CRYPTO LAB-4

NAME: ABISHEK K G **DATE: 06/02/2026**

REGNO: 23BCE1739 **SLOT: L29+L30**

Source code:

```
import java.io.*;

import java.net.*;

import java.util.*;


public class RSASocket {


   // RSA Helper Methods
   static int gcd(int a, int b) {
      while (b != 0) {
         int temp = b;
         b = a % b;
         a = temp;
      }
      return a;
   }


   static int modInv(int e, int phi) {
      int d = 0, x1 = 0, x2 = 1, y1 = 1;
      int tempPhi = phi;


      while (e > 0) {
```

```
        int t1 = tempPhi / e;
        int t2 = tempPhi - t1 * e;
        tempPhi = e;
        e = t2;

        int x = x2 - t1 * x1;
        int y = d - t1 * y1;

        x2 = x1;
        x1 = x;
        d = y1;
        y1 = y;
    }

    if (tempPhi == 1) {
        return d + phi;
    }
    return -1;
}

static long modPow(long base, int exp, int mod) {
    long result = 1;
    base = base % mod;

    while (exp > 0) {
        if (exp % 2 == 1) {
```

```
            result = (result * base) % mod;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// RSA Keys Class
static class Keys {
    int e, d, n;

    Keys() {
        int p = 61;
        int q = 53;

        n = p * q;
        int phi = (p - 1) * (q - 1);

        e = 17;
        while (gcd(e, phi) != 1) {
            e += 2;
        }

        d = modInv(e, phi);
```

```
        System.out.println("Generated Public Key: (e=" + e + ", n=" + n + ")");
        System.out.println("Generated Private Key: (d=" + d + ", n=" + n + ")");
    }
}

// Encryption
static ArrayList<Integer> encrypt(String msg, int e, int n) {
    ArrayList<Integer> cipher = new ArrayList<>();
    for (int i = 0; i < msg.length(); i++) {
        int m = (int) msg.charAt(i);
        int c = (int) modPow(m, e, n);
        cipher.add(c);
    }
    return cipher;
}

// Decryption
static String decrypt(ArrayList<Integer> cipher, int d, int n) {
    String plain = "";
    for (int c : cipher) {
        int m = (int) modPow(c, d, n);
        plain += (char) m;
    }
    return plain;
}
```

```
    // Server Method
    static void runServer(int port) {
        System.out.println("=== RSA Server (Receiver) ===");

        Keys keys = new Keys();

        try {
            ServerSocket ss = new ServerSocket(port);
            System.out.println("\nServer listening on port " + port + "...");
            System.out.println("Waiting for client connection...\n");

            Socket s = ss.accept();
            System.out.println("Connected to client: " + s.getInetAddress());

            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            String data = br.readLine();

            String[] parts = data.split(" ");
            ArrayList<Integer> cipher = new ArrayList<>();
            for (String part : parts) {
                cipher.add(Integer.parseInt(part));
            }

            System.out.println("\nCiphertext received: " + data);
```

```
        String plain = decrypt(cipher, keys.d, keys.n);
        System.out.println("Decrypted Message: " + plain + "\n");

        br.close();
        s.close();
        ss.close();

    } catch (Exception ex) {
        System.out.println("Server Error: " + ex.getMessage());
    }
}

// Client Method
static void runClient(String msg, int port) {
    System.out.println("=== RSA Client (Sender) ===");

    Keys keys = new Keys();

    System.out.println("\nPlaintext message: " + msg);

    ArrayList<Integer> cipher = encrypt(msg, keys.e, keys.n);

    System.out.print("Ciphertext: ");
    for (int c : cipher) {
        System.out.print(c + " ");
    }
```

```
        System.out.println();

        try {
            Socket s = new Socket("localhost", port);

            PrintWriter pw = new PrintWriter(s.getOutputStream(), true);

            String cipherStr = "";
            for (int c : cipher) {
                cipherStr += c + " ";
            }

            pw.println(cipherStr.trim());

            System.out.println("\nMessage sent to server!");

            pw.close();
            s.close();

        } catch (ConnectException ex) {
            System.out.println("\nError: Cannot connect to server. Make sure server is running
first!");
        } catch (Exception ex) {
            System.out.println("Client Error: " + ex.getMessage());
        }
    }
```

```
// Main Method
public static void main(String[] args) {
    if (args.length < 1) {
        System.out.println("Usage:");
        System.out.println("  Server: java RSASocket server");
        System.out.println("  Client: java RSASocket client <message>");
        System.out.println("\nExample:");
        System.out.println("  java RSASocket server");
        System.out.println("  java RSASocket client MESSI");
        return;
    }

    String mode = args[0].toLowerCase();
    int port = 9999;

    if (mode.equals("server")) {
        runServer(port);
    } else if (mode.equals("client")) {
        if (args.length < 2) {
            System.out.println("Error: Please provide a message to send");
            System.out.println("Example: java RSASocket client RONALDO");
            return;
        }
        String msg = args[1];
        runClient(msg, port);
```

```
        } else {
            System.out.println("Invalid mode. Use 'server' or 'client'");
        }
    }
}
```

Detailed explanation:

RSA Socket Implementation - Simple Explanation

What This Program Does:

This program lets two computers talk to each other securely. One computer (client) sends a secret message, and the other computer (server) receives and reads it. The message is encrypted so nobody can read it in the middle.

The Main Parts

1. Helper Methods (The Math Tools)

gcd() - Greatest Common Divisor

This finds the biggest number that divides both numbers evenly

We need this to pick a good encryption key

Like finding what 12 and 8 have in common (answer: 4)

modInv() - Modular Inverse

This is the trickiest part - it finds the decryption key

It uses something called Extended Euclidean Algorithm

Basically, it finds a number that "undoes" the encryption

modPow() - Modular Power

This does big calculations without making huge numbers

It calculates things like (7^13) mod 55 efficiently

This is the actual encryption/decryption math

2. Keys Class (Making the Lock and Key)

What it does:

Creates two special numbers: p=61 and q=53 (both are prime)

Multiplies them: n = 61 × 53 = 3233

Calculates phi = (61-1) × (53-1) = 3120

Picks e=17 (public key - this is the lock)

Calculates d using modInv (private key - this is the key)

Why these numbers?

Small enough to be fast

Big enough to actually work

In real life, we'd use MUCH bigger primes for security

3. encrypt() (Locking the Message)

How it works:

Takes each letter in your message

Converts it to a number (like 'M' = 77)

Does the math: (number^e) mod n

Saves all these encrypted numbers in a list

Example: "HI" becomes two numbers like [2341, 1876]

4. decrypt() (Unlocking the Message)

How it works:

Takes each encrypted number

Does the reverse math: (number^d) mod n

Converts back to letters

Puts all letters together to make the original message

Example: [2341, 1876] becomes "HI" again

5. runServer() (The Receiver)

What it does step by step:

Says "I'm ready to receive messages"

Generates the keys (lock and key)

Opens port 9999 and waits

When client connects, it says "Hey, I see you!"

Receives the encrypted numbers

Uses the private key to decrypt

Shows you the original message

Closes everything

6. runClient() (The Sender)

What it does step by step:

Says "I'm sending a message"

Generates keys (same way as server for demo)

Takes your message like "MESSI"

Encrypts each letter

Connects to the server at port 9999

Sends all the encrypted numbers

Says "Message sent!"

Closes the connection

7. main() (The Control Center)

What it does:

Checks what you typed in the command

If you said "server" → runs the receiver

If you said "client MESSI" → runs the sender with message "MESSI"

If you messed up → shows you how to use it correctly

Important Points

Both generate same keys - In real RSA, only server makes keys and shares the public key. For simplicity, both sides make identical keys here.

Small primes - We use p=61, q=53 for speed. Real RSA uses primes with hundreds of digits!

Port 9999 - Just a random port number. Could be anything above 1024.

Localhost - Means both programs run on same computer. Could be changed to IP address for different computers.

No libraries - Everything is coded from scratch using basic math!

- **Common Issues**
- **"Cannot connect" → Start server first!**
- **Wrong numbers → Keys might be different (shouldn't happen here)**
- **Port busy → Another program is using port 9999**

Output screenshots:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Abishek14> cd "c:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4"
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> javac RSASocket.java
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> java RSASocket client MESSI
=== RSA Client (Sender) ===
Generated Public Key: (e=17, n=3233)
Generated Private Key: (d=2753, n=3233)

Plaintext message: MESSI
Ciphertext: 3123 28 2680 2680 1486

Message sent to server!
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> java RSASocket client RONALDO
=== RSA Client (Sender) ===
Generated Public Key: (e=17, n=3233)
Generated Private Key: (d=2753, n=3233)

Plaintext message: RONALDO
Ciphertext: 1859 1307 3165 2790 2726 1759 1307

Message sent to server!
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4>
```

```
PS C:\Users\Abishek14> cd "c:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4"
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> javac RSASocket.java
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> java RSASocket server
=== RSA Server (Receiver) ===
Generated Public Key: (e=17, n=3233)
Generated Private Key: (d=2753, n=3233)

Server listening on port 9999...
Waiting for client connection...

Connected to client: /127.0.0.1

Ciphertext received: 3123 28 2680 2680 1486
Decrypted Message: MESSI

PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4> java RSASocket server
=== RSA Server (Receiver) ===
Generated Public Key: (e=17, n=3233)
Generated Private Key: (d=2753, n=3233)

Server listening on port 9999...
Waiting for client connection...

Connected to client: /127.0.0.1

Ciphertext received: 1859 1307 3165 2790 2726 1759 1307
Decrypted Message: RONALDO

PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 4>
```