

**Tutorial #4**  
**Security in Computing COSC2356/2357**

**Q1**

Alice has a message  $M=50$  to send to Bob securely using ElGamal encryption algorithm. Bob chooses  $p=71$ ,  $g=69$ ,  $x=11$ . Alice chooses  $r=23$ . Show the encryption and decryption steps.

**ANSWER:**

**Q2**

Alice has a message  $M=1$  to send to Bob securely using Paillier encryption algorithm. Bob chooses  $p=5$ ,  $q=7$ , and selects an integer  $g=164$ . Alice selects a random number  $r=17$ . Show the encryption and decryption steps.

**ANSWER:**

### Task 1 (Demonstration of ElGamal Cryptosystems using Java and Python programming Language)

Log in to your CANVAS and download the code of a very simple RSA encryption scheme. There are two files in the CANVAS: **ElGamalEncryption.java** and **ElGamalEncryption.py**. Download and run either of the file based on your familiarity on **JAVA** and **Python** programming language. Use **Q2** and **Q3** for required information. The codes are given as follows:

\*\*\*\*\* **JAVA CODE :: File Name: ElGamalEncryption.java**\*\*\*\*\*

```
import java.math.BigInteger;
import java.util.Scanner;

public class ElGamalEncryption {
    public static void main(String[] args) {
        // Declare required parameters as BigInteger objects
        BigInteger m;          // Input Message
        BigInteger p;          // 1st public parameter of Bob
        BigInteger g;          // 2nd public parameter of Bob
        BigInteger x;          // Bob's Private Key
        BigInteger y;
        BigInteger r;          // Chosen Random Number
        BigInteger k,k_bob;
        BigInteger k_Inv;      //  $k^{-1} \bmod p$ 
        BigInteger C1;         // 1st Ciphertext C1
        BigInteger C2;         // 2nd Ciphertext C2
        BigInteger M;          // Decrypted Message

        // Take required parameter as Input

        Scanner input = new Scanner(System.in); // initialize Scanner
        object for taking input

        System.out.println("Welcome to ElGamal Encryption Program
!!!");

        System.out.println("BOB generates Public and Private Key.");
        System.out.println("Please provide the following information
as Integer !!!");
        System.out.print("Enter the value of 'p' (as Integer): ");
        String p_str = input.next();
        System.out.print("Enter the value of 'g' (as Integer): ");
        String g_str = input.next();
        System.out.print("Enter the value of 'x' (as Integer): ");
        String x_str = input.next();

        // assign values to bi1, bi2
        p = new BigInteger(p_str);
        g = new BigInteger(g_str);
        x = new BigInteger(x_str);
        System.out.println();
        //System.out.println("Here is the detail Solution: ");
        y = g.modPow(x, p);
        System.out.println("The value of 'y' is: "+y);
        System.out.println("The Private-Key x := "+x);
        System.out.println("The Public-Key (p,y,g) := (" +p+", "+y+",
"+g+") is sent to ALICE.");
        System.out.println("ALICE Encrypts the Message, m ");
```

```

        System.out.print("Enter the message (as Integer) to encrypt, m
:= ");
        String m_str = input.next();
        m = new BigInteger(m_str);
        System.out.print("Enter the value of 'r' (as Integer): ");
        String r_str = input.next();
        r = new BigInteger(r_str);
        k = y.modPow(r, p);
        System.out.println("The value of 'k' is: "+k);
        C1 = g.modPow(r, p);
        C2 = (m.multiply(k)).mod(p);
        System.out.println("Ciphertexts, C1 := "+C1);
        System.out.println("Ciphertexts, C2 := "+C2);
        System.out.println("\n(C1, C2)=( "+C1+", "+C2+" ) are sent to
BOB.");
        System.out.println("BOB decrypts the Original Message.");
        k_bob = C1.modPow(x, p);
        System.out.println("The value of 'k' @BOB is: "+k_bob);
        k_Inv = k.modInverse(p);
        System.out.println("The value of 'k^-1' is: "+k_Inv);
        M = (k_Inv.multiply(C2)).mod(p);
        System.out.println("Extracted Message, M := "+M);
    }
}

```

\*\*\*\*\* PYTHON CODE :: File Name: ElGamalEncryption.py\*\*\*\*\*

# Function Declarations

# Calculates Euclidian GCD, return (g, x, y)...  $a*x + b*y = \text{gcd}(x, y)$

```

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

```

```

def modinv(a, m):                # Calculates Inverse Mod
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

```

# End of Function Declarations

```

print ("Welcome to ElGamal Encryption Program !!!")
print ("Please provide the following information as Integer.");

```

```

p_str = input("Enter the value of 'p' (as Integer): ")
p = int (p_str)

```

```

g_str = input("Enter the value of 'g' (as Integer): ")
g = int (g_str)

```

```

x_str = input("Enter the value of 'x' (as Integer): ")
x = int (x_str)

```

```

y = pow(g,x,p)

```

```

print ("The value of 'y' is: "+ str(y))

```

```

print ("The Private-Key x := "+str(x))

print ("The Public-Key (p,y,g) := (" +str(p)+", "+str(y)+", "+str(g)+") is
sent to ALICE.")

print ("ALICE Encrypts the Message, m ")

m_str = input ("Enter the message (as Integer) to encrypt, m := ")

m = int (m_str)

r_str = input ("Enter the value of 'r' (as Integer): ")

r = int (r_str)

k = pow(y,r,p)

print ("The value of 'k' is: "+ str(k))

C1 = pow(g,r, p)

C2 = (m * k) % p

print("Ciphertexts, C1 := "+str(C1))
print("Ciphertexts, C2 := "+str(C2))
print("\n(C1, C2)=(" +str(C1)+", "+str(C2)+") are sent to BOB.")

print("BOB decrypts the Original Message.")

k_bob = pow(C1,x, p)

print("The value of 'k' @BOB is: "+str(k_bob))

k_Inv = modinv(k,p)

print("The value of 'k^-1' is: "+str(k_Inv));

M = (k_Inv * C2) % p
print ("Extracted Message, M := "+str(M))

```

## Task 2 (Demonstration of Paillier Cryptosystems using Java and Python programming Language)

Log in to your CANVAS and download the code of a very simple RSA encryption scheme. There are two files in the CANVAS: **RSAEncryption.java** and **RSAEncryption.py**. Download and run either of the file based on your familiarity on **JAVA** and **Python** programming language. Use **Q2** and **Q3** for required information. The codes are given as follows:

```
***** JAVA CODE :: File Name: PaillierEncryption.java*****
import java.math.BigInteger;
import java.util.Scanner;

public class PaillierEncryption {
    public static void main(String[] args) {
        // Declare required parameters as BigInteger objects
        BigInteger m;        // Input Message
        BigInteger p;        // 1st prime number
        BigInteger q;        // 2nd prime number

        BigInteger n;        // Key parameter "n"
        BigInteger g;        // Chosen Integer

        BigInteger lambda; // Private Key parameter
        BigInteger u;
        BigInteger k;
        BigInteger meu;    // Private Key parameter
        BigInteger r;      // Random Integer
        BigInteger C;      // Encrypted Message
        BigInteger M;      // Decrypted Message

        // Take required parameter as Input

        Scanner input = new Scanner(System.in); // initialize Scanner
        object for taking input

        System.out.println("Welcome to Paillier Encryption Program
!!!");
        System.out.println("BOB generates Public and Private Key.");

        System.out.println("Please provide the following information
as Integer !!!");

        System.out.print("Enter the value of 'p' (as Integer): ");
        String p_str = input.next();

        System.out.print("Enter the value of 'q' (as Integer): ");
        String q_str = input.next();

        System.out.print("Enter the value of 'g' (as Integer): ");
        String g_str = input.next();

        // assign values to bi1, bi2

        p = new BigInteger(p_str);
        q = new BigInteger(q_str);
```

```

g = new BigInteger(g_str);

System.out.println();

System.out.println("Here is the detail Solution: ");

n = p.multiply(q);
System.out.println("The value of 'n' is: "+n);

System.out.println("The Public-Key (n,g) := (" +n+", "+g+") is
sent to ALICE.");
System.out.println("BOB computes Private-Key parameters
(Lambda, Meu) ");

BigInteger p_minus_1 = p.subtract(new BigInteger("1"));
BigInteger q_minus_1 = q.subtract(new BigInteger("1"));

lambda =
p_minus_1.multiply(q_minus_1.divide(p_minus_1.gcd(q_minus_1)));
System.out.println("The value of 'Lambda' is: "+lambda);

u = g.modPow(lambda, n.multiply(n));

BigInteger L_of_u = u.subtract(new BigInteger("1")).divide(n);
k = L_of_u;
System.out.println("The value of k is := "+k);

meu = k.modInverse(n);
System.out.println("The value of 'Meu' is := "+meu);

System.out.println("BOB's Private-Key (Lambda,Meu) :=
("+lambda+", "+meu+)");

System.out.println("ALICE Encrypts the Message, m ");

System.out.print("Enter the message (as Integer) to encrypt, m
:= ");

String m_str = input.next();
m = new BigInteger(m_str);

System.out.print("Enter the value of 'r' (as Integer): ");
String r_str = input.next();
r = new BigInteger(r_str);

BigInteger temp1 = g.modPow(m, n.multiply(n));
BigInteger temp2 = r.modPow(n, n.multiply(n));

C = temp1.multiply(temp2).mod(n.multiply(n));

System.out.println("Ciphertext, C := "+C);
System.out.println("BOB decrypts the Original Message.");

BigInteger u1 = C.modPow(lambda, n.multiply(n));

```

```

        BigInteger L_of_u1 = u1.subtract(new
BigInteger("1")).divide(n);

        M = L_of_u1.multiply(meu).mod(n);

        System.out.println("Extracted Message, M := "+ M);
    }
}

***** PYTHON CODE :: File Name: PaillierEncryption.py*****
# Function Declarations

# Calculates Euclidian GCD, return (g, x, y) ... a*x + b*y = gcd(x, y)
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

def modinv(a, m):                # Calculates Inverse Mod
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

# define a function
def lcm(x, y):
    """This function takes two integers and returns the L.C.M."""

    # choose the greater number
    if x > y:
        greater = x
    else:
        greater = y

    while(True):
        if((greater % x == 0) and (greater % y == 0)):
            LCM = greater
            break
        greater += 1

    return LCM
# End of Function Declarations

print("Welcome to Paillier Encryption Program !!!")
print("BOB generates Public and Private Key.")

print("Please provide the following information as Integer !!!")

p_str = input("Enter the value of 'p' (as Integer): ")
p = int (p_str)

q_str = input("Enter the value of 'q' (as Integer): ")
q = int (q_str)

```



```

g_str = input("Enter the value of 'g' (as Integer): ")
g = int (g_str)
n = p * q
print("The value of 'n' is: "+str(n))

print("The Public-Key (n,g) := (" +str(n)+", " +str(g)+") is sent to ALICE.")
print("BOB computes Private-Key parameters (Lambda, Meu) ")

L = lcm((p-1),(q-1))

print("The value of 'Lambda' is: "+str(L))

u = pow(g,L,n*n)

k = int ((u-1)/n)

print("The value of k is := "+str(k))

meu = int(modinv(k,n))

print("The value of 'Meu' is := "+str(meu))

print("BOB's Private-Key (Lambda,Meu) := (" +str(L)+", " +str(meu)+")");

print("ALICE Encrypts the Message, m ");

m_str = input("Enter the value of Message 'm' (as Integer): ")
m = int (m_str)

r_str = input("Enter the value of 'r' (as Integer): ")
r = int (r_str)

C = int (((g**m)*(r**n)) % (n*n))

print("Ciphertext, C := "+str(C));
print("BOB decrypts the Original Message.")

u1 = pow(C,L,n*n)

L_of_u1 = int ((u1-1)/n)

M = int ((L_of_u1 * meu ) % n)

print("Extracted Message, M := "+ str(M))

```

## Task 2 (RSA Publik-Key Encryption and Decryption using OpenSSL).

### Windows OS:

Please download and install OpenSSL from the following link:

<http://downloads.sourceforge.net/gnuwin32/openssl-0.9.8h-1-bin.zip>

Unzip the file and run “openssl.exe”.

### Linux or recent MacOS:

You already have OpenSSL. Open terminal and run the following command:

```
>openssl
```

You are ready to run OpenSSL commands.

### Using RSA Algorithm using OpenSSL

Assume that you have a plain-text file, called “**plain-text.txt**”, with your *name* and *student ID* in that file. Apply OpenSSL’s *RSA algorithm* for a **2048-bit key** to encrypt the “**plain-text.txt**” file. Say, the name of the cipher-text file is “**cipher-text.txt**”. Now, decrypt the “**cipher-text.txt**” file using OpenSSL’s *RSA algorithm*.

### **Answer:**

**Step-1:** Generate 2048-bit RSA private/public key pairs using the following command. The private key is stored in **key.pem**.

```
genrsa -out key.pem 2048
```

You should get the following outputs in the console:

**Generating RSA private key, 2048 bit long modulus**

.....+++

.....+++

**e is 65537 (0x010001)**

A file called **key.pem** is generated, which is the **Private-key**.

**Step-2:** See the detail key information using the following command:

```
rsa -in key.pem -text -noout
```

You should get the following output:

**Private-Key: (2048 bit)**

**modulus:**

```
00:bb:03:20:79:2e:2a:4e:8c:3c:66:ba:4f:e8:5c:
d0:28:8e:ed:4f:23:96:2c:6e:a8:60:92:99:05:c7:
43:99:3c:3c:f1:e4:71:d3:f0:4a:41:db:32:6c:2a:
eb:8d:b6:c9:e7:49:1a:4b:37:a5:49:41:1f:f5:90:
bd:ae:b6:87:ba:65:45:c8:fc:5a:22:c4:fc:2f:90:
```

f7:51:9b:83:dc:25:63:58:6e:de:22:db:7b:89:44:  
25:75:50:b4:ac:eb:ce:d8:88:2c:2d:34:e1:aa:26:  
a7:e6:a2:f3:8a:fe:45:be:24:e7:04:6e:7d:6c:4c:  
6f:34:c8:69:7a:9e:6f:f3:fd:47:3c:d9:a8:30:5b:  
2b:20:fa:a3:a0:86:4e:4f:3b:68:d1:90:92:cb:dc:  
bc:a2:f0:19:af:55:4b:f0:1c:2e:4c:80:ee:71:ef:  
02:0d:ff:e2:67:2b:01:ac:f9:6e:57:75:e2:79:2b:  
75:01:f7:54:15:8b:a7:5e:a2:aa:73:41:60:3f:f2:  
8e:a2:71:57:90:73:22:d9:66:52:fc:33:7a:19:40:  
d6:1f:14:88:db:2d:d5:d6:5d:59:37:fe:8d:9c:15:  
4f:19:5b:65:d7:0c:40:1b:9b:53:1e:46:6a:62:de:  
1e:56:96:38:0f:7f:09:76:e0:c8:6e:c6:97:13:d2:  
93:e5

publicExponent: 65537 (0x10001)

privateExponent:

5e:7b:07:ee:f1:09:e2:c1:2a:ca:e3:99:f7:54:dc:  
bd:80:e8:17:b1:6c:ef:69:c0:9b:79:b4:e1:9c:78:  
64:74:70:7d:ec:e2:2d:27:1a:fd:06:97:04:da:f2:  
42:98:74:8c:ea:fb:e3:c0:6b:3b:05:31:f6:48:77:  
ec:4a:bf:6b:c6:3a:69:7e:44:b3:88:3d:b8:72:4e:  
e0:e5:e6:ca:54:01:4a:ee:48:3f:e8:0f:13:9c:60:  
28:52:eb:d4:e9:15:89:83:d2:7d:cc:57:ae:34:f5:  
62:aa:34:cc:a6:05:ea:38:8e:96:48:94:09:20:dc:  
96:18:22:62:16:a5:8c:e8:2d:7d:45:9b:c8:2e:13:  
e4:47:1f:82:5e:3a:8c:e4:a5:72:57:74:bf:e7:1b:  
98:30:88:91:b9:d9:ed:f3:d2:59:d7:11:11:e5:61:  
e8:6a:e1:82:7a:da:73:71:27:58:3a:84:78:82:86:  
8e:26:c9:ad:c9:34:6a:5e:95:c3:3c:c9:ce:c9:a4:  
86:fd:38:8d:93:66:62:4d:9d:f2:68:a4:33:de:01:  
43:c8:5b:27:7a:a7:4b:73:58:8d:d3:00:71:6e:8f:  
66:77:41:1d:e0:d9:ff:c3:ba:fe:c6:0a:4d:a6:2b:  
2c:9a:f2:eb:70:fd:1e:bc:d0:1f:5e:af:fd:da:ae:  
49

prime1:

00:ea:01:ab:d7:d3:0e:d5:5b:eb:8a:c1:c7:fd:bb:  
43:c6:7a:97:b6:4e:9e:0e:1e:c3:23:7c:7b:91:67:  
a0:cf:a8:40:48:ec:18:8c:10:b1:38:11:7c:3f:fd:  
09:95:7f:a2:1f:fa:4b:af:6e:2f:7b:f6:c0:d3:93:  
bf:2d:40:18:a1:8a:48:72:99:28:66:b1:32:cb:7b:  
c0:ef:3d:b1:c6:90:01:bd:d2:dc:24:21:31:40:9d:  
ab:13:c1:9b:f8:98:1d:df:37:a0:d8:a4:33:f8:cd:  
66:27:07:61:de:98:03:fe:68:ad:14:8f:93:ed:1e:  
14:f9:74:d5:c2:17:4e:da:d7

prime2:

00:cc:96:be:f6:6f:fe:a9:29:88:9c:9f:bb:33:10:  
94:4b:16:20:50:e3:85:b1:c8:d9:38:6d:60:83:18:  
fd:c1:a8:5e:cf:1c:ca:3b:0f:29:62:17:fa:e4:66:  
98:98:65:42:26:26:8d:c2:92:74:b2:9d:d9:93:fa:  
3e:f4:8c:a0:98:64:e3:09:4e:11:a9:7c:64:92:a3:  
dd:c4:9b:a5:aa:39:1c:c9:b2:a7:76:17:ea:01:4f:

```

af:90:af:10:09:09:1b:b1:58:c3:32:ed:c4:e7:51:
71:12:86:31:19:19:e9:f0:08:56:48:d3:68:38:f0:
f1:6c:c0:05:f5:cf:ba:0b:a3
exponent1:
00:cf:94:9b:f3:e0:6e:10:26:72:53:ac:82:d4:3a:
02:6d:56:e2:ad:fe:1f:87:37:12:b3:b0:01:8d:82:
f7:cc:3d:dc:88:d3:a7:12:d8:db:dc:78:e6:57:7d:
07:bb:6e:75:4b:18:a5:7b:01:ab:6d:b3:fe:69:b1:
6e:ad:9d:66:3c:26:87:0d:e1:7f:4d:59:73:4d:be:
81:ef:b8:32:b3:89:9b:81:e0:43:18:69:b9:5f:30:
7e:4a:10:3d:63:d0:cc:ee:ee:51:e8:dc:00:9e:7c:
d6:59:58:db:20:b2:89:18:6d:92:db:e2:61:be:be:
28:ad:01:4f:7d:d5:5f:46:11
exponent2:
00:c9:42:09:fd:37:d3:16:ea:0a:bf:b8:ca:58:c3:
98:7d:fc:f8:31:5a:80:ec:91:9e:4e:4a:1a:c5:1c:
52:94:ad:63:06:ef:55:69:9f:d2:9f:f2:e3:16:c8:
6e:98:8c:13:f4:9f:bc:98:89:a6:4f:07:c5:40:32:
ce:b7:97:97:6c:12:e2:dd:06:75:8d:7b:17:1c:c2:
22:a9:04:4c:86:15:c4:e2:0d:e3:7a:e2:af:8a:36:
af:88:ef:0e:21:35:5a:8e:ad:b8:e8:62:ca:6e:9b:
c9:55:e5:b8:6a:ee:f9:18:ed:ba:a3:cd:84:1b:6f:
ba:af:b6:7e:a6:7f:80:8f:6d
coefficient:
43:d5:50:4f:d4:97:2b:51:70:48:3d:b3:5c:d5:83:
2d:05:dc:cf:2c:24:85:da:81:32:71:79:66:c5:bd:
90:88:dd:8d:c1:ae:25:1b:9e:93:61:37:90:ad:9c:
c0:82:7e:a2:fa:56:a2:6e:fe:bf:f6:d8:32:bf:31:
57:7a:f8:cb:f1:8c:2c:c4:99:c1:de:d3:a5:e1:39:
a4:29:9e:f6:1f:17:ab:9f:fa:5b:e9:a4:09:06:ea:
09:a4:f7:b2:1d:72:d5:ba:6a:6b:da:8f:a3:42:ae:
29:2f:b1:03:cf:f5:f4:3c:23:32:81:0f:69:e5:4f:
57:86:47:75:95:5f:d3:52

```

**Step-3:** Extract RSA Public key from the private key using the following command:

```
rsa -in key.pem -pubout -out pub-key.pem -text
```

A file called **pub-key.pem** is generated, which is the public-key.

**Step-4:** Encrypt the plain-text file “**plain-text.txt**” with the public key **pub-key.pem** using the following command:

```
rsautl -in plain-text.txt -out cipher-text.txt -pubin -inkey pub-key.pem -encrypt
```

An encrypted file called **cipher-text.txt** is generated, which contains something similar to the followings:

