BCSE309L - Cryptography and Network Security
LAB 4
Implementation of RAS Encryption and Decryption

Name:Aryan Sharma
Register Number: 23BCE1320
Language Used: Java

Code:

CryptCore.java
```
import java.math.BigInteger;

public class CryptCore {
    public static BigInteger calculateGCD(BigInteger x, BigInteger y) {
        return x.gcd(y);
    }

    public static BigInteger performEncryption(BigInteger msg,
BigInteger pub, BigInteger mod) {
        return msg.modPow(pub, mod);
    }

    public static BigInteger performDecryption(BigInteger cipher,
BigInteger priv, BigInteger mod) {
        return cipher.modPow(priv, mod);
    }
}
```

RSASender.java
```
import java.io.*;
import java.net.*;
import java.math.BigInteger;
import java.util.Scanner;

public class RSASender {
    public static void main(String[] args) throws Exception {
```

```java
        Scanner console = new Scanner(System.in);
        Socket connection = new Socket("localhost", 9999);

        ObjectInputStream inputSource = new
ObjectInputStream(connection.getInputStream());
        ObjectOutputStream outputTarget = new
ObjectOutputStream(connection.getOutputStream());

        BigInteger keyE = (BigInteger) inputSource.readObject();
        BigInteger keyN = (BigInteger) inputSource.readObject();
        System.out.println("Connected. Modulus: " + keyN);

        System.out.print("Message to send: ");
        String text = console.nextLine();
        byte[] characters = text.getBytes();

        BigInteger[] encryptedData = new BigInteger[characters.length];
        for (int i = 0; i < characters.length; i++) {
            BigInteger m = BigInteger.valueOf(characters[i]);
            encryptedData[i] = CryptCore.performEncryption(m, keyE,
keyN);
        }

        outputTarget.writeObject(encryptedData);
        System.out.println("Transmission complete.");

        connection.close();
    }
}
```

RSAReceiver.java

```java
import java.io.*;
import java.net.*;
import java.math.BigInteger;
import java.util.Scanner;

public class RSAReceiver {
    public static void main(String[] args) throws Exception {
        Scanner input = new Scanner(System.in);
```

```
        System.out.print("Prime P: ");
        BigInteger valP = input.nextBigInteger();
        System.out.print("Prime Q: ");
        BigInteger valQ = input.nextBigInteger();

        BigInteger n = valP.multiply(valQ);
        BigInteger phi =
(valP.subtract(BigInteger.ONE)).multiply(valQ.subtract(BigInteger.ONE))
;

        BigInteger e;
        while (true) {
            System.out.print("Exponent E (coprime to " + phi + "): ");
            e = input.nextBigInteger();
            if (e.gcd(phi).equals(BigInteger.ONE)) break;
            System.out.println("Invalid E, try again.");
        }

        BigInteger d = e.modInverse(phi);

        ServerSocket listener = new ServerSocket(9999);
        System.out.println("Receiver active on port 9999...");
        Socket link = listener.accept();

        ObjectOutputStream outStream = new
ObjectOutputStream(link.getOutputStream());
        ObjectInputStream inStream = new
ObjectInputStream(link.getInputStream());

        outStream.writeObject(e);
        outStream.writeObject(n);

        BigInteger[] dataIncoming = (BigInteger[])
inStream.readObject();

        StringBuilder outputText = new StringBuilder();
        System.out.print("Inbound Cipher: ");
        for (BigInteger block : dataIncoming) {
            System.out.print(block + " ");
            BigInteger raw = CryptCore.performDecryption(block, d, n);
            outputText.append((char) raw.intValue());
        }
```

```
        System.out.println("\nFinal Result: " + outputText.toString());

        link.close();
        listener.close();
    }
}
```

Server Side

```
Prime P: 13
Prime Q: 17
Exponent E (coprime to 192): 83
Receiver active on port 9999...
Inbound Cipher: 143 147 127 11 206 128 60 195 11 147 190 11 128 84 51 196 33 205 43 51 84 211
Final Result: Aryan Sharma 23BCE1320
◆ lab4 ⚡ main ⊗
> pwd

Path
----
C:\Users\aryan\Workspace\EDU\semester-6\cryptography\cryptography-network-security-lab\lab4

◆ lab4 ⚡ main ⊗
```

Client Side

```
> java RSASender
Connected. Modulus: 221
Message to send: Aryan Sharma 23BCE1320
Transmission complete.
◆ lab4 ⚡ main ⊗
> pwd

Path
----
C:\Users\aryan\Workspace\EDU\semester-6\cryptography\cryptography-network-security-lab\lab4

◆ lab4 ⚡ main ⊗
>
```

Conclusion:
This implementation of the RSA protocol successfully bridges the gap between abstract number theory and practical network security. By manually calculating the modular inverse and enforcing the coprimality constraint between the public exponent and Euler's totient, we ensure the mathematical integrity required for valid decryption. The transition from a 128-bit block-based symmetric system (AES) to this character-based asymmetric approach highlights the fundamental difference in key management: here, security is derived from the computational complexity of integer factorization rather than bit-level confusion and diffusion. Through this socket-based exchange, we demonstrate how public keys can be safely distributed over an open connection while keeping the private components isolated on the receiver's end, providing a functional model of modern digital signatures and secure key exchanges.