Lab 5

- **Program**

- **Write a Program to find the primitive roots for the Multiplicative Group with respect to Prime Modulus. Using that Implement Elgamal Cryptosystem.**

- **Code:**

```
import java.util.Scanner;
import java.util.Random;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import java.math.BigInteger;

public class ElgamalCryptosystem
{
   public long[] calculateGCD(long r1,long r2)
   {
      long arr[] = new long[2];
      long s1 = 0, s2 = 1, t1 = 1, t2 = 0, temp;
      while (r2 != 0)
      {
         long q = r1 / r2;
         long r = r1 % r2;

         r1 = r2;
         r2 = r;

         temp = s1;
         s1 = t1 - q * s1;
         t1 = temp;

         temp = s2;
         s2 = t2 - q * s2;
         t2 = temp;
      }
```

```
        arr[0] = r1;
        arr[1] = t1;
        return arr;
    }

    public long modMulInverse(long a,long mod)
    {
        long arr[] = calculateGCD(a, mod);
        long inverse = 0;

        if(arr[0] != 1)
        {
            System.out.println("Inverse of " + a +" is not possible!!! with respect to
Z"+mod);
            System.exit(0);
        }
        else
        {
            if(arr[1] < 0)
            {
                inverse = arr[1] + mod;
            }
            else
            {
                inverse = arr[1];
            }
        }
        return inverse;
    }

    public long modularExponentiation(long base,long exponent,long modulo)
    {
        long result = 1;

        base = base % modulo;

        while(exponent > 0)
        {
            if((exponent & 1) == 1)
                result = (result * base) % modulo;
```

```
        exponent >>= 1;
        base = (base * base) % modulo;
    }
    return result;
}

public boolean miilerRabinPrimalityTest(long d,long largeNum)
{
    long a = 2 + (int)(Math.random() % (largeNum - 4));

    long ans = modularExponentiation(a, d, largeNum);

    if(ans == 1 || ans == largeNum - 1)
        return true;

    while(d != largeNum - 1)
    {
        ans = (ans * ans) % largeNum;
        d *= 2;

        if (ans == 1)
            return false;
        if (ans == largeNum - 1)
            return true;
    }
    return false;
}
public boolean isPrimeNumber(long largeNum,int iterations)
{
    if(largeNum <= 1)
        return false;
    if(largeNum <= 3)
        return true;

    long d = largeNum - 1;

    while(d % 2 == 0)
        d /= 2;
```

```
    for(int i = 0; i <= iterations; i++)
    {
        if(!miilerRabinPrimalityTest(d, largeNum))
            return false;
    }
    return true;
}

public long power(long N, long P)
{
    if (P == 0)
        return 1;
    else
        return N * power(N, P - 1);
}
public long[] encryption(long message,long e1,long e2,long r,long longVal)
{
    long arr[] = new long[2];
    long c1 = modularExponentiation(e1, r, longVal);
    System.out.println(c1);
    long ans = power(e2, r);
    System.out.println(ans);
    long c2 = (ans * message) % longVal;
    System.out.println(c2);

    arr[0] = c1;
    arr[1] = c2;
    return arr;
}

public long decryption(long c1,long c2,long d,long longVal)
{
    long ans = power(c1, d);
    System.out.println(ans);

    long inverse = modMulInverse(ans, longVal);
    System.out.println(inverse);

    long message = (c2 * inverse) % longVal;
    System.out.println(message);
```

```
        return message;
    }

    public static void main(String []args)
    {
        ElgamalCryptosystem ecs = new ElgamalCryptosystem();
        Scanner sc = new Scanner(System.in);
        Random random = new Random();
        ArrayList<Long> groupMembers = new ArrayList<Long>();

        int  max = 2,iterations = 5,countPrimitiveRoots = 0;
        byte[] bytes = new byte[max];

        BigInteger largeNum;
        long longVal;

        System.out.println("Enter your message");
        long message = sc.nextLong();

        while(true)
        {
            random.nextBytes(bytes);
            largeNum = new BigInteger(bytes);
            //largeNum = new BigInteger("7");
            longVal = largeNum.longValue();

            if(ecs.isPrimeNumber(longVal, iterations))
                break;
            else
                ecs.isPrimeNumber(longVal, iterations);
        }
        System.out.println("Generated random prime number is
"+largeNum.abs());
        System.out.println("Members of (Z"+longVal+"*"+",*) are from
1..."+(longVal - 1));

        long phin = longVal - 1;
        long mod = longVal;

        System.out.println("Phi("+longVal+") = "+phin);
```

```
        for(long value  = 0; value <= (longVal - 2); value += 1)
           groupMembers.add(value + 1);

        Set<Long> primitiveRoots = new HashSet<Long>();
        ArrayList<Long> primitiveRootsArray = new ArrayList<Long>();

        for(int y = 1; y <= phin; y++)
        {
           long a = groupMembers.get(y - 1);
           for(int i = 1; i <= phin; i += 1)
           {
              long ans = ecs.modularExponentiation(a, i, mod);
              primitiveRoots.add(ans);
           }

           if(primitiveRoots.size() == phin)
           {
              countPrimitiveRoots += 1;
              primitiveRootsArray.add(a);
           }
           primitiveRoots.clear();
        }

        System.out.println("Group (Z"+longVal+"*"+",*) has total
"+countPrimitiveRoots+" primitive roots");

        //int randNum =  (int) ((Math.random() * (primitiveRootsArray.size() - 0
+ 1)) + 1);
        //System.out.println(randNum);
        long e1 = primitiveRootsArray.get(3);
        System.out.println("Randomly selected Primitive Root out of
"+countPrimitiveRoots+" Primitive Roots is "+e1);

        long d = 1 + (int) ((Math.random() * ((longVal - 2) - 1 + 1)) + 1);
        System.out.println("Randomly selected value for d from the rang 1 to
"+(longVal - 2)+" is "+d);

        long e2 = ecs.modularExponentiation(e1, d, longVal);
        int r =  (int) ((Math.random() * (groupMembers.size() - 0 + 1)) + 1);
```

```
        System.out.println("Generated public key is
("+e1+","+e2+","+longVal+")");
        System.out.println("Generated private key is "+d);
        System.out.println("Generated random number(r) from the group
(Z"+longVal+"*"+",*) is "+groupMembers.get(r));

        long arr[] = ecs.encryption(message, e1, e2, (long)r, longVal);
        long c1 = arr[0], c2 = arr[1];

        System.out.println("Generated cipher texts for the message "+message+"
are c1 = "+c1+" and c2 = "+c2);
        System.out.println("Decrypted message is "+ecs.decryption(c1, c2, d,
longVal));
    }
}
```

- **Output:**

```
E:\Jeet\D2D\Sem 6\NIS\Labs\Lab 5>java ElgamalCryptosystem
Enter your message
8
Generated random prime number is 31
Members of (Z31*,*) are from 1...30
Phi(31) = 30
Group (Z31*,*) has total 8 primitive roots
Randomly selected Primitive Root out of 8 Primitive Roots is 13
Randomly selected value for d from the rang 1 to 29 is 9
========== 22
Generated public key is (13,29,31)
Generated private key is 9
Generated random number(r) from the group (Z31*,*) is 8
22
17249876309
30
Generated cipher texts for the message 8 are c1 = 22 and c2 = 30
1207269217792
23
8
Decrypted message is 8
```