], 7);

        // Find the WordD
        int64 WordD = Message[g - 16];

        // Find the resultant code
        int64 T = WordA + WordB + WordC + WordD;

        // Return the resultant Hash
        CodeMessage[g] = T;
    }
}

// Function to find the major of a,
b, cint64 maj(int64 a, int64 b, int64
c)
{
    return (a & b) ^ (b & c) ^ (c & a);
}

// Function to find the ch value of a,
// b, and c
int64 Ch(int64 e, int64 f, int64 g)
{
    return (e & f) ^ (~e & g);
}
```

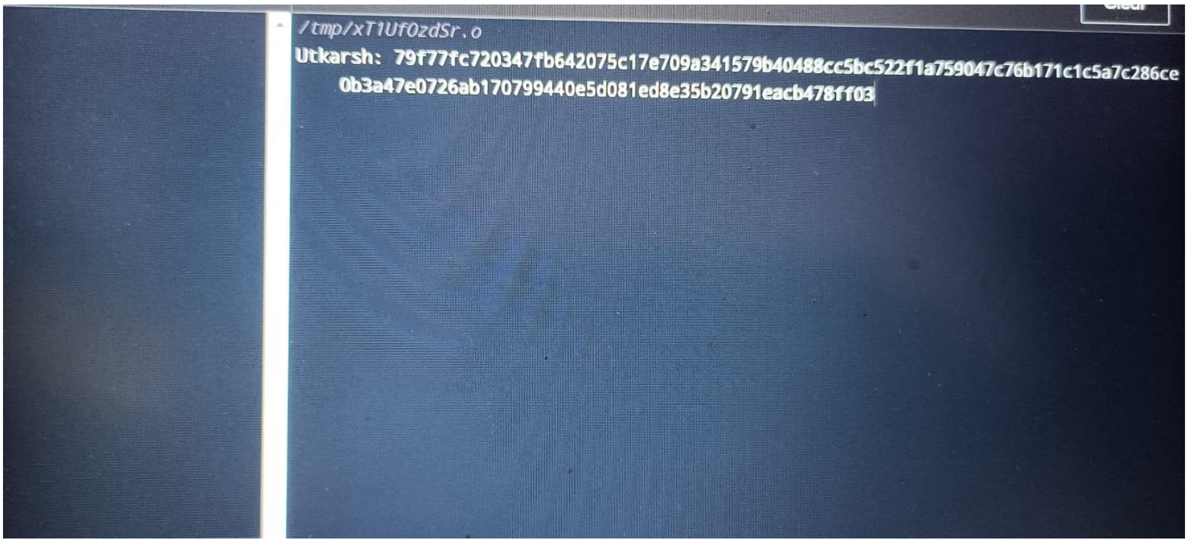
```
// Function to find the Bitwise XOR with
// the right rotate over 14, 18, and
41int64 sigmaE(int64 e)
{
    // Return the resultant
    valuereturn rotate_right(e,
    14)
        ^ rotate_right(e, 18)
        ^ rotate_right(e, 41);
}

// Function to find the Bitwise XOR with
// the right rotate over 28, 34, and
39int64 sigmaA(int64 a)
{

    // Return the resultant
    valuereturn rotate_right(a,
    28)
        ^ rotate_right(a, 34)
        ^ rotate_right(a, 39);
}

// Function to generate the hash
codevoid Func(int64 a, int64 b,
int64 c,
        int64& d, int64 e, int64
        f,int64 g, int64& h, int
        K)
{
    // Find the Hash Code
    int64 T1 = h + Ch(e, f, g) + sigmaE(e) + Message[K]
            + Constants[K];
    int64 T2 = sigmaA(a) + maj(a, b, c);

    d = d +
    T1; h = T1
    + T2;
}

// Function to convert the hash value
// of a given string
string SHA512(string myString)
{
    // Stores the 8 blocks of size
    64 int64 A =
    0x6a09e667f3bcc908; int64 B
    = 0xbb67ae8584caa73b;int64
    C = 0x3c6ef372fe94f82b;
    int64 D = 0xa54ff53a5f1d36f1;
    int64 E =
    0x510e527fade682d1; int64 F
    = 0x9b05688c2b3e6c1f; int64
    G = 0x1f83d9abfb41bd6b;
    int64 H =
    0x5be0cd19137e2179;

    int64 AA, BB, CC, DD, EE, FF, GG, HH;

    stringstream fixedstream;

    // Traverse the string
    Sfor (int i = 0;
        i < myString.size(); ++i) {

        // Add the character to stream
        fixedstream << bitset<8>(myString[i]);
    }

    // Stores string of size
    1024string s1024;

    // Stores the string in the
    // fixedstream
    s1024 = fixedstream.str();

    // Stores the length of
    stringint orilen =
    s1024.length(); int
    tobeadded;

    // Find modded string length
    int modded = s1024.length() % 1024;

    // If 1024-128 is greater than
    moddedif (1024 - modded >= 128)
    {
        tobeadded = 1024 - modded;
    }

    // Else if 1024-128 is less than
    moddedelse if (1024 - modded <
    128) {
        tobeadded = 2048 - modded;
    }

    // Append 1 to
    strings1024 += "1";
```

```
// Append tobeadded-129 zeros
// in the string
for (int y = 0; y < tobeadded - 129;
        y++) {s1024 += "0";
}

// Stores the binary representation
// of string
lengthstring
lengthbits
        = std::bitset<128>(orilen).to_string();

// Append the lengthbits to
strings1024 += lengthbits;

// Find the count of chunks of
// size 1024 each
int blocksnumber = s1024.length() / 1024;

// Stores the numbering of
chunksint chunknum = 0;

// Stores hash value of each
blocksstring
Blocks[blocksnumber];

// Traverse the string s1024
for (int i = 0; i <
s1024.length();
        i += 1024, ++chunknum) {
        Blocks[chunknum] = s1024.substr(i,
        1024);
```

```
}

// Traverse tha array
Blocks[]for (int letsgo = 0;
        letsgo < blocksnumber;
        ++letsgo) {

        // Divide the current string
        // into 80 blocks size 16
        each
        separator(Blocks[letsgo]);

        AA =
        A;BB
        = B;
        CC =
        C; DD
        = D;
        EE =
        E; FF
        = F;
        GG =
        G;HH
        = H;

        int count = 0;

        // Find hash values
        for (int i = 0; i < 10; i++) {

                // Find the Hash Values

                Func(A, B, C, D, E, F, G, H, count);
                count++;
        Func(H, A, B, C, D, E, F, G, count);
        count++;
        Func(G, H, A, B, C, D, E, F, count);
        count++;
        Func(F, G, H, A, B, C, D, E, count);
        count++;
        Func(E, F, G, H, A, B, C, D, count);
        count++;
        Func(D, E, F, G, H, A, B, C, count);
        count++;
        Func(C, D, E, F, G, H, A, B, count);
        count++;
        Func(B, C, D, E, F, G, H, A, count);
        count++;
    }

    // Update the value of A, B, C,
    // D, E, F, G, H


    A +=
    AA;
    B +=
    BB;
    C +=
    CC;
    D +=
    DD;
    E +=
    EE;
    F +=
    FF;
    G +=
    GG;
    H +=
    HH;
}
```

```
    stringstream output;

    // Print the hexadecimal value of
    // strings as the resultant SHA-
    512output << decimaltohex(A);
    output << decimaltohex(B);
    output << decimaltohex(C);
    output << decimaltohex(D);
    output <<  decimaltohex(E);
    output <<  decimaltohex(F);
    output << decimaltohex(G);
    output << decimaltohex(H);

    // Return the
    stringreturn
    output.str();
}

// Driver
Codeint
main()
{
    // Input
    string S = "Utkarsh";

    // Function Call
    cout << S << ": " << SHA512(S);

    return 0;
}
```

Output:



```
/tmp/xT1Uf0zdSr.o
Utkarsh: 79f77fc720347fb642075c17e709a341579b40488cc5bc522f1a759047c76b171c1c5a7c286ce
      0b3a47e0726ab170799440e5d081ed8e35b20791eacb478ff03
```