LAB06: RSA and Elgamal

By: BHUVANA PRABHA B(22BCE1639)

RSA CRYPTOSYSTEM

CODE:

1. RSATools/code.java source file

```
package RSATools;
import java.util.*;
public class code{

    public static int p, q, e;

    // Helper function to compute GCD
    public static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }//method end

        // Function to compute the modular inverse using the Extended
Euclidean Algorithm
    public static int modInverse(int e, int phi) {
        int t = 0, newT = 1;
        int r = phi, newR = e;

        while (newR != 0) {
            int quotient = r / newR;

            // Update t and newT
            int tempT = t;
            t = newT;
            newT = tempT - quotient * newT;

            // Update r and newR
            int tempR = r;
            r = newR;
            newR = tempR - quotient * newR;
        }
```

```
        // If r > 1, then e has no modular inverse
        if (r > 1) {
            throw new ArithmeticException("e has no modular inverse mod
phi(n)");
        }

        // Ensure d is positive
        if (t < 0) {
            t += phi;
        }

        return t;
    }//method end

    //method to check if a number is prime
    public static boolean isPrime(int num){
        for(int i = 2; i < num; i++){
            if(num % i == 0)
              return false;
        }//for end
        return true;
    }//method end

//To generate random numbers between 11 and 999 ( 2 to 3 digit varying random
prime numbers)
 public static int generateRandomPrime(Random rand) {
        int num;
        do {
            num = rand.nextInt(989) + 11; // Generates numbers from 11 to 999
        } while (!isPrime(num));
        return num;
    }


   //To calculate the value n (product of two randomly chosen prime numbers p
and q)
    public static void generatePQ(){
         Random rand = new Random();

        do {
            p = generateRandomPrime(rand);
            q = generateRandomPrime(rand);
        } while (p == q);  // Ensure p and q are distinct

        int n = p * q;
        System.out.println("Generated primes: p = " + p + ", q = " + q);
        System.out.println("n = " + n);
    }//method end
```

```
    //Method to generate 'e' (by taking phi(n) as the input)
        public static int generateE(int phi) {
        Random rand = new Random();
        int e;

        do {
            e = rand.nextInt(phi - 2) + 2; // Generates e in the range 2 to
phi(n)-1
        } while (gcd(e, phi) != 1);

        return e;
    }//method end

    //Returns the private key component 'd'
    public static int keyGeneration(){

    //generates two random prime integers p and q
       generatePQ();

    //Calculating the Euler's totient of n=pq (which is phi(n) = (p-1)*(q-1))
       int phin = (p-1)*(q-1);

    //Generates a random integer e such that 1 < e < phi(n) and gcd(e, phi(n))
= 1
    e = generateE(phin);

    System.out.println("Generated public key: e = " + e);

    //Calculating d (private key) which is the modular multiplicative inverse
of e(mod phi(n))
    int d = modInverse(e, phin);
    System.out.println("Generated private key component: d = " + d);

    System.out.println("Resulting PU: {e, n} is: { " + e + ", " + (p*q) + "
}");
    System.out.println("Resulting PR: {d, n} is: { " + d + ", " + (p*q) + "
}");

    return d;

    }//method end


    // Function to encode a character based on the given scheme (a-z) -> (00-
25), (A-Z) -> (26-51), (0-9) -> (52-61), Space -> 62
    public static int encodeChar(char ch) {
        if (ch >= 'a' && ch <= 'z') return ch - 'a'; // 00-25
        if (ch >= 'A' && ch <= 'Z') return ch - 'A' + 26; // 26-51
```

```java
        if (ch >= '0' && ch <= '9') return ch - '0' + 52; // 52-61
        if (ch == ' ') return 62; // Space -> 62
        return -1; // Invalid character
    }//method end

    // Function to decode a number back to its character
    public static char decodeCharacter(int num) {
        if (num >= 0 && num <= 25) {
            return (char) ('a' + num);  // 'a' to 'z'
        } else if (num >= 26 && num <= 51) {
            return (char) ('A' + (num - 26));  // 'A' to 'Z'
        } else if (num >= 52 && num <= 61) {
            return (char) ('0' + (num - 52));  // '0' to '9'
        } else if (num == 62) {
            return ' ';  // Space
        }
        return '?';  // Invalid character
    }//method end

    // Function to encode a string
    public static List<Integer> encodeText(String text) {
        List<Integer> encodedValues = new ArrayList<>();
        for (char ch : text.toCharArray()) {
            int encoded = encodeChar(ch);
            if (encoded != -1) {
                encodedValues.add(encoded);
            }
        }
        return encodedValues;
    }//method end

    // Function to determine the max block size
    public static int getMaxBlockSize(int n) {
        int maxEncodedValue = 62; // Highest encoding value
        int blockSize = 1;
        int value = maxEncodedValue;

        while (value * 100 + maxEncodedValue < n) { // Ensure next addition
doesn't exceed n
            value = value * 100 + maxEncodedValue; // Simulate adding another
encoded value
            blockSize++;
        }
        return blockSize;
    }//method end

    // Function to divide the encoded values into uniform blocks
```

```
    public static List<String> createBlocks(List<Integer> encodedValues, int
n) {
        List<String> blocks = new ArrayList<>();
        int blockSize = getMaxBlockSize(n); // Get max numbers per block
        System.out.println("The max number of characters per block is: " +
blockSize);

        StringBuilder block = new StringBuilder();
        for (int i = 0; i < encodedValues.size(); i++) {
            block.append(String.format("%02d", encodedValues.get(i))); //
Ensure two-digit format

            if ((i + 1) % blockSize == 0 || i == encodedValues.size() - 1) {
                blocks.add(block.toString());
                block.setLength(0); // Reset block
            }
        }//for end
        return blocks;
    }

    //Function to process the given string and convert it into blocks of equal
size(just before encryption)
    public static List<String> processString(String text, int n) {

        //First encode the text as per the mentioned scheme
        //(a-z) -> (00-25), (A-Z) -> (26-51), (0-9) -> (52-61), Space -> 62
        List<Integer> encodedValues = encodeText(text);

        //To determine the plain text block size (i.e. number of characters
per block)
        //int n = p * q;
        System.out.println("The value of n(=p*q) is(in processString(text)
after encoding text): " + n);

        //Divide the encoded values into uniform blocks
        List<String> blocks = createBlocks(encodedValues, n);
        System.out.println("The blocks of text are: " + blocks);

        return blocks;

    }//method end


    public static int fastExp(int base, int pow, int n){
            int f = 1;
            String b = Integer.toBinaryString(pow);
            for(int i = 0; i < b.length(); i++){
              f = (f*f) % n;
              if(b.charAt(i) == '1')
```

```
                f = (f*base) % n;
            }//for end
            return f;
    }//method end

    // Function to encrypt each plaintext block using C = M^e mod n
    public static List<Integer> encryptBlocks(List<String> plainBlocks, int e,
int n) {
        List<Integer> cipherBlocks = new ArrayList<>();
        for (String M : plainBlocks) {
            int m = Integer.parseInt(M);
            int C = fastExp(m, e, n); // Encrypt using RSA formula
            cipherBlocks.add(C);
        }
        return cipherBlocks;
    }//method end


    public static List<Integer> encryption(String text, int e, int n){

        List<String> blocks = processString(text, n);

        //System.out.println("The values of p and q are: " + p + ", " + q);
        List <Integer> cipherBlocks = encryptBlocks(blocks, e, n);

        System.out.println("Encrypted blocks: " + cipherBlocks);
        return cipherBlocks;
    }//method end

    // Function to decrypt each cipher block using M = C^d mod n
    public static List<Integer> decryptBlocks(List<Integer> cipherBlocks, int
d, int n) {
        List<Integer> plainBlocks = new ArrayList<>();
        for (int C : cipherBlocks) {
            int M = fastExp(C, d, n); // Decrypt using RSA formula
            plainBlocks.add(M);
        }
        return plainBlocks;
    }//method end

    // Function to decode a plaintext block into characters
    public static String decodePlaintextBlocks(List<Integer> plainBlocks) {
        StringBuilder decodedMessage = new StringBuilder();

        for (int block : plainBlocks) {
            // Extract individual encoded values from the block
            Stack<Integer> values = new Stack<>();
            while (block > 0) {
                values.push(block % 100);  // Extract last two digits
```

```
                block /= 100;  // Remove last two digits
            }

            // Convert extracted values to characters
            while (!values.isEmpty()) {
                decodedMessage.append(decodeCharacter(values.pop()));
            }
        }
        return decodedMessage.toString();
    }//method end

    public static String decryption(List<Integer> cipherBlocks, int e, int p,
int q){

    System.out.println("The value of e, p and q are : " + e + ", " + p + ", "
+ q);
    //get the private key component
    int d = modInverse(e, (p-1)*(q-1));

    //decrypt the cipher blocks
    List<Integer> plainBlocks = decryptBlocks(cipherBlocks, d, p*q);

    //decode the plain blocks
    String plainText = decodePlaintextBlocks(plainBlocks);
    //System.out.println("Decrypted text: " + plainText);
    return plainText;

    }//method end
}//class end
```

2. Server.java source file (Bob - Sender side)

```
3. import java.io.*;
4. import java.net.*;
5. import java.util.*;
6. import java.util.stream.Collectors;
7.
8. import RSATools.code;
9.
10.public class Server {
11.
12.    public static int e, n;
13.    public static void main(String[] args) {
14.        int port = 5000; // Define a port number
15.        try (ServerSocket serverSocket = new ServerSocket(port)) {
16.            System.out.println("Server started. Waiting for
   client...");
17.
```

```
18.             Socket socket = serverSocket.accept(); // Accept client
    connection
19.             System.out.println("Client connected!");
20.
21.             BufferedReader input = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
22.             PrintWriter output = new
    PrintWriter(socket.getOutputStream(), true);
23.             BufferedReader userInput = new BufferedReader(new
    InputStreamReader(System.in));
24.
25.             String message;
26.             while ((message = input.readLine()) != null) {
27.                 System.out.println("Client: " + message);
28.
29.                 //Get the public key components of Alice
30.
31.                 // Split using space as a delimiter
32.                 String[] publicKeys = message.trim().split("\\s+");
33.                 if (publicKeys.length == 2) {
34.                     try{
35.                         e = Integer.parseInt(publicKeys[0]);
36.                         n = Integer.parseInt(publicKeys[1]);
37.
38.                         System.out.println("Received Public Key Values:
    e = " + e + ", n = " + n);
39.                         //output.println("Public key received
    successfully.");
40.                     } catch (NumberFormatException e) {
41.                         System.out.println("Invalid public key
    format.");
42.                     }//try catch end
43.                 }//if end
44.
45.                 String M = userInput.readLine();
46.                 System.out.println("Generated message(Bob side): " +
    M);
47.
48.                 //Encrypt the message
49.                 List<Integer> cipherBlocks = code.encryption(M, e, n);
50.
51.                 // Convert the List<Integer> to a space-separated
    string
52.                 String encryptedMessage = cipherBlocks.stream()
53.                                         .map(String::valueOf)
54.                                         .collect(Collectors.joining("
    "));
55.
```

```
                // Send the encrypted message to the client
                output.println(encryptedMessage);

                if (message.equalsIgnoreCase("bye")) {
                    System.out.println("Client disconnected.");
                    break;
                }
            }//while end

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

3. Client.java source file (Alice receiver side)

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.stream.Collectors;

import RSATools.code;

public class Client {

    public static int p, q, e, d;
    public static void main(String[] args) {
        String serverAddress = "localhost"; // Server address
        int port = 5000;

        try (Socket socket = new Socket(serverAddress, port)) {
            System.out.println("Connected to the server.");

            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(),
true);
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));

            //Generate the public and private key components first
            d = code.keyGeneration();
            p = code.p;
            q = code.q;
            e = code.e;
```

```
            String message;
            while (true) {
                System.out.print("Enter message(send the public key value[e n]
if you are entering the message for the first time): ");
                message = userInput.readLine();
                output.println(message);

                if (message.equalsIgnoreCase("bye")) {
                    break;
                }//if end

                //Read the encrypted message from the server
                String encryptedMessage = input.readLine();
                if (encryptedMessage == null) {
        System.out.println("Server closed the connection.");
            } else {
                System.out.println("Server: " + encryptedMessage);
            }

            // Convert the space-separated string back to a List<Integer>
            List<Integer> cipherBlocks =
Arrays.stream(encryptedMessage.split(" "))
                                    .map(Integer::parseInt)
                                    .collect(Collectors.toList());

                //Decrypt the message
                String decryptedMsg = code.decryption(cipherBlocks, e, p, q);
                System.out.println("Decrypted message: " + decryptedMsg);
            }//while end

        } catch (IOException e) {
            System.err.println("Connection lost: " + e.getMessage());
        }
    }//Main end
}//class end
```

OUTPUT:

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\sixth_sem\Crypto
\RSA> java Server
Server started. Waiting for client...
Client connected!
Client: 973 23213
Received Public Key Values: e = 973, n = 23213
HELL79
Generated message(Bob side): HELL79
The value of n(=p*q) is(in processString(text) after encodi
ng text): 23213
The max number of characters per block is: 2
The blocks of text are: [3330, 3737, 5961]
Encrypted blocks: [22528, 4958, 510]
Client: bye
```

(Server-side terminal: Receives the public key components from the client side and uses it to encrypt the message to be sent)

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\sixth_sem\Crypto\RSA> java Client
Connected to the server.
Generated primes: p = 167, q = 139
n = 23213
Generated public key: e = 973
Generated private key component: d = 2425
Resulting PU: {e, n} is: { 973, 23213 }
Resulting PR: {d, n} is: { 2425, 23213 }
Enter message(send the public key value[e n] if you are entering the message for the
 first time): 973 23213
Server: 22528 4958 510
The value of e, p and q are : 973, 167, 139
Decrypted message: HELL79
Enter message(send the public key value[e n] if you are entering the message for the
 first time): bye
```

(Client-side terminal: generates the public and private key components and decrypts the message sent from the server side)

ELGAMAL CRYPTOSYSTEM

CODE:

1. Elgamal.java source file

```
2. package ElgamalTools;
3.
4. import java.util.ArrayList;
5. import java.util.HashSet;
6. import java.util.List;
7. import java.util.Random;
8. import java.util.Set;
9.
10.public class elgamal {
11.     public static int q, alpha;
```

```
12.
13.     //method to check if a number is prime
14.     public static boolean isPrime(int num){
15.         for(int i = 2; i < num; i++){
16.             if(num % i == 0)
17.               return false;
18.         }//for end
19.         return true;
20.     }//method end
21.
22.//Function to generate prime numbe q ka random value
23.  public static int generateRandomPrime(Random rand) {
24.         int num;
25.         do {
26.             num = rand.nextInt(100);
27.         } while (!isPrime(num));
28.         return num;
29.     }//method end
30.
31.     // Utility Function to check if a number is a primitive root of q
32.     public static boolean isPrimitiveRoot(int g, int q) {
33.         Set<Integer> set = new HashSet<>();
34.         int value = 1;
35.
36.         for (int i = 1; i < q; i++) {
37.             value = (value * g) % q;
38.             if (set.contains(value)) {
39.                 return false;
40.             }
41.             set.add(value);
42.         }
43.
44.         return set.size() == q - 1;
45.     }//method end
46.
47.       //Utility Function to find all primitive roots of q
48.     public static List<Integer> findPrimitiveRoots(int q) {
49.         List<Integer> primitiveRoots = new ArrayList<>();
50.         for (int i = 2; i < q; i++) {
51.             if (isPrimitiveRoot(i, q)) {
52.                 primitiveRoots.add(i);
53.             }
54.         }
55.         return primitiveRoots;
56.     }//method end
57.
58.     //Function to assign a value to the primitive root alpha
59.     public static void assignPrimitiveRoot(int q) {
```

```
60.         List<Integer> primitiveRoots = findPrimitiveRoots(q);
61.
62.         if(!primitiveRoots.isEmpty()){
63.             Random random = new Random();
64.             int randomIndex = random.nextInt(primitiveRoots.size());
65.             alpha = primitiveRoots.get(randomIndex);
66.
67.             System.out.println("Primitive root alpha: " + alpha);
68.         }//if end
69.         else{
70.             System.out.println("No primitive roots found for " + q);
71.         }//else end
72.     }//method end
73.
74.//Function to calculate the modular exponentiation
75.         public static int fastExp(int base, int pow, int n){
76.             int f = 1;
77.             String b = Integer.toBinaryString(pow);
78.             for(int i = 0; i < b.length(); i++){
79.               f = (f*f) % n;
80.               if(b.charAt(i) == '1')
81.                 f = (f*base) % n;
82.             }//for end
83.             return f;
84.     }//method end
85.
86.//This function returns private and public key
87.     public static int[] publicKeyGeneration(){
88.
89.         //Generate a random prime number q
90.         Random rand = new Random();
91.         q = generateRandomPrime(rand);
92.         System.out.println("Generated prime number: q = " + q);
93.
94.         //Assign a value to the primitive root alpha
95.         assignPrimitiveRoot(q);
96.
97.         //Generate a random private key value X which is 1 < X < q-1
98.          Random random = new Random();
99.          int privateKey = random.nextInt(q-3) + 2;
100.               System.out.println("The generated private key is: " +
    privateKey);
101.
102.             //Calculate the public key value Y = alpha^X mod q
103.             int publicKey = fastExp(alpha, privateKey, q);
104.
105.             System.out.println("Generated public key: Y = " +
    publicKey);
```

```
106.
107.             return new int[]{privateKey, publicKey};
108.         }//method end
109.
110.     // Function to generate Message and small value k
111.         public static int generateRandomNumberLessthanQ(int q){
112.             Random rand = new Random();
113.                 if (q <= 2) {
114.             throw new IllegalArgumentException("q must be greater
    than 2");
115.         }//end if
116.         System.out.println("Generating random number with q = " + q);
117.         return rand.nextInt(q-1) + 1;
118.         }//method end
119.
120.         public static int[] encryption(int M, int publicKey, int q,
    int alpha){
121.             //Select a random integer k
122.             int k = generateRandomNumberLessthanQ(q);
123.
124.             //Calculate one time key K
125.             int K  = fastExp(publicKey, k, q);
126.
127.             //Calculate C1 = alpha^k mod q
128.             int C1 = fastExp(alpha, k, q);
129.
130.             //Calculate C2 = M * K mod q
131.             int C2 = (M * K) % q;
132.
133.             return new int[]{C1, C2};
134.         }//method end
135.
136.     //Function to calculate modInverse of a given number
137.     public static int modInverse(int e, int phi){
138.         int t = 0, newT = 1;
139.         int r = phi, newR = e;
140.
141.         while (newR != 0) {
142.                 int quotient = r / newR;
143.
144.                 // Update t and newT
145.                 int tempT = t;
146.                 t = newT;
147.                 newT = tempT - quotient * newT;
148.
149.                 // Update r and newR
150.                 int tempR = r;
151.                 r = newR;
```

```
                newR = tempR - quotient * newR;
            }//while end

            // If r > 1, then e has no modular inverse
            if (r > 1) {
                throw new ArithmeticException("e has no modular
   inverse mod phi(n)");
            }

            // Ensure d is positive
            if (t < 0) {
                t += phi;
            }

            return t;

    }//method end

    //Function to decrypt the message
        public static int decryption(int[] C, int privateKey){

            //Use C1 to get back the one-time key K
            int K = fastExp(C[0], privateKey, q);

            //Calculate modInverse of K
            int inverseK = modInverse(K, q);

            //Calculate M = C2 * K^-1 mod q
            int M = (C[1] * inverseK) % q;

            return M;

        }//method end


    }//class end

```

2. Server.java source file

```
import java.io.*;
import java.net.*;
import ElgamalTools.elgamal;

public class Server {

    public static int q, alpha,Yalice;
    public static void main(String[] args) {
```

```
        int port = 5000; // Define a port number
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server started. Waiting for client...");

            Socket socket = serverSocket.accept(); // Accept client connection
            System.out.println("Client connected!");

            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(),
true);


            String message;
            while ((message = input.readLine()) != null) {
                System.out.println("Client: " + message);

                //Get the public key components of Alice

                // Split using space as a delimiter
                String[] publicKeys = message.trim().split("\\s+");
                if (publicKeys.length == 3) {
                    try{
                        Yalice = Integer.parseInt(publicKeys[0]);
                        alpha = Integer.parseInt(publicKeys[1]);
                        q = Integer.parseInt(publicKeys[2]);

                        System.out.println("Received Public Key Values: Y = "
+ Yalice + ", alpha = " + alpha + ", q = " + q);
                        //output.println("Public key received successfully.");
                    } catch (NumberFormatException e) {
                        System.out.println("Invalid public key format.");
                    }//try catch end
                }//if end

                //Generate a random Integer message
                int M = elgamal.generateRandomNumberLessthanQ(q);
                System.out.println("Generated message(Bob side): " + M);

                //Encrypt the message
                int[] C = elgamal.encryption(M, Yalice, q, alpha);
                System.out.println("Encrypted message: " + C[0] + " " + C[1]);

                //Send the encrypted message to the client
                output.println(C[0] + " " + C[1]);

                if (message.equalsIgnoreCase("bye")) {
                    System.out.println("Client disconnected.");
                    break;
```

```
                }
            }//while end

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. Client.java source file

```
import java.io.*;
import java.net.*;
import ElgamalTools.elgamal;

public class Client {

    public static int q, alpha, Xalice, Yalice;
    public static void main(String[] args) {
        String serverAddress = "localhost"; // Server address
        int port = 5000;

        try (Socket socket = new Socket(serverAddress, port)) {
            System.out.println("Connected to the server.");

            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(),
true);
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));

            //To store the keys
            int[] keys = elgamal.publicKeyGeneration();
            q = elgamal.q;
            alpha = elgamal.alpha;
            Xalice = keys[0]; //Private key of Alice
            Yalice = keys[1]; //Public key of Alice

            String message;
            while (true) {
                System.out.print("Enter message(send the public key value[Y
alpha q] if you are entering the message for the first time): ");
                message = userInput.readLine();
                output.println(message);
```

```
                if (message.equalsIgnoreCase("bye")) {
                    break;
                }//if end

                //Read the encrypted message from the server
                String encryptedMessage = input.readLine();
                if (encryptedMessage == null) {
        System.out.println("Server closed the connection.");
            } else {
                System.out.println("Server: " + encryptedMessage);
            }
                System.out.println("Encrypted message: " + encryptedMessage);

                //Decrypt the message
                String[] parts = encryptedMessage.split(" ");
                int C1 = Integer.parseInt(parts[0]);
                int C2 = Integer.parseInt(parts[1]);

                int[] C = new int[2];
                C[0] = C1;
                C[1] = C2;

                int decryptedMessage = elgamal.decryption(C, Xalice);
                System.out.println("Decrypted message: " + decryptedMessage);
            }//while end

        } catch (IOException e) {
            System.err.println("Connection lost: " + e.getMessage());
        }
    }//Main end
}//class end
```

OUTPUT:

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\sixth_sem\Crypto\RSA> java Client
Connected to the server.
Generated prime number: q = 97
Primitive root alpha: 15
The generated private key is: 11
Generated public key: Y = 29
Enter message(send the public key value[Y alpha q] if you are entering the message for the
first time): 29 15 97
Server: 75 49
Encrypted message: 75 49
Decrypted message: 86
Enter message(send the public key value[Y alpha q] if you are entering the message for the
first time): bye
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\sixth_sem\Crypto\RSA>
```

(Client-side Output – The user who generates the public key components and decrypts the message sent from the server-side user)

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\sixth_sem\Crypto\RSA> java Server
Server started. Waiting for client...
Client connected!
Client: 29 15 97
Received Public Key Values: Y = 29, alpha = 15, q = 97
Generating random number with q = 97
Generated message(Bob side): 86
Generating random number with q = 97
Encrypted message: 75 49
```

(Server-side Output – The user who wants to send the encrypted message using client side user's public key)