CRYPTOGRAPHY LABORATORY FILE
CS-511

Submitted to:
Dr. Samayveer Singh
Assistant Professor
Department of Computer Science

Submitted by:
Shashi Shekhar Azad
Roll No.: 23203029
M. Tech. CSE 1st Semester

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DR. B. R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR

Objective 1

Write a program to implement the encryption and decryption process using RSA Algorithm.

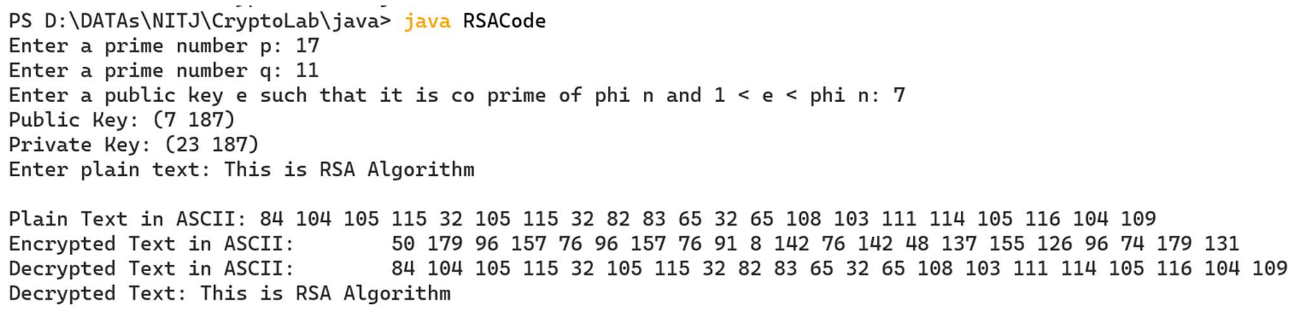
RSA Code:

```
import java.util.Scanner;
public class RSACode {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long p = 0, q = 0, E = 2, D = 0;
        System.out.print("Enter a prime number p: ");
        p = sc.nextLong();
        while (!isPrime(p)) {
            System.out.print(p + " is not a prime number, Enter prime number: ");
            p = sc.nextLong();
        }
        System.out.print("Enter a prime number q: ");
        q = sc.nextLong();
        while (!isPrime(q) || (p == q)) {
            System.out.print(q + " is not a prime number or p == q, Enter another number: ");
            q = sc.nextLong();
        }
        long n = p * q;
        long phi = (p - 1) * (q - 1);
        System.out.print("Enter a public key e such that it is co prime of phi n and 1 < e < phi n: ");
        E = sc.nextLong();
        while (!areCoprime(E, phi)) {
            System.out.print("Please select correct public key: ");
            E = sc.nextLong();
        }
        if (multiplicativeInverse(phi, E) != -1) {
            D = multiplicativeInverse(phi, E);
        } else {
            System.out.println("Private key can not be generated");
        }
        sc.nextLine();
        System.out.println("Public Key: (" + E + " " + n + ")");
        System.out.println("Private Key: (" + D + " " + n + ")");
        System.out.print("Enter plain text: ");
        String inpuString = sc.nextLine();
        byte[] plainText = inpuString.getBytes();
        System.out.print("\nPlain Text in ASCII: ");
        for (int i = 0; i < plainText.length; i++) {
            System.out.print(plainText[i] + " ");
        }
        long[] cipherText = new long[plainText.length];
        long[] decipheredText = new long[plainText.length];
        for (int i = 0; i < plainText.length; i++) {
            cipherText[i] = power(plainText[i], E, n);
        }
        System.out.print("\nEncrypted Text in ASCII:\t");
        for (int i = 0; i < cipherText.length; i++) {
            System.out.print(cipherText[i] + " ");
        }
        for (int i = 0; i < cipherText.length; i++) {
            decipheredText[i] = power(cipherText[i], D, n);
        }
        System.out.print("\nDecrypted Text in ASCII:\t");
        for (int i = 0; i < decipheredText.length; i++) {
            System.out.print(decipheredText[i] +" ");
        }
        System.out.print("\nDecrypted Text:\t");
        for (int i = 0; i < decipheredText.length; i++) {
            System.out.print((char) decipheredText[i]);
        }
        System.out.println("\n");
        sc.close();
    }
    public static boolean isPrime(long n) {
        if (n <= 1) {
            return false;
        }
        for (long i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
```

```
static long multiplicativeInverse(long a, long b) {
    long min = Math.min(a, b);
    long max = Math.max(a, b);
    a = max;
    b = min;
    long t1 = 0, t2 = 1;
    long t = 0, q = 0, r = 1;
    while (r != 0) {
        q = a / b;
        r = a % b;
        t = t1 - (t2 * q);
        a = b;
        b = r;
        t1 = t2;
        t2 = t;
    }
    if (t1 < 0) {
        t1 += t;
    }
    return t1;
}
static boolean areCoprime(long a, long b) {
    return gcd(a, b) == 1;
}
static long gcd(long a, long b) {
    if (a == 0)
        return b;
    return gcd(b % a, a);
}
static long power(long base, long exponent, long modulo) {
    long result = 1;
    base = base % modulo;
    while (exponent > 0) {
        if (exponent % 2 == 1) {
            result = (result * base) % modulo;
        }
        exponent = exponent >> 1;
        base = (base * base) % modulo;
    }
```

```
        return result;
    }
}
```

Output:



```
PS D:\DATAs\NITJ\CryptoLab\java> java RSACode
Enter a prime number p: 17
Enter a prime number q: 11
Enter a public key e such that it is co prime of phi n and 1 < e < phi n: 7
Public Key: (7 187)
Private Key: (23 187)
Enter plain text: This is RSA Algorithm

Plain Text in ASCII: 84 104 105 115 32 105 115 32 82 83 65 32 65 108 103 111 114 105 116 104 109
Encrypted Text in ASCII:        50 179 96 157 76 96 157 76 91 8 142 76 142 48 137 155 126 96 74 179 131
Decrypted Text in ASCII:        84 104 105 115 32 105 115 32 82 83 65 32 65 108 103 111 114 105 116 104 109
Decrypted Text: This is RSA Algorithm
```

Assignment 9

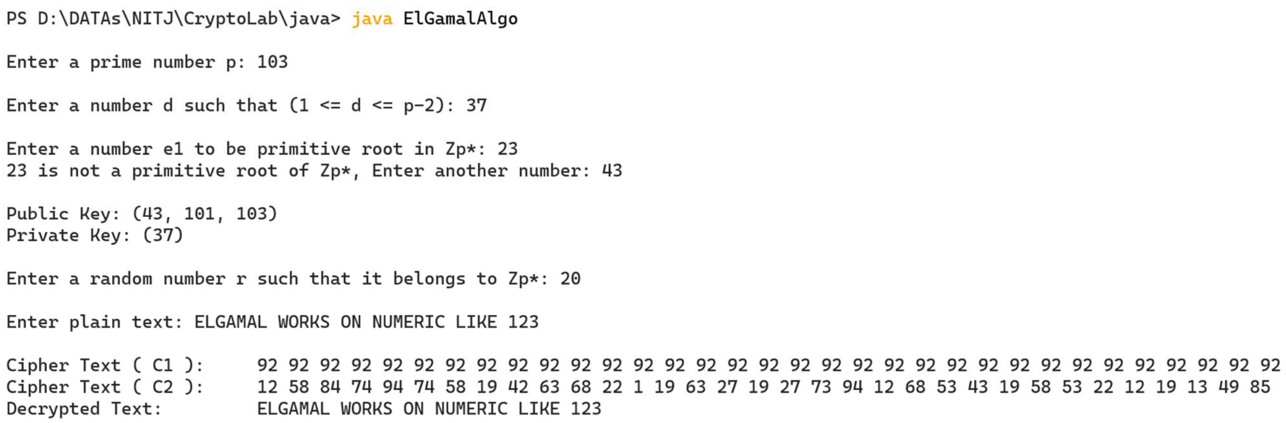
Write a program to implement encryption and decryption process using ElGamal Algorithm.

ElGamal Code:

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.Set;
class ElGamalAlgo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long p = 0, d = 0, e1 = 1, e2 = 0, r;
        char plainText[];
        System.out.print("\nEnter a prime number p: ");
        p = sc.nextLong();
        while (!isPrime(p)) {
            System.out.print(p + " is not a prime number, Enter prime number: ");
            p = sc.nextLong();
        }
        System.out.print("\nEnter a number d such that (1 <= d <= p-2): ");
        d = sc.nextLong();
        while (!isGroupMember(d, p)) {
            System.out.print(d + " is not a member of Zp*, Enter another number: ");
            d = sc.nextLong();
        }
        System.out.print("\nEnter a number e1 to be primitive root in Zp*: ");
        e1 = sc.nextLong();
        while (!isPrimitiveRoot(e1, p)) {
            System.out.print(e1 + " is not a primitive root of Zp*, Enter another number: ");
            e1 = sc.nextLong();
        }
        /// Key generation
        e2 = powerMod(e1, d, p);
        System.out.println("\nPublic Key: (" + e1 + ", " + e2 + ", " + p + ")");
        System.out.println("Private Key: (" + d + ")");
        /// Encryption process
        System.out.print("\nEnter a random number r such that it belongs to Zp*: ");
        r = sc.nextLong();
        while (!isGroupMember(r, p)) {
            System.out.print(r + " is not a member of Zp*, Enter another number: ");
            r = sc.nextLong();
        }
        sc.nextLine();
        System.out.print("\nEnter plain text: ");
        plainText = sc.nextLine().toUpperCase().toCharArray();
        long[] cipherText1 = new long[plainText.length];
        long[] cipherText2 = new long[plainText.length];
        long[] decipheredText = new long[plainText.length];
        /// Encryption process
        for (int i = 0; i < plainText.length; i++) {
            long[] ans = encrypt(plainText[i], e1, e2, r, p);
            cipherText1[i] = ans[0];
            cipherText2[i] = ans[1];
        }
        System.out.print("\nCipher Text ( C1 ):\t");
        for (int i = 0; i < cipherText1.length; i++) {
            System.out.print(cipherText1[i] + " ");
        }
        System.out.print("\nCipher Text ( C2 ):\t");
        for (int i = 0; i < cipherText2.length; i++) {
            System.out.print(cipherText2[i] + " ");
        }
        /// Decryption process
        for (int i = 0; i < plainText.length; i++) {
            decipheredText[i] = decrypt(cipherText1[i], cipherText2[i], d, p);
        }
        System.out.print("\nDecrypted Text:\t\t");
        for (int i = 0; i < decipheredText.length; i++) {
            System.out.print((char) (decipheredText[i] % 97));
        }
        System.out.println("\n");
        sc.close();
    }
    private static long[] encrypt(long pt, long e1, long e2, long r, long p) {
        long[] cipherText = new long[2];
        cipherText[0] = powerMod(e1, r, p);
        long x = powerMod(e2, r, p);
        cipherText[1] = (pt * x) % p;
        return cipherText;
```

```
}
private static long decrypt(long c1, long c2, long d, long p) {
    long deciphered = 0;
    long x = powerMod(c1, d, p);
    deciphered = (c2 * multiplicativeInverse(x, p)) % p;
    return deciphered;
}
public static boolean isPrime(long n) {
    if (n <= 1) {
        return false;
    }
    for (long i = 2; i <= Math.sqrt(n); i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
public static boolean isGroupMember(long n, long p) {
    if (n < 1 || n > p || p % n == 0) {
        return false;
    }
    return true;
}
public static boolean isPrimitiveRoot(long g, long p) {
    if (g < 2 || p < 2) {
        return false;
    }
    if (!isCoprime(g, p)) {
        return false;
    }
    long phi = p - 1;
    Set<Long> factors = new HashSet<>();
    long tempPhi = phi;
    for (int i = 2; i <= Math.sqrt(phi); i++) {
        if (tempPhi % i == 0) {
            factors.add((long) i);
            while (tempPhi % i == 0) {
                tempPhi /= i;
            }
        }
    }
    if (tempPhi > 1) {
        factors.add(tempPhi);
    }
    for (long factor : factors) {
        if (powerMod(g, phi / factor, p) == 1) {
            return false;
        }
    }
    return true;
}
public static boolean isCoprime(long a, long b) {
    while (b != 0) {
        long temp = b;
        b = a % b;
        a = temp;
    }
    return a == 1;
}
public static long powerMod(long base, long exponent, long modulus) {
    long result = 1;
    while (exponent > 0) {
        if (exponent % 2 == 1) {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exponent /= 2;
    }
    return result;
}
static long multiplicativeInverse(long a, long b) {
    long min = Math.min(a, b);
    long max = Math.max(a, b);
    a = max;
    b = min;
    long t1 = 0, t2 = 1;
    long t = 0, q = 0, r = 1;
    while (r != 0) {
      q = a / b;
      r = a % b;
      t = t1 - (t2 * q);
      a = b;
      b = r;
      t1 = t2;
      t2 = t;
    }
    if (t1 < 0) {
      t1 += t;
    }
    return t1;
  }
}
```

Output:



```
PS D:\DATAs\NITJ\CryptoLab\java> java ElGamalAlgo

Enter a prime number p: 103

Enter a number d such that (1 <= d <= p-2): 37

Enter a number e1 to be primitive root in Zp*: 23
23 is not a primitive root of Zp*, Enter another number: 43

Public Key: (43, 101, 103)
Private Key: (37)

Enter a random number r such that it belongs to Zp*: 20

Enter plain text: ELGAMAL WORKS ON NUMERIC LIKE 123

Cipher Text ( C1 ):      92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92
Cipher Text ( C2 ):      12 58 84 74 94 74 58 19 42 63 68 22 1 19 63 27 19 27 73 94 12 68 53 43 19 58 53 22 12 19 13 49 85
Decrypted Text:          ELGAMAL WORKS ON NUMERIC LIKE 123
```