**Lab Practical #11:**

1. Implementation of RSA Algorithm technique.

**Program:**

import java.math.BigInteger;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class RSA {

    public static void main(String[] args) {

        double mul = 1;

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter P :  ");

        int p = sc.nextInt();

        System.out.println("Enter Q :  ");

        int q = sc.nextInt();

        System.out.println("Enter Message :");

        int m = sc.nextInt();

        int n = p\*q;

        int phiOfN = (p-1)\*(q-1);

        int e = relativelyPrime(phiOfN);

        int d = findModularInverse(e, q,phiOfN);

        int interger = 2147483647;

        //Encryption ::: c = M^e % n

        BigInteger eBase = BigInteger.valueOf(m);

        BigInteger eExponent = BigInteger.valueOf(e);

        BigInteger eModulus = BigInteger.valueOf(n);

        BigInteger cipher = eBase.modPow(eExponent, eModulus);

        System.out.println("PU: ("+e + "," + n+")");

        System.out.println("PR: ("+d + "," + n+")");

        System.out.println("Cipher: " + cipher);

        //Decryption :

    BigInteger dBase = BigInteger.valueOf(Integer.parseInt(cipher.toString()));

        BigInteger dExponent = BigInteger.valueOf(d);

        BigInteger dModulus = BigInteger.valueOf(n);

        BigInteger decrypted = dBase.modPow(dExponent, dModulus);

        System.out.println("Plain: " + decrypted);

}

    public static boolean areRelativelyPrime(int a, int b) {

        while (b != 0) {

            int temp = b;

            b = a % b;

            a = temp;

        }

        return a == 1;

    }

    public static int relativelyPrime(int n) {

        int i = 2;

        while (true) {

            if (areRelativelyPrime(n, i)) {

                return i;

            }

            i++;

        }

    }

    public static int findModularInverse(int e, int q,int phiOfN) {

        for (int i = 1; i < phiOfN; i++) {

            double d = (1+(i\*phiOfN))/(double)e;

            if(d %1==0){

                return (int)d;

            }

        }

        return -1;

    }

}

            a = temp;

        }

        return a == 1;

    }

    public static int relativelyPrime(int n) {

        int i = 2;

        while (true) {

            if (areRelativelyPrime(n, i)) {

                return i;

            }

            i++;

        }

    }

    public static int findModularInverse(int e, int q,int phiOfN) {

        for (int i = 1; i < phiOfN; i++) {

            double d = (1+(i\*phiOfN))/(double)e;

            if(d %1==0){

                return (int)d;

            }

        }

        return -1;

    }

}

**Output :**

Enter P :

17

Enter Q :

11

Enter Message :

88

PU: (3,187)

PR: (107,187)

Cipher: 44

Plain: 88

**Output:**