**PRACTICAL FILE Of**

**INFORMATION SYSTEM SECURITY LAB**



**SUBMITTED BY: SUBMITTED TO:**

***GAURAV BHARDWAJ Dr. KULDEEP KUMAR***

***18103034 ASSISTANT PROFESSOR***

***CSE/5th SEMESTER CSE DEPARTMENT***

***GROUP: G2***

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR**

Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Topics** | **Page No.** | **Sign.** |
| 1. | Lab 1 |  |  |
| 2. | Lab 2 |  |  |
| 3. | Lab 3 |  |  |
| 4. | Lab 4 |  |  |
| 5. | Lab 5 |  |  |
| 6. | Lab 6 |  |  |
| 7. | Lab 7 |  |  |
| 8. | Lab 8 |  |  |
| 9. | Lab 9 |  |  |
| 10. | Lab 10 |  |  |

# **Lab 1**

1. **Inverse of matrix:**

#include <iostream>

using namespace std;

Void Inverse(int Mat[][3], int I[][3],int n){

int P[3];

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

P[j] = Mat[j][j];

for(int k=0;k<3;k++){

for(int j=0;j<3;j++){

Mat[k][j] = Mat[k][j]/P[k];

I[k][j] = I[k][j]/P[k];

}

for(int i=0;i<3;i++){

for(int l=0;l<3;l++){

if(i != k){

Mat[i][l] -= Mat[i][l]\*Mat[k][l];

I[i][l] -= I[i][l]\*I[k][l];

} } } }

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

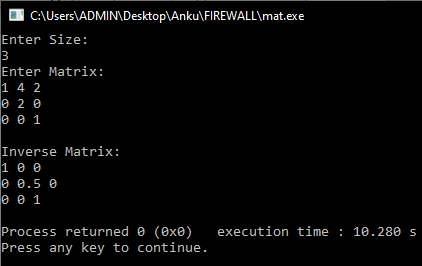
for(int i=0;i<n;i++){

cout<<I[i][j]<<” “;}

cout<<endl;}

}

Int main(){ Output:

 int n;

cin>>n;

int Mat [n][n],I[n][n];

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

for(int j=0;j<n;j++){

if(I == j)

I[i][i] = 1;

else

I[i][j] = 0;

}}

cout<<”Enter Matrix:”<<endl;

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

for(int j=0;j<n;j++)

cin>>Mat[i][j];

Inverse(Mat,I,n); }

1. **Polynomial multiplication and division:**

#include <iostream>

using namespace std;

void Multiply(int arr[],int brr[],int n,int m){

int P[n+m];

for(int i=0;i<(n+m);i++)

P[i] = 0;

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

for(int j=0;j<m;j++)

P[i+j] += arr[i]\*brr[j];

cout<<"Multiplied =>"<<endl;

for(int j=0;j<(n+m);j++)

cout<<P[j]<<" ";

}

int main(){

int a,b;

cout<<"Enter maximum degree of polynomial 1 & 2 (resp.):"<<endl;

cin>>a>>b;

int A[a+1],B[b+1];

cout<<"Enter coef. of poly.1 start with coef. to max degree:"<<endl;

for(int i=0;i<(a+1);i++)

cin>>A[i];

cout<<"Enter coef. of poly.2 start with coef. to max degree:"<<endl;

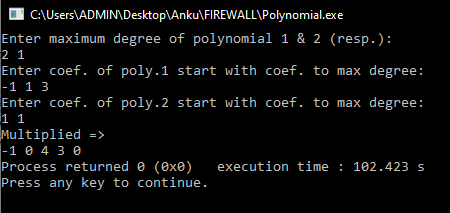
for(int i=0;i<(b+1);i++)

cin>>B[i];

Multiply(A,B,a+1,b+1);

}

**Output:**

****

1. **Euclidean and extended euclidean algo.:**

#include <iostream>

using namespace std;

int gcd(int a, int b){

if(b == 0)

return a;

else

gcd(b,a%b);

}

int extended\_gcd(int a,int b){

int x,y;

return x,y;

}

int main(){

int a,b;

cout<<”Enter Two Numbers:”<<endl;

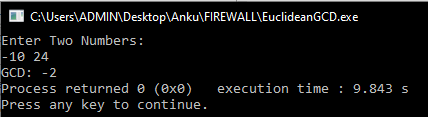
cin>>a>>b;

cout<<”GCD: ”<<gcd(a,b);

return 0;

}

**Output:**



# Lab 2

**Uni-Code for both Playfair and Hill Cipher Mehod**

>|  **Written and Executed in C++ Language =>**

#include<iostream>

#include<cstring>

using namespace std;

//Playfair ciphers

bool present(char a,char p[][5]){

    for(int i=0;i<5;i++)

        for(int j=0;j<5;j++)

            if(p[i][j] == a)

                return false;

    return true;

}

void print(char p[][5]){

    cout<<"Play fair Matrix:"<<endl;

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            cout<<p[i][j]<<" ";

        }

        cout<<endl;

    }

}

void crypt(char a,char b,char p[][5],int d){

    if(a == 'J')

        a='I';

    else if(b=='J')

        b='I';

    int ai,aj,bi,bj;

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            if(p[i][j] == a){

                ai=i;aj=j;

            }

            else if(p[i][j] == b){

                bi=i;bj=j;

            }

    }}

    if(d == 1){

        if(ai == bi){

            int c =(aj+1)%5 ,d=(bj+1)%5;

            a = p[ai][c];

            b = p[bi][d];

        }

        else if(aj == bj){

            int c = (ai+1)%5,d=(bi+1)%5;

            a = p[c][aj];

            b = p[d][bj];

        }

        else{

            a = p[ai][bj];

            b = p[bi][aj];

        }

    }

    else{

        if(ai == bi){

            int c,d;

            //int c =(aj-1)%5 ,d=(bj-1)%5;

            if((aj-1)<0)

                c=4;

            else

                c=aj-1;

            if((bj-1)<0)

                d=4;

            else

                d=bj-1;

            cout<<c<<" "<<d<<endl;

            a = p[ai][c];

            b = p[bi][d];

        }

        else if(aj == bj){

            int c,d;

            if((ai-1)<0)

                c=4;

            else

                c=ai-1;

            if((bi-1)<0)

                d=4;

            else

                d=bi-1;

            a = p[c][aj];

            b = p[d][bj];

        }

        else{

            a = p[ai][bj];

            b = p[bi][aj];

        }

    }

    cout<<a<<b;

}

void Playfair(string text,int c){

    /\* [[P L A Y F

         I R E X M] Playfair Example => text

         B C D G H

         K O Q S T

         U V W X Z]\*/

    int len = text.length();

    char ctext[len+1];

    strcpy(ctext,text.c\_str());

    char PF[5][5]= {{'-','-','-','-','-'},

                           {'-','-','-','-','-'},

                             {'-','-','-','-','-'},

                             {'-','-','-','-','-'},

                             {'-','-','-','-','-'}};//Empty Matrix

    int l=65; // using Ascii value system

    int k=0;

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            if(k<len){

                if(present(ctext[k],PF))

                    PF[i][j] = ctext[k];

                else

                    j--;

                k++;}

            else if(k == len){

                if(present(l,PF) && (l != 74))

                    PF[i][j] = l;

                else

                    j--;

                l++;}

    }}//matrix formation

    print(PF);

    string ms;

    cout<<"Enter Message:"<<endl;

    cin>>ms;

    int len2=ms.length();

    if(len2%2 == 0)//complier consider '\0' as one character

        len2++;    //so string length=>odd becomes even with '\0'

    char msg[len2+1];

    strcpy(msg,ms.c\_str());

    msg[len2] = 'X';

    //msg[len2+1] = '\0';

    cout<<"Converted text:"<<endl;

    if(c == 1){

        for(int i=0;i<len2;i += 2)

            crypt(msg[i],msg[i+1],PF,c);

    }

    else{

        for(int i=0;i<len2;i += 2)

            crypt(msg[i],msg[i+1],PF,c);

    }

}

//Hill Cipher

void Hill(int K[][2],int n,string text,int c=1){

    if(c == 1){

        int b[n][1];

        for(int i=0;i<n;i++)

            b[i][0] = (text[i] - 65);

        int c[n][1];

        for(int i=0;i<n;i++){

                int sum=0;

            for(int j=0;j<n;j++){

                sum += K[i][j]\*b[j][0];

            }

            c[i][0] = sum;

            sum=0;

        }

        for(int i=0;i<n;i++)

            text[i]=65+(c[i][0]%26);

        cout<<"Encrypted Text:"<<text<<endl;

    }

    else{

        int InA[2][2]={{K[1][1],0-K[0][1]},

                        {0-K[1][0],K[0][0]}};

        //int InA[2][2] = {{6,-3},{-3,2}};

    int d=(K[0][0]\*K[1][1])-(K[0][1]\*K[1][0]);

        int b[n][1];

        for(int i=0;i<n;i++)

            b[i][0] = (text[i] - 65);

        cout<<b[0][0]<<" "<<b[1][0]<<endl;

        int c[n][1];

        for(int i=0;i<n;i++){

                int sum=0;

            for(int j=0;j<n;j++){

                sum += InA[i][j]\*b[j][0];

            }

            c[i][0] = sum/d;

            sum=0;

        }

        for(int i=0;i<n;i++)

            text[i]=65+((c[i][0]+26)%26);

        cout<<"Decrypted Text:"<<text<<endl;

    }

}

int main(){

    int c;

    cout<<"1. Playfair\t 2.Hill Cipher Method:"<<endl;

    cin>>c;

    if(c == 1){

        cout<<"Entering Playfair Cipher Method..."<<endl;

        string key;

        cout<<"Enter Key:"<<endl;

        cin>>key;

        int d;

        cout<<"1. Encrypt \t 2. Decrypt:"<<endl;

        cin>>d;

        Playfair(key,d);

    }

    else{

        cout<<"Entering Hill Cipher Method..."<<endl;

        int K[2][2];

        cout<<"Enter 2x2 Matrix:"<<endl;

        for(int i=0;i<2;i++)

            for(int j=0;j<2;j++)

                cin>>K[i][j];

        string Text;

        cout<<"Enter Capitalized 2 Letters:"<<endl;

        cin>>Text;

        int c;

        cout<<"1) Encrypt \ t2) Decrypt:"<<endl;

        cin>>c;

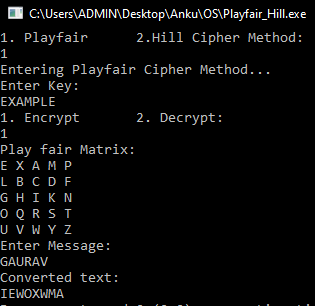
        Hill(K,2,Text,c);

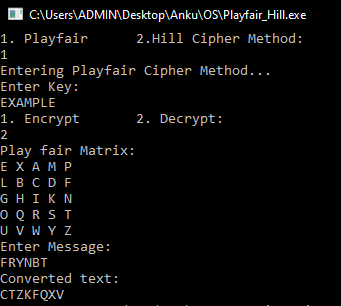
    }

    return 0;

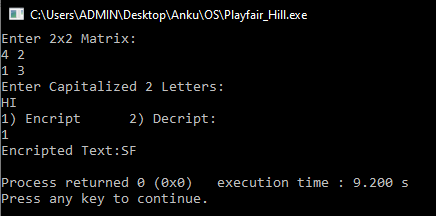
}

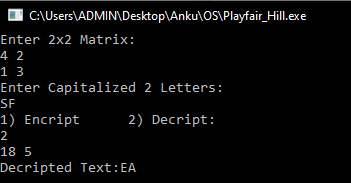
**Playfair Cipher Outputs:**

****

****

**Hill Cipher Outputs:**





# **Lab 3**

**Simplified - DES**

Code:

#include <iostream>

using namespace std;

#define N 10

/\*S-0

        Col 0 1 2 3

        Rows

        0 01 00 11 10

        1 11 10 01 00

        2 00 10 01 11

        3 11 01 11 10\*/

int S0[4][4] = { { 01,00,11,10 }, { 11,10,01,00 },{ 00,10,01,11 },{ 11,01,11,10 } };

/\*S-1

        Col 0 1 2 3

        Rows

        0 00 01 10 11

        1 10 00 01 11

        2 11 00 01 00

        3 10 01 00 11\*/

int S1[4][4]={{00,01,10,11},{10,00,01,11},{11,00,01,00},{10,01,00,11}};

void P\_10(int k[]){

    /\*Input 1 2 3 4 5 6 7 8 9 10

    Output 3 5 2 7 4 10 1 9 8 6  \*/

    int temp\_k[10];

    for(int i=0;i<10;i++)

        temp\_k[i]=k[i];

    int temp[]={2,4,1,6,3,9,0,8,7,5};

    for(int i=0;i<10;i++)

        k[i] = temp\_k[temp[i]];

}

void IP\_8(int pt[]){

    /\*Bits number 1 2 3 4 5 6 7 8

Permute the bits 2 6 3 1 4 8 5 7  \*/

    int temp\_pt[8];

    for(int i=0;i<8;i++)

        temp\_pt[i]=pt[i];

    int temp[8] = {1,5,2,0,3,7,4,6};

    for(int i=0;i<8;i++)

        pt[i]=temp\_pt[temp[i]];

}

void E\_P(int pt[],int l,int h){

    /\*Right half bits 1 0 0 0

                Number 1 2 3 4 5 6 7 8

            Expand bits 4 1 2 3 2 3 4 1

        Output of bits 0 1 0 0 0 0 0 1 \*/

    int temp1[4];

    for(int i=l;i<h;i++)

        temp1[i-l]=pt[i];

    int temp[8]={3,0,1,2,1,2,3,0};

    for(int i=l;i<h+4;i++)

        pt[i]=temp1[temp[i]];

}

void P\_8(int k[]){

    /\*Input 1 2 3 4 5 6 7 8 9 10

    Output 6 3 7 4 8 5 10 9 \*/

    int temp\_k[10];

    for(int i=0;i<10;i++)

        temp\_k[i]=k[i];

    int temp[10]={5,2,6,3,7,4,9,8,0,0};

    for(int i=0;i<10;i++)

        k[i]=temp\_k[temp[i]];

}

void P\_4(int p[]){

    /\*Numbers 1 2 3 4

        Output 2 4 3 1\*/

    int temp\_p[4];

    for(int i=0;i<4;i++)

        temp\_p[i]=p[i];

    int temp[4]={1,3,2,0};

    for(int i=0;i<4;i++)

        p[i]=temp\_p[temp[i]];

}

void SwapNibble(int tp[]){

    int t[4];

    for(int i=0;i<4;i++){

        t[i]=tp[i];

        tp[i]= tp[i+4];

        }

    for(int i=4;i<8;i++)

        tp[i]=t[i-4];

}

void RShift(int k[N],int l,int h,int cnt=1){

    while(cnt != 0){

    int temp=k[l];

    for(int i=l;i<h;i++)

        k[i]= k[i+1];

    k[h-1]=temp;

    cnt--;

    }

}

void print(int k[N],int sz){

    for(int i=0;i<sz;i++)

        cout<<k[i]<<" ";

}

int main(){

    /\*It is a block cipher.

    It has 8-bits block size of plain text or cipher text.

    It uses 10-bits key size for encryption.

    It is a symmetric cipher.

    It has Two Rounds.\*/

    //Just select a random key of 10-bits

    int k1[]={1,0,1,0,0,0,0,0,1,0},k2[]={1,0,1,0,0,0,0,0,1,0};

    //Put this key into P.10 Table and permute the bits.

    cout<<"Original Key:\t";

    print(k1,10);

    cout<<endl<<endl;

    P\_10(k1);

    cout<<"Key 1 (p 10):\t";

    print(k1,10);

    cout<<endl;

    //Divide the key into two halves, left half and right half

    //Now apply the one bit Round shift on each half

    RShift(k1,0,5);

    RShift(k1,5,10);

    cout<<"Key 1 Right Shift:\t";

    print(k1,10);

    cout<<endl;

    //Now once again combine both halve of the bits, right and left. Put them into the P8 table.

    P\_8(k1);

    cout<<"Key 1 (p 8):\t";

    print(k1,8);

    cout<<endl<<endl;

    //we need to generate a second bit and after that we will move to encrypt the plain text or message.

    cout<<"Key 2 original:\t";

    print(k2,10);

    cout<<endl;

    P\_10(k2);

    cout<<"Key 2 (p 10):\t";

    print(k2,10);

    cout<<endl;

    RShift(k2,0,5);

    RShift(k2,5,10);

    cout<<"Key 2 R Shift:\t";

    print(k2,10);

    cout<<endl;

    //Select those halves which are output of first round shift.take the output of first round shift in above step 4.

    //After the two rounds shift on each half out-put of each half will be.

    RShift(k2,0,5,2);

    RShift(k2,5,10,2);

    cout<<"Key 2 R Shift(2):\t";

    print(k2,10);

    cout<<endl;

    P\_8(k2);

    cout<<"Key 2 (p 8):\t";

    print(k2,8);

    cout<<endl<<endl;

    //Encryption of Plain text into Cipher text in S-DES:

    //Suppose this is our plain text in binary which is 8-bit.

    int Plain\_text[12]={0,1,1,1,0,0,1,0,0,0,0,0};

    int Plain\_text2[12];

    cout<<"Plain text Original:\t";

    print(Plain\_text,8);

    cout<<endl;

    //Put the plain text into IP-8(initial permutation) table and permute the bits.

    IP\_8(Plain\_text);

    cout<<"Plain Text (IP-8):\t";

    print(Plain\_text,8);

    cout<<endl;

    int temp\_r\_pt[4];

    for(int i=0;i<4;i++)

        temp\_r\_pt[i]=Plain\_text[i+4];

    //Now break the bits into two halves, each half will consist of 4 bits. The halves will be right and left.

    //Take the right 4 bits and put them into E.P (expand and per-mutate) Table.

    E\_P(Plain\_text,4,8);

    cout<<"Plain Text (EP):\t";

    print(Plain\_text,12);

    cout<<endl;

    //Now, just take the output and XOR it with First key Or K1

    for(int i=4;i<12;i++)

        Plain\_text[i] = Plain\_text[i]^k1[i-4];

    cout<<"Plain Text XOR k1:\t";

    print(Plain\_text,12);

    cout<<endl;

    //Once again split the output of XOR’s bit into two halves and each half will  consist of 4 bits

    //Now put the each half into the s-boxes, there is only two s-boxes. S-0 and S-1

    int r0,c0,r1,c1;

    r0=(Plain\_text[4]\*Plain\_text[4]) + Plain\_text[7];

    c0=(Plain\_text[5]\*Plain\_text[5]) + Plain\_text[6];

    r1=(Plain\_text[8]\*Plain\_text[8]) + Plain\_text[11];

    c1=(Plain\_text[9]\*Plain\_text[9]) + Plain\_text[10];

    int left=S0[r0][c0],right=S1[r1][c1];

    int P[4] = {left%100,left%10,right%100,right%10};

    cout<<"P Original:\t";

    print(P,4);

    cout<<endl;

    //Now get XOR the output with left 4 bits of Initial Per-mutation. The left bits of initial per-mutation are in step 3.

    for(int i=0;i<4;i++)

        P[i]=P[i]^Plain\_text[i];

    cout<<"Plain Text XOR (p):\t";

    print(P,4);

    cout<<endl;

    //Now get the right half of the initial permutation, which is step 3, and combine that with this out- put.

    int Temp\_p[8];

    for(int i=0;i<4;i++)

        Temp\_p[i]=P[i];

    for(int i=4;i<8;i++)

        Temp\_p[i]=temp\_r\_pt[i-4];

    cout<<"P After combine:\t";

    print(Temp\_p,8);

    cout<<endl;

    //Now swap both halves, which means put the left half in place of right and vice versa

    SwapNibble(Temp\_p);

    cout<<"Temp p (Swapped):\t";

    print(Temp\_p,8);

    cout<<endl<<endl;

    for(int i=0;i<8;i++)

        Plain\_text[i]=Temp\_p[i];

    //Repeat from Step 3 with key k2

    for(int i=0;i<4;i++)

        temp\_r\_pt[i]=Plain\_text[i+4];

    //Now break the bits into two halves, each half will consist of 4 bits. The halves will be right and left.

    //Take the right 4 bits and put them into E.P (expand and per-mutate) Table.

    E\_P(Plain\_text,4,8);

    cout<<"Plain Text (EP):\t";

    print(Plain\_text,12);

    cout<<endl;

    //Now, just take the output and XOR it with Second key Or K2

    for(int i=4;i<12;i++)

        Plain\_text[i] = Plain\_text[i]^k2[i-4];

    cout<<"Plain Text XOR k2:\t";

    print(Plain\_text,12);

    cout<<endl;

    //Once again split the output of XOR’s bit into two halves and each half will  consist of 4 bits

    //Now put the each half into the s-boxes, there is only two s-boxes. S-0 and S-1

    r0=(Plain\_text[4]\*Plain\_text[4]) + Plain\_text[7];

    c0=(Plain\_text[5]\*Plain\_text[5]) + Plain\_text[6];

    r1=(Plain\_text[8]\*Plain\_text[8]) + Plain\_text[11];

    c1=(Plain\_text[9]\*Plain\_text[9]) + Plain\_text[10];

    left=S0[r0][c0],right=S1[r1][c1];

    P[0] = left%100;P[1]=left%10;P[2]=right%100;P[3]=right%10;

    cout<<"P Original:\t";

    print(P,4);

    cout<<endl;

    //Now get XOR the output with left 4 bits of Initial Per-mutation. The left bits of initial per-mutation are in step 3.

    for(int i=0;i<4;i++)

        P[i]=P[i]^Plain\_text[i];

    cout<<"Plain Text XOR (p):\t";

    print(P,4);

    cout<<endl;

    //Now get the right half of the initial permutation, which is step 3, and combine that with this out- put.

    for(int i=0;i<4;i++)

        Temp\_p[i]=P[i];

    for(int i=4;i<8;i++)

        Temp\_p[i]=temp\_r\_pt[i-4];

    cout<<"P2 After combine:\t";

    print(Temp\_p,8);

    cout<<endl;

    //Now swap both halves, which means put the left half in place of right and vice versa

    SwapNibble(Temp\_p);

    cout<<"Temp p (Swapped):\t";

    print(Temp\_p,8);

    cout<<endl;

    //Take IP-Inverse from IP^{-1} Table => IP Table

    IP\_8(Temp\_p);

    int Cipher\_Text[8];

    for(int i=0;i<8;i++)

        Cipher\_Text[i]=Temp\_p[i];

    cout<<"Cipher Text :\t";

    print(Cipher\_Text,8);

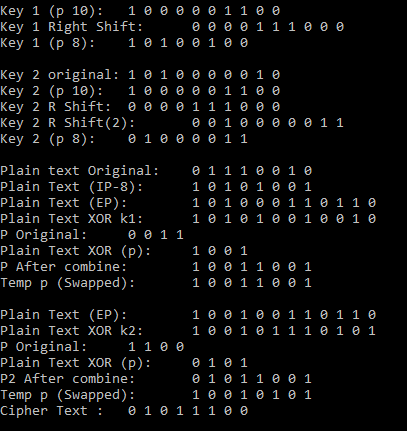
    cout<<endl;

    //Phew!! (-\_-)^

    return 0;

}

**Output:**

****

# **Lab 6**

**Implementation of Simplified RC4 Algorithm:**

→|

#include<iostream>

using namespace std;

void Swap(int s[],int a, int b){

    int t= s[a];

    s[a]=s[b];

    s[b]=t;

}

void KeyStream(int s[],int n,int ks[],int pc\_len=6){

    int i=0,j=0;

    for(i=1;i<pc\_len;i++){

     j = (j + s[i])%n;

     Swap(s,i,j);

     int t = (s[i] + s[j])%n;

     ks[i-1] = s[t];

    }

}

void print(int a[],int n){

    for(int i=0;i<n;i++)

        cout<<a[i]<<" ";

    cout<<endl;

}

int main(){

    int n,msg;

    cin>>n;

    int S[n],T[n];

    for(int i=0;i<n;i++)

        S[i]=i;

    cout<<"S block: ";

    print(S,n);

    int k\_size;

    cin>>k\_size;

    int K[k\_size];

    for(int i=0;i<k\_size;i++)

        cin>>K[i];

    cout<<"Key: ";

    print(K,k\_size);

    //int P[4],C[4]={7,1,5,4},Kstrm[4];

    cout<<"Message size:"<<endl;

    cin>>msg;

    int Text[msg],Cipher[msg],Kstrm[msg];

    if (k\_size < n)

        for(int i=0;i<n;i++)

            T[i] = K[i%k\_size];

    cout<<"Key: ";

    print(T,n);

    //Key scheduling

    int i=0,j=0;

    for(i=0;i<n;i++){

        j= (j + S[i] + T[i])%n;

        Swap(S,i,j);

    }

    cout<<"s block: ";

    print(S,n);

    cout<<"Message/Cipher:"<<endl;

    for(int i=0;i<msg;i++)

        cin>>Text[i];

    KeyStream(S,n,Kstrm,k\_size);

    print(Kstrm,msg);

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

    for(int i=0;i<msg;i++)

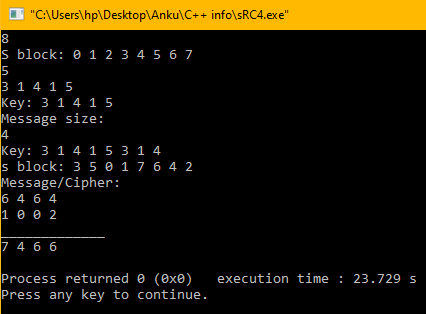
        Cipher[i]=Kstrm[i]^Text[i];

    print(Cipher,msg);

    return 0;

}

**Output:**



# **Lab 7**

**Implementation of Miller Rabin and CRT Algorithms**

**>| Miller Rabin:**

#include <iostream>

#include <cmath>

#define lln long long

using namespace std;

void Miller\_Rabin(int num,int rt){

    int temp\_num,k=0,m=0;

    if(num%2 != 0)

        temp\_num= num-1;

    else

        temp\_num=num;

    int temp = temp\_num;

    while(temp\_num%2 == 0){

        k++;

        temp\_num /= 2;

    }

    m=temp\_num;

    cout<<"k:"<<k<<"\t m:"<<m<<endl;

    while(k != 0){

        lln int t = pow(rt,m);

        cout<<"t:"<<t<<"\t";

        t = t%num;

        cout<<"T:"<<t<<"\t";

        if(t == temp)

            break;

        else

            m = 2\*m;

        cout<<"m:"<<m<<endl;

        k--;

    }

    if(k != 0)

        cout<<num<<" is prime number."<<endl;

    else

        cout<<num<<" is not a prime number."<<endl;

}

int main(){

    cout<<"======>Miller-Rabin-Primality-Test<======"<<endl;

    int number,root;

    cout<<"Enter Number and root:"<<endl;

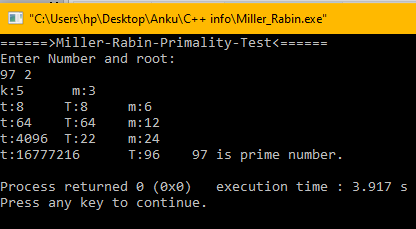
    cin>>number>>root;

    Miller\_Rabin(number,root);

    return 0;

}

**Output:**



**Chinese Remainder Theorem:**

#include <iostream>

#include <cmath>

using namespace std;

void print(int a[],int n){

    for(int i=0;i<n;i++)

        cout<<a[i]<<" ";

    cout<<endl;

}

int MI(int a,int m){

    int i,b=0;

    for(i=1;i<INT\_MAX;i++){

        int t= (a\*i)%m;

        if(t == 1){

           b=i;

           break;

        }

    }

    if (b != 0)

        return b;

    else

        return -1;

}

void CRT(int a[],int m[],int n){

    int M=1;

    int Mv[n],Mi[n];

    cout<<"--------('Hidden')--------"<<endl;

    for(int i=0;i<n;i++)

        M \*= m[i];

    cout<<"M: "<<M<<endl;

    for(int i=0;i<n;i++)

        Mv[i] = M/m[i];

    cout<<"M vector: ";

    print(Mv,n);

    /\* Inverse function \*/

    for(int i=0;i<n;i++)

        Mi[i] = MI(Mv[i],m[i]);

    cout<<"M Inverse: ";

    print(Mi,n);

    cout<<"C : ";

    print(a,n);

    cout<<"Sum => (a\*M\*Mi): 0";

    int f=0;

    for(int i=0;i<n;i++){

        f += (a[i]\*Mv[i]\*Mi[i]);

        cout<<"->"<<f;

        }

    int x = f%M;

    cout<<endl<<"--------(++++++++)--------"<<endl;

    cout<<"Number:"<<x<<endl;

}

int main(){

    cout<<"======>Chinese-Remainder-Theorem-Test<======"<<endl;

    int n;

    cout<<"Enter the number of statements(n):";

    cin>>n;

    int c[n],m[n];

    cout<<"Enter c,m for n statements: eg- c(mod m)"<<endl;

    for(int i=0;i<n;i++)

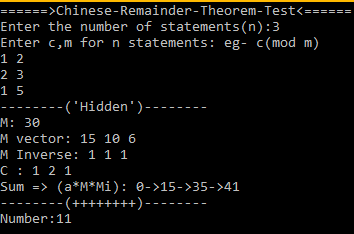
        cin>>c[i]>>m[i];

    CRT(c,m,n);

    return 0;

}

**Output:**



# **Lab 8**

**Implementation of RSA algorithm:**

#include<iostream>

#include<cmath>

using namespace std;

int gcd(int a,int b){

    if(b == 0)

        return a;

    else

        return gcd(b,a%b);

}

int MI(int a,int m){

    int i,b=0;

    for(i=1;i<INT\_MAX;i++){

        int t= (a\*i)%m;

        if(t == 1){

           b=i;

           break;

        }

    }

    if (b != 0)

        return b;

    else

        return -1;

}

void RSA(int p,int q,int cryp){

    //Key Generation

    int n = p\*q;

    cout<<"n:"<<n<<endl;

    int Tn= (p-1)\*(q-1);

    cout<<"Totient function:"<<Tn<<endl;

    int e,ch;

    cout<<"1. Private Key \t 2. Public Key:"<<endl;

    cin>>ch;

    l2:cout<<"Enter key"<<endl;

    cin>>e;

    if(e>1 && e<Tn && gcd(e,Tn)== 1)

        cout<<"";

    else

        goto l2;

    cout<<"public key:"<<e<<endl;

    int d= MI(e,Tn);

    cout<<"private key:"<<d<<endl;

    //Encryption

    int m,c;

    cout<<"Enter message:"<<endl;

    cin>>m;

    if(m > n)

        m= m%n;

    c = pow(m,e);

    c = c%n;

    if(c < 0)

        c = n+c;

    if(cryp == 1){

        cout<<"Encrypted message:"<<c<<endl;

    }

    //Decryption

    else{

        cout<<"Decrypted message:"<<m<<endl;

    }

}

int main(){

    int p,q;

    l1: cout<<"Two Prime numbers p,q:"<<endl;

    cin>>p>>q;

    if(p%2 == 0 || q%2 == 0){

        cout<<"p and q should be prime"<<endl;

        goto l1;

    }

    int en;

    cout<<"1. Encryption \t 2. Decryption:"<<endl;

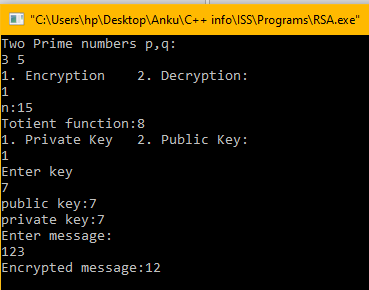
    cin>>en;

    RSA(p,q,en);

    return 0;

}

**Output:**

****

**Implementation of Rabin Algorithm:**#include<iostream>

#include<cmath>

using namespace std;

int gcdExtended(int a, int b, int \*x, int \*y)  {

    // Base Case

    if (a == 0){

        \*x = 0;

        \*y = 1;

        return b;

    }

    int x1, y1; // To store results of recursive call

    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of

    // recursive call

    \*x = y1 - (b/a) \* x1;

    \*y = x1;

    return gcd;

}

int gcd(int a,int b){

    if(b == 0)

        return a;

    else

        return gcd(b,a%b);

}

int MI(int a,int m){

    int i,b=0;

    for(i=1;i<INT\_MAX;i++){

        int t= (a\*i)%m;

        if(t == 1){

           b=i;

           break;

        }

    }

    if (b != 0)

        return b;

    else

        return -1;

}

void Rabin(int p,int q,int cryp){

    //Key Generation

    int n = p\*q;

    int Tn= (p-1)\*(q-1);

    cout<<"Totient function:"<<Tn<<endl;

    cout<<"public key:"<<n<<endl;

    //int d= MI(e,Tn);

    cout<<"private key(p,q):("<<p<<","<<q<<")"<<endl;

    //Encryption

    int m,c;

    cout<<"Enter message:"<<endl;

    if(cryp == 1){

        cin>>m;

        if(m > n)

            m= m%n;

        c = pow(m,2);

        c = c%n;

        if(c < 0)

            c = n+c;

        cout<<"Encrypted message:"<<c<<endl;

    }

    //Decryption

    else{

        cin>>c;

        int mpp,mpn,mqp,mqn;

        int a= (p+1)/4,b = (q+1)/4,x,y;

        a = pow(c,a);

        b = pow(c,b);

        mpp = a%p; //-

        mqp = b%q; //-

        // Extended Gcd

        int g = gcdExtended(p,q,&x,&y);

        int X = (mqp\*x\*p + mpp\*y\*q)%p;

        int Y = (mqp\*x\*p - mpp\*y\*q)%q;

        mpp = X,mqp=Y,mpn = n-X,mqn=n-Y;

        cout<<"Decrypted message:"<<endl;

        cout<<mpp<<" |"<<mpn<<" |"<<mqp<<" |"<<mqn<<endl;

    }

}

int main(){

    int p,q;

    l1: cout<<"Two Prime numbers p,q:"<<endl;

    cin>>p>>q;

    if((p%2 == 0 || q%2 == 0) && (p%4 !=3 && q%4 !=3) && (p != 2 || q != 2)){

        cout<<"p and q should be prime"<<endl;

        goto l1;

    }

    int en;

    cout<<"1. Encryption \t 2. Decryption:"<<endl;

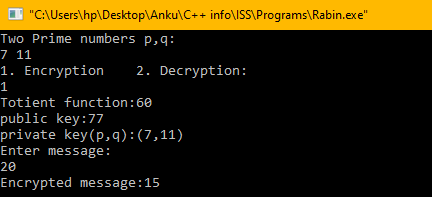
    cin>>en;

    Rabin(p,q,en);

    return 0;

}

**Output:**

****

# **Lab 9**

**Implementation of MAC Protocol:**

**>|**

#include<iostream>

#include "sDES.cpp"

#include <cstring>

using namespace std;

int to\_integer(int bin[]){

int dec=0, prod=1;

for(int i=0; i<8; i++){

        if(bin[i] > 1)

            bin[i] = 1;

dec += prod\*bin[i];

prod \*= 2;

}

return dec;

}

int to\_binary(int a[],int num){

    int number=0,i=0;

    while(num>0){

        a[i] = num%2;

        num /= 2;

        i++;

    }

    for(int j=0;j<int(i/2);j++){

        int temp= a[j];

        a[j] = a[i-j-1];

        a[i-j-1] = temp;

    }

    print(a,i);

}

void to\_binary(int arr[],string s){

    int len = s.length();

    int a[len];

    for(int i=0;i<len;i++){

        arr[i] = int(s[i]);

        to\_binary(a,arr[i]);

    }

}

void MAC\_algorithm(int msg[],int msg\_len,int key[]){

    int b=10;

    int a[12],temp[8],ch[b];

    cout<<"Working on DES......................"<<endl;

    for(int i=0;i<msg\_len;i++){

        to\_binary(a,msg[i]);

        sDES(key,a,b,msg\_len,temp);

        ch[i] = to\_integer(temp);

    }

    char c[10];

    for(int i=0;i<6;i++)

        c[i] = ch[i]%26 + 65;

    cout<<"TAG: "<<c<<endl;

}

int main(){

    string msg="Gaurav";

    cout<<"Message: "<<msg<<endl;

    int len=msg.length();

    char c[len+1];

    int key[]={0,1,0,1,0,0,1,1,1,0},mSg[len+1];

    strcpy(c,msg.c\_str());

    for(int i=0;i<len;i++)

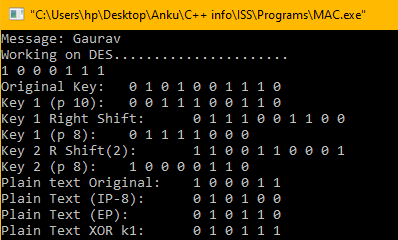
        mSg[i] = c[i];

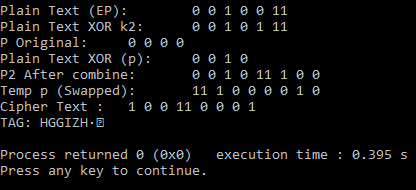
    MAC\_algorithm(mSg,len,key);

    return 0;

}

**Output:**

****

****

**Implementation of HMAC protocol:**

>|#include <iostream>

#include <cstring>

#include <string>

#include "SHA1.h"

using namespace std;

void print(int arr[],int len){

    for(int i=0;i<len;i++)

        cout<<arr[i];

    cout<<endl;

}

string Hash(string msg,int k){

    char K= k%26 + 65;

    msg = to\_string(K)+msg;

    string msgE;

    msgE = sha1(msg);

    return msgE;

}

void HMAC(string msg,int lenm,int K,int b\_size){

    cout<<"Working on HMAC............................"<<endl;

    if(lenm%b\_size != 0)

        while(lenm%b\_size != 0){msg = msg+"0";lenm++;}

    char msg\_c[lenm+1];

    int c[lenm+1];

    strcpy(msg\_c,msg.c\_str());

    const unsigned int ipad = 0x36;

    const unsigned int opad = 0x5c;

    int k1 = K^ipad,k2=K^opad;

    cout<<"K1: "<<k1<<"\nk2: "<<k2<<endl;

    for(int i=0;i<lenm+1;i++)

        c[i]=int(msg\_c[i]);

    msg="";

    for(int i=0;i<lenm+1;i++)

        msg = msg + to\_string(c[i]);

    string cipher=Hash(Hash(msg,k1),k2);

    cout<<"\nHMAC :"<<cipher<<endl;

}

int main(){

    int K,b;

    string msg;

    cout<<"Key: ";

    cin>>K;

    cout<<"Message: ";

    cin>>msg;

    cout<<"Block size: ";

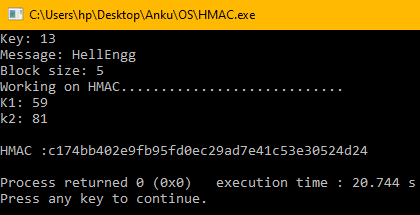
    cin>>b;

    HMAC(msg,msg.length(),K,b);

    return 0;

}

**Output:**



# **Lab 10**

**Implementation of Diffie - Hll Key Exchange Protocol:**

**>|**#include <iostream>

#include <cmath>

using namespace std;

int main(){

    long long int p,g,a,Ka,Kb;

    cout<<"Enter p and g:";

    cin >> p >> g;

    cout<<"Enter random no.(< p):";

    cin>>a;//Random no. 0 < a < p-1

    long long int x = pow(g,a);

    x =x%p;

    long long int b;

    cout<<"Enter random no.(< p):";

    cin>>b;// Random no. 0 < b < p-1

    long long int y = pow(g,b);

    y = y % p;

    Ka = pow(y,a);

    Ka = Ka % p;

    Kb = pow(x,b);

    Kb = Kb % p;

    if (Ka < 0)

        Ka = p+Ka;

    if(Kb < 0)

        Kb = p+Kb;

    cout<<"Secret Key Alice:"<<Ka <<endl;

    cout<<"Secret Key Bob:"<<Kb <<endl;

    return 0;

}

**Output:**

