P1-> Ceaser Bruteforce

```
#include <iostream>
using namespace std;
void decrypt(char msg[]);
char* encrypt(char message[]){
 int i, key=3;
 char ch;
for(i = 0; message[i] != '\0'; ++i){ //traverse till eof
 ch = message[i];
 if(ch >= 'a' && ch <= 'z'){

 ch = ch + key;
 // cout<<"ch="<<ch<<endl;
 if(ch > 'z'){
 ch = ch - 'z' + 'a' - 1;
 // cout<<"ch>z="<<ch<<endl;
 }
 message[i] = ch;
 }
 else if(ch >= 'A' && ch <= 'Z'){
 ch = ch + key;
 if(ch > 'Z'){
 ch = ch - 'Z' + 'A' - 1;
 }
 message[i] = ch;
}
}
cout << "Encrypted=" << message;
//decrypt(message);
return message;
}
```

```
char* decrypt(char message[],int key){
 char ch;
int i;
for(i = 0; message[i] != '\0'; ++i){
 ch = message[i];
 if(ch >= 'a' && ch <= 'z'){
 ch = ch - key;

 //cout<<endl<<"ch = ch - key;"<<int(ch);
 if(ch < 'a'){
 ch = ch + 'z' - 'a' + 1;
 // cout<<endl<<"ch<a="<<ch;
 }
 message[i] = ch;
 }
 else if(ch >= 'A' && ch <= 'Z'){
 ch = ch - key;
 if(ch > 'a'){
 ch = ch + 'Z' - 'A' + 1;
 }
 message[i] = ch;
}
}
cout <<endl<<"Decrypted=" << message;
 }
int main()
{
char message[100];
cout << "Message=";
cin.getline(message, 100);
char* x=encrypt(message);
//decrypt(x,3);


cout<<endl<<"Bruteforcing"<<endl;
for(int i=0;i<26;i++){
 char* y=decrypt(x,i);
 if(y==x){
 cout<<"Found key="<<i;


 break;
 }
}
return 0;
}
```

P2->Playfair

```
import java.awt.Point;
import java.util.Scanner;
public class crnsp2
{

private int length = 0;

private String [][] table;

public static void main(String args[])
{
crnsp2 pf = new crnsp2();
}

private crnsp2()
{

System.out.print("Enter the key for playfair cipher: ");
Scanner sc = new Scanner(System.in);
String key = parseString(sc);
while(key.equals(""))
key = parseString(sc);
table = this.cipherTable(key);

System.out.print("Enter the plaintext to be encipher: ");

String input = parseString(sc);
while(input.equals(""))
input = parseString(sc);
```

```
String output = cipher(input);
String decodedOutput = decode(output);
//output the results to user
this.keyTable(table);
this.printResults(output,decodedOutput);
}

private String parseString(Scanner sc)
{
String parse = sc.nextLine();

parse = parse.toUpperCase();

parse = parse.replaceAll("[^A-Z]", "");

parse = parse.replace("J", "I");
return parse;
}

private String[][] cipherTable(String key)
{

String[][] playfairTable = new String[5][5];
String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

for(int i = 0; i < 5; i++)
for(int j = 0; j < 5; j++)
playfairTable[i][j] = "";
for(int k = 0; k < keyString.length(); k++)
{
boolean repeat = false;
boolean used = false;
for(int i = 0; i < 5; i++)
{
for(int j = 0; j < 5; j++)
{
if(playfairTable[i][j].equals("" + keyString.charAt(k)))
{
repeat = true;
}
else if(playfairTable[i][j].equals("") && !repeat && !used)
{
playfairTable[i][j] = "" + keyString.charAt(k);
used = true;
}
}
}
}
return playfairTable;
}

private String cipher(String in)
{
length = (int) in.length() / 2 + in.length() % 2;

for(int i = 0; i < (length - 1); i++)
{
if(in.charAt(2 * i) == in.charAt(2 * i + 1))
{
in = new StringBuffer(in).insert(2 * i + 1, 'X').toString();
length = (int) in.length() / 2 + in.length() % 2;
}
```

```
}

String[] digraph = new String[length];
//loop iterates over the plaintext
for(int j = 0; j < length ; j++)
{

if(j == (length - 1) && in.length() / 2 == (length - 1))

in = in + "X";
digraph[j] = in.charAt(2 * j) +""+ in.charAt(2 * j + 1);
}

String out = "";
String[] encDigraphs = new String[length];
encDigraphs = encodeDigraph(digraph);
for(int k = 0; k < length; k++)
out = out + encDigraphs[k];
return out;
}

private String[] encodeDigraph(String di[])
{
String[] encipher = new String[length];
for(int i = 0; i < length; i++)
{
char a = di[i].charAt(0);
char b = di[i].charAt(1);
int r1 = (int) getPoint(a).getX();
int r2 = (int) getPoint(b).getX();
int c1 = (int) getPoint(a).getY();
int c2 = (int) getPoint(b).getY();

if(r1 == r2)
{
c1 = (c1 + 1) % 5;
c2 = (c2 + 1) % 5;
}

else if(c1 == c2)
{
r1 = (r1 + 1) % 5;
r2 = (r2 + 1) % 5;
}

else
{
int temp = c1;
c1 = c2;
c2 = temp;
}

encipher[i] = table[r1][c1] + "" + table[r2][c2];
}
return encipher;
}

private String decode(String out)
{
String decoded = "";
for(int i = 0; i < out.length() / 2; i++)
{
```

```
char a = out.charAt(2*i);
char b = out.charAt(2*i+1);
int r1 = (int) getPoint(a).getX();
int r2 = (int) getPoint(b).getX();
int c1 = (int) getPoint(a).getY();
int c2 = (int) getPoint(b).getY();
if(r1 == r2)
{
c1 = (c1 + 4) % 5;
c2 = (c2 + 4) % 5;
}
else if(c1 == c2)
{
r1 = (r1 + 4) % 5;
r2 = (r2 + 4) % 5;
}
else
{
int temp = c1;
c1 = c2;
c2 = temp;
}
decoded = decoded + table[r1][c1] + table[r2][c2];
}

return decoded;
}

private Point getPoint(char c)
{
Point pt = new Point(0,0);
```

```
for(int i = 0; i < 5; i++)

for(int j = 0; j < 5; j++)

if(c == table[i][j].charAt(0))

pt = new Point(i,j);

return pt;

}


private void keyTable(String[][] printTable)

{

System.out.println("Playfair Cipher Key Matrix: ");

System.out.println();


for(int i = 0; i < 5; i++)

{


for(int j = 0; j < 5; j++)

{


System.out.print(printTable[i][j]+" ");

}

System.out.println();

}

System.out.println();

}

//method that prints all the results

private void printResults(String encipher, String dec)

{

System.out.print("Encrypted Message: ");

//prints the encrypted message

System.out.println(encipher);

System.out.println();
System.out.print("Decrypted Message: ");

//prints the decryted message

System.out.println(dec);

System.out.println("19DCS060\nPriyanshu Maurya");

}

}
```

3->Rail Fence

```
public class crnspract3 {


        public static void main(String[] args) {
                // TODO Auto-generated method stub
                crnspract3 p=new crnspract3();
                String str="300 achieved glory at hot gate, unite for Greece";
                str=str.replaceAll("\\s","");//removing white spaces
                System.out.println(str.length());
                String encrypted=p.encryptRailFence(str, 4);
                System.out.println("Encrypted="+encrypted);
                String decrypted=p.decryptRailFence(encrypted, 4);
                System.out.println("\nDecrypted="+decrypted);
                System.out.println("\n19DCS060\nPriyanshu Maurya");


        }
        String encryptRailFence(String text, int key)
        {
                char rail[][]=new char[key][(text.length())];


           // filling the rail matrix to distinguish filled
           // spaces from blank ones
           for (int i=0; i < key; i++)
               for (int j = 0; j < text.length(); j++)
                   rail[i][j] = '\n';


           boolean dir_down = false;
           int row = 0, col = 0;


           for (int i=0; i < text.length(); i++)
           {
               // check the direction of flow
               // reverse the direction if we've just
               // filled the top or bottom rail
               if (row == 0 || row == key-1)
                dir_down = !dir_down;//we change dir down when either we are at end row of matrix or first
                //System.out.println("I="+i+" Row="+row+" Dir="+dir_down);

               // fill the corresponding alphabet
//          rail[row][col++] = text[i];
               rail[row][col++] = text.charAt(i);
               //System.out.println("Row="+row+" COl="+col);
               // find the next row using direction flag
//          dir_down?row++ : row--;
               if(dir_down) {
                row++;
               }
               else {
                row--;
               }
           }

           //now we can construct the cipher using the rail matrix
           String result="";
           for (int i=0; i < key; i++)
               for (int j=0; j < text.length(); j++) {
                //System.out.print(rail[i][j]);
                if (rail[i][j]!='\n')
                    //result.push_back(rail[i][j]);
                                result=result+rail[i][j];}
```

```
          return result;
        }
//      String decryptRailFence(String text, int key) {
//
//      }
        String decryptRailFence(String cipher, int key)
        {
          // create the matrix to cipher plain text
          // key = rows , length(text) = columns
              char rail[][]=new char[key][(cipher.length())];

          // filling the rail matrix to distinguish filled
          // spaces from blank ones
          for (int i=0; i < key; i++)
            for (int j=0; j < cipher.length(); j++)
              rail[i][j] = '\n';

          // to find the direction
          boolean dir_down=true;

          int row = 0, col = 0;

          // mark the places with '*'
          for (int i=0; i < cipher.length(); i++)
          {
            // check the direction of flow
            if (row == 0)
              dir_down = true;
            if (row == key-1)
        dir_down = false;

    // place the marker
    rail[row][col++] = '*';

    // find the next row using direction flag
   // dir_down?row++ : row--;
    if(dir_down) {
        row++;
    }
    else {
        row--;
    }
}

// now we can construct the fill the rail matrix
int index = 0;
for (int i=0; i<key; i++)
    for (int j=0; j<cipher.length(); j++)
        if (rail[i][j] == '*' && index<cipher.length())
            rail[i][j] = cipher.charAt(index++);



// now read the matrix in zig-zag manner to construct
// the resultant text
String result="";

row = 0;
col = 0;
for (int i=0; i< cipher.length(); i++)
```

```
        {
          // check the direction of flow
          if (row == 0)
            dir_down = true;
          if (row == key-1)
            dir_down = false;

          // place the marker
          if (rail[row][col] != '*') {
            result=result+(rail[row][col++]);}

          // find the next row using direction flag
          //dir_down?row++: row--;
          if(dir_down) {
            row++;
          }
          else {
            row--;
          }
        }
        return result;
      }

}
```

4->RSA

```
import java.lang.Math;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Random;
public class crnsp4 {
int gcd(int a, int b)
{
 // Everything divides 0
 if (a == 0)
 return b;
 if (b == 0)
 return a;

 // base case
 if (a == b)
 return a;

 // a is greater
 if (a > b)
 return gcd(a-b, b);
 return gcd(a, b-a);
}
public static BigInteger largePrime(int bits) {
Random randomInteger = new Random();
BigInteger largePrime = BigInteger.probablePrime(bits, randomInteger);
return largePrime;
}
public static void main(String[] args) {
BigInteger p1=largePrime(1024);
 BigInteger p2=largePrime(1024);
 //System.out.println("1024bit prime number1="+p1);
 //System.out.println("1024bit prime number2="+p2);
// BigInteger a=largePrime(1024);
// BigInteger b=largePrime(1024);
// TODO Auto-generated method stub
crnsp4 p4 =new crnsp4();
double message=15;
// double str1 = Double.parseDouble(message);
System.out.println("Message="+message);
int a=61;
int b=53;
// BigInteger m1=new BigInteger("-1");
// BigInteger n=a.multiply(b);
// BigInteger euler=(a.subtract(m1)).multiply((b.subtract(m1)));

int n=a*b;
int euler=(a-1)*(b-1);
int enc=2,temp;
while(enc<euler) {
temp=p4.gcd(enc,euler);
if(temp==1) {
break;
}
else {
enc++;
}
}
//System.out.println("Enc="+enc);
int d=0;
for(int i=0;i<=9;i++) {
int x=1+(i*euler);
if(x%enc==0) {
d=x/enc;
break;
}
}
double c=Math.pow(message, enc)%n;
System.out.println("Encrypted="+c);
//double m=c.pow(d).mod(n);
BigInteger C = BigDecimal.valueOf(c).toBigInteger();
BigInteger N = BigInteger.valueOf(n);
BigInteger msgback = (C.pow(d)).mod(N);
// k = 2; // A constant value
 // double d1 = (1 + (k*euler))/enc;
// double m=Math.pow(c, d1)%n;
 System.out.println("Decrypted="+msgback);
 System.out.println("\n19DCS060\nPriyanshu Maurya");


}
}
```

p->7 Nmap

TCP scan for Open port

```
nmap -sT -p 445 192.168.1.102
```

TCP scan for closed port

```
nmap -sT -p 3389 192.168.1.102
```

Stealth scan for Open port

```
nmap -sS -p 22 192.168.1.102
```

Stealth scan for closed port

```
nmap -sS -p 3389 192.168.1.102
```

Fin scan for open port

```
nmap -sF -p 22 192.168.1.102
```

Fin scan for closed port

```
nmap -sF -p 3389 192.168.1.102
```

Null scan for open port

```
nmap -sN -p 22 192.168.1.102
```

Null scan for closed port

```
nmap -sN -p 3389 192.168.1.102
```

UDP scan for Open Port

```
nmap -sU -p 161 192.168.1.119
```

UDP scan for closed port

```
nmap -sU -p 53 192.168.1.119
```

Xmas scan for open port

```
nmap -sX -p 22 192.168.1.102
```

Xmas scan for closed port

```
nmap -sX -p 3389 192.168.1.102
```

```
Dmitry -winspo demo.txt hackthissite.org
```

```
Ua-tester -u www.charusta.ac.in -d  M D
```

```
Whatweb -v  www.charusta.ac.in
```