Project Report:
Implementation of RSA and Attack on RSA

- **Table of Contents_**

- # Introduction_

This project demonstrates the implementation of RSA (Rivest-Shamir-Adleman) encryption in a networked environment using a client-server model. RSA is a widely used public-key cryptosystem that ensures secure data transmission. In this project, two clients communicate securely through a server that acts as an intermediary and simulates a Man-in-the-Middle (MITM) attack. This report provides a comprehensive overview of the project's design, implementation, and results.

- # Objective_

The primary objective of this project is to implement secure communication between two clients using the RSA algorithm. The server facilitates key exchange and message relay while attempting to decrypt intercepted messages, thereby simulating a MITM attack scenario.

- # Overview of RSA Algorithm_

RSA is a public-key cryptosystem that involves three main steps:

1.Key Generation: Generating a pair of keys, a public key and a private key.

2.Encryption: Converting plaintext into ciphertext using the recipient's public key.

3.Decryption: Converting ciphertext back to plaintext using the recipient's private key.

Key Generation

- Choose two distinct large prime numbers p and q.
- Compute $n=p\times q$.
- Compute the totient $\phi(n)=(p-1)\times(q-1)$.
- Choose an integer e such that $1<e<\phi(n)$ and $\gcd(e,\phi(n))=1$.
- Compute the private key d such that $e\times d\equiv 1 \pmod{\phi(n)}$.

Encryption

- Convert the plaintext message to an integer m such that $0\leq m<n$.

- Compute the ciphertext c using the public key: $c \equiv m^e \pmod{n}$.

Decryption

- Compute the plaintext message m using the private key: $m \equiv c^d \pmod{n}$.

System Architecture_

Server (Man-in-the-Middle)

The server performs the following functions:

- Facilitates key exchange between the two clients.
- Relays encrypted messages between the clients.
- Attempts to decrypt the intercepted messages using factored keys.

Client 1

Client 1 performs the following functions:

- Generates its own RSA key pair.
- Requests and receives the public key of Client 2 from the server.
- Encrypts messages using Client 2's public key.
- Sends encrypted messages to the server.

Client 2

Client 2 performs the following functions:

- Generates its own RSA key pair.
- Requests and receives the public key of Client 1 from the server.
- Encrypts messages using Client 1's public key.
- Sends encrypted messages to the server.

Implementation Details_

Key Generation

Each client generates an RSA key pair using pre-defined prime numbers and computes the modulus n, totient $\phi(n)$, public exponent e, and private exponent d.

Encryption

Each client converts the plaintext message into its ASCII representation, encrypts each character using the recipient's public key, and sends the encrypted message to the server.

Decryption

Each client receives the encrypted message from the server, decrypts each character using its private key, and reconstructs the plaintext message.

- ## Communication Protocol_

Message Flow

1. Client 1 sends its public key to the server.
2. Client 2 sends its public key to the server.
3. The server exchanges the public keys between the clients.
4. Client 1 encrypts a message using Client 2's public key and sends it to the server.
5. The server relays the encrypted message to Client 2.
6. Client 2 decrypts the message using its private key and reads the plaintext.
7. Client 2 encrypts a response using Client 1's public key and sends it to the server.
8. The server relays the encrypted response to Client 1.
9. Client 1 decrypts the response using its private key and reads the plaintext.

Security Measures

- Public keys are transmitted securely between clients via the server.
- Encrypted messages ensure that the content cannot be deciphered without the private key.

- ## Detailed Code Explanation _

Client 1 Code:

Overview: ChatClient1 is responsible for generating its RSA key pair, exchanging keys with ChatClient2 via the server, encrypting messages using Client 2's public key, and decrypting messages received from Client 2 using its private key.

Key Points:

1. **Connection Establishment:**
    - The client connects to the server at a specified IP address and port.
2. **RSA Key Generation:**
    - Prime numbers **p1** and **q1** are chosen to compute **n1**, the modulus.
    - The totient **phi_n1** is calculated.
    - Public exponent **e1** is selected.
    - Private exponent **d1** is derived using the **calculatePrivateKey** method.
3. **Key Exchange:**
    - The client's public key **(n1, e1)** is sent to the server.
    - The public key of Client 2 is received from the server.
4. **Message Encryption and Decryption:**
    - Messages are encrypted using Client 2's public key before sending.
    - Received messages are decrypted using the client's private key.
5. **Communication Loop:**
    - The client reads user input, encrypts it, and sends it to the server.
    - It reads the encrypted response from the server, decrypts it, and displays the plaintext.
    -

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

class ChatClient1 {
    public static void main(String[] args) throws Exception {
        // Establish a connection to the server
        Socket socket = new Socket("192.168.115.202", 1234);
        System.out.println("Connected to server");

        Scanner scanner = new Scanner(System.in);

        // RSA key generation
        int p1 = 61;
        int q1 = 53;
```

```
        int e1 = 17;
        int phi_n1 = (p1 - 1) * (q1 - 1);
        int n1 = p1 * q1;
        int d1 = calculatePrivateKey(e1, phi_n1);

        System.out.println("Calculated public key locally: (" + e1 + "," + n1 +
")");
        System.out.println(d1 + " this is private key \n");

        // Send public key to the server
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        out.println(n1);
        out.println(e1);

        // Receive public key of Client 2 from the server
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        int n2 = Integer.parseInt(in.readLine());
        int e2 = Integer.parseInt(in.readLine());

        System.out.println("Received public key : (" + e2 + "," + n2 + ")");

        BigInteger keypu = BigInteger.valueOf(e2);
        BigInteger npu = BigInteger.valueOf(n2);
        BigInteger keypr = BigInteger.valueOf(d1);
        BigInteger npr = BigInteger.valueOf(n1);

        // Receive start chat message from server
        System.out.println("SERVER: " + in.readLine());

        // Start chat
        while (true) {
            System.out.print("ME: ");
            String kbstr = scanner.nextLine();
            String encryptedMessage = encryptMessage(kbstr, keypu, npu);
            out.println(encryptedMessage);

            String receivedMessage = in.readLine();
            String decryptedMessage = decryptMessage(receivedMessage, keypr, npr);
            System.out.println("Client 2: " + decryptedMessage);
        }
    }

    private static int calculatePrivateKey(int e, int phi) {
        int d;
        for (d = 1; d > 0; d++) {
            if ((e * d) % phi == 1) {
                return d;
            }
        }
        return -1; // This line should never be reached
    }
```

```
    private static String encryptMessage(String message, BigInteger key, BigInteger
n) {
        StringBuilder asciiString = new StringBuilder();
        for (char ch : message.toCharArray()) {
            BigInteger ch1 = BigInteger.valueOf((int) ch);
            BigInteger encryptedCh = ch1.modPow(key, n);
            asciiString.append(encryptedCh).append(' ');
        }
        if (asciiString.length() > 0) {
            asciiString.setLength(asciiString.length() - 1);
        }
        return asciiString.toString();
    }

    private static String decryptMessage(String message, BigInteger key, BigInteger
n) {
        StringBuilder decryptedMessage = new StringBuilder();
        String[] cipherText = message.split(" ");
        for (String cipher : cipherText) {
            BigInteger encryptedCh = new BigInteger(cipher);
            BigInteger decryptedCh = encryptedCh.modPow(key, n);
            decryptedMessage.append((char) decryptedCh.intValue());
        }
        return decryptedMessage.toString();
    }
}
```

Client 2 Code:

Overview: ChatClient2 is similar to ChatClient1 in functionality. It generates its RSA key pair, exchanges keys with ChatClient1 via the server, encrypts messages using Client 1's public key, and decrypts messages using its private key.

Key Points:

1. **Connection Establishment:**
    - The client connects to the server at a specified IP address and port.
2. **RSA Key Generation:**
    - Prime numbers **p2** and **q2** are chosen to compute **n2**, the modulus.
    - The totient **phi_n2** is calculated.
    - Public exponent **e2** is selected.

- Private exponent **d2** is derived using the **calculatePrivateKey** method.

3. **Key Exchange:**

- The client's public key **(n2, e2)** is sent to the server.
- The public key of Client 1 is received from the server.

4. **Message Encryption and Decryption:**

- Messages are encrypted using Client 1's public key before sending.
- Received messages are decrypted using the client's private key.

5. **Communication Loop:**

- The client reads user input, encrypts it, and sends it to the server.
- It reads the encrypted response from the server, decrypts it, and displays the plaintext.

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

class ChatClient2{
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("192.168.205.202", 1234);
        System.out.println("Connected to server");


        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String cStr = in.readLine();
        int n1 = Integer.parseInt(cStr);


        String cStr2 = in.readLine();
        int e1 = Integer.parseInt(cStr2);


        System.out.println("Recieved public key : (" + e1);
        System.out.print("," + n1 +")\n");


        Scanner scanner = new Scanner(System.in);

        int p2 = 13;
        int q2 = 17;
```

```
        int e2 = 7;

        int phi_n2 = (p2 - 1) * (q2 - 1);
        int n2 = p2 * q2;
        System.out.println("Calculated public key locally: (" + e2);
        System.out.println("," + n2 +")\n");

        int d2;
       for (d2 = 1; d2 > 0; d2++)
       {
           if (((e2 * d2) % phi_n2) == 1)
           {
            System.out.println(d2 + " this is private key\n ");
               break;
           }
       }

        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        out.println(n2);
        out.println(e2);

        BigInteger keypu = BigInteger.valueOf(e1);
        BigInteger npu = BigInteger.valueOf(n1);
        BigInteger keypr = BigInteger.valueOf(d2);
        BigInteger npr = BigInteger.valueOf(n2);

        // Receive start chat message from server
        String message = in.readLine();
        System.out.println("SERVER: " + message);


        // Start chat
        while (true) {


            System.out.print("ME: ");
            String kbstr = scanner.nextLine();

            //sending encrypted msg
            StringBuilder asciiString = new StringBuilder();
            for (char ch : kbstr.toCharArray()) {
                int cv = (int) ch ;

                BigInteger ch1 = BigInteger.valueOf(cv);

                BigInteger encryptedCh = ch1.modPow(keypu, npu);

            asciiString.append(encryptedCh).append(' ');
        }
        if (asciiString.length() > 0) {
            asciiString.setLength(asciiString.length() - 1);
        }
```

```
        out.println(asciiString.toString());


            String msg = in.readLine();

             String[] cipherText = msg.split(" ");
                    System.out.print("Client 1: ");
                    for (String cipher : cipherText) {
                        BigInteger encryptedCh = new BigInteger(cipher);
                        BigInteger decryptedCh = encryptedCh.modPow(keypr, npr);
                        int charValue = decryptedCh.intValue();
                        System.out.print((char) charValue);  // ASCII conversion
                    }
                    System.out.println();


        }
    }
}
```

Attacker Code:

Overview: The server acts as an intermediary between the two clients. It facilitates the key exchange and relays messages. It also demonstrates the ability to decrypt intercepted messages using calculated private keys.

Key Points:

1. **Connection Management:**
    - The server listens on a specified port and accepts connections from two clients.
2. **Key Exchange Facilitation:**
    - The server receives the public keys from both clients and forwards them to each other.
3. **MITM Decryption:**
    - The server calculates private keys for both clients using the **calculatePrivateKey** method.
    - Intercepts and decrypts messages exchanged between the clients.
4. **Message Relaying:**

- The server relays encrypted messages between the clients.
- It prints both the intercepted encrypted messages and the decrypted plaintext.

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
import java.security.*;
import javax.crypto.Cipher;
import java.util.Base64;
import java.math.BigInteger;


class ManInMiddle{

    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(1234);
        System.out.println("Server started. Waiting for clients...");

        // Accept two clients
        Socket client1Socket = serverSocket.accept();
        System.out.println("Client 1 connected");
        Socket client2Socket = serverSocket.accept();
        System.out.println("Client 2 connected");

        // Create input and output streams for both clients
        PrintWriter out1 = new PrintWriter(client1Socket.getOutputStream(), true);
        BufferedReader in1 = new BufferedReader(new
InputStreamReader(client1Socket.getInputStream()));
        PrintWriter out2 = new PrintWriter(client2Socket.getOutputStream(), true);
        BufferedReader in2 = new BufferedReader(new
InputStreamReader(client2Socket.getInputStream()));

        // Receive key from client 1
        String cStr = in1.readLine();
        int n1 = Integer.parseInt(cStr);
        System.out.println("Received n1 from client 1: " + n1);

        String cStr2 = in1.readLine();
        int e1 = Integer.parseInt(cStr2);
        System.out.println("Received e1 from client 1: " + e1);

        // Send e/n to client 2
        out2.println(n1);
        out2.println(e1);

        double ex, nx, a1, b1;
        int d1 = 0;
```

```
        nx = n1;
        ex = e1;

        double temp1;
        temp1 = Math.sqrt(nx);

        double p1 = 0, q1 = 0;

        boolean found1 = false;
        for (a1 = Math.floor(temp1); a1 < nx; a1++) {
            b1 = Math.sqrt((a1 * a1) - nx);
            b1 = Math.floor(b1);
            if (b1 * b1 == ((a1 * a1) - nx)) {
                p1 = a1 + b1;
                q1 = a1 - b1;

                double phi_n1;
                phi_n1 = (p1 - 1) * (q1 - 1);

                for (d1 = 1; ; d1++) {
                    if (((ex * d1) % phi_n1) == 1) {
                        System.out.println(d1 + " this is private key client 1 ");
                        break;
                    }
                }


                found1 = true;
                break;
            }
        }

        if (!found1) {
            System.out.println("Failed to find factors p and q.");
        }


        // Receive key from client 2
        String eStr = in2.readLine();
        int n2 = Integer.parseInt(eStr);
        System.out.println("Received n2 from client 2: " + n2);

        String eStr2 = in2.readLine();
        int e2 = Integer.parseInt(eStr2);
        System.out.println("Received e2 from client 1: " + e2);

         // Send e/n to client 1
         out1.println(n2);
         out1.println(e2);

         double ey, ny, a2, b2;
         int d2 = 0;
```

```
        ny = n2;
        ey = e2;

        double temp2;
        temp2 = Math.sqrt(ny);

        double p2 = 0, q2 = 0;

        boolean found2 = false;
        for (a2 = Math.floor(temp2); a2 < ny; a2++) {
            b2 = Math.sqrt((a2 * a2) - ny);
            b2 = Math.floor(b2);
            if (b2 * b2 == ((a2* a2) - ny)) {
                p2 = a2 + b2;
                q2 = a2 - b2;

                double phi_n2;
                phi_n2 = (p2 - 1) * (q2 - 1);

                for (d2 = 1; ; d2++) {
                    if (((ey * d2) % phi_n2) == 1) {
                        System.out.println(d2 + " this is private key client 2 ");
                        break;
                    }
                }

                found2 = true;
                break;
            }
        }
        System.out.println(d2 + " this is private key client 2 ");
        if (!found2) {
            System.out.println("Failed to find factors p and q.");
        }


        BigInteger keypr1 = BigInteger.valueOf(d1);
        BigInteger npr1 = BigInteger.valueOf(n1);
        BigInteger keypr2 = BigInteger.valueOf(d2);
        BigInteger npr2 = BigInteger.valueOf(n2);

    // Start chat between clients
    out1.println("Start chatting!");
    out2.println("Start chatting!");

    // Forward messages between clients
    while (true) {
        String message1 = in1.readLine();
        out2.println(message1);
        System.out.print("Client 1 encrypted: "+ message1 +"\n");
        String[] cipherText1 = message1.split(" ");
```

```
            System.out.print("Client 1 decrypted: ");
            for (String cipher1 : cipherText1) {
                BigInteger encryptedCh1 = new BigInteger(cipher1);
                BigInteger decryptedCh1 = encryptedCh1.modPow(keypr2, npr2);
                int charValue1 = decryptedCh1.intValue();
                System.out.print((char) charValue1);  // ASCII conversion
            }
            System.out.println();



            String message2 = in2.readLine();
            out1.println(message2);
            System.out.print("Client 2 encrypted: "+ message2 +"\n");
            String[] cipherText2 = message2.split(" ");
            System.out.print("Client 2 decrypted: ");
            for (String cipher2 : cipherText2) {
                BigInteger encryptedCh2 = new BigInteger(cipher2);
                BigInteger decryptedCh2 = encryptedCh2.modPow(keypr1, npr1);
                int charValue2 = decryptedCh2.intValue();
                System.out.print((char) charValue2);  // ASCII conversion
            }
            System.out.println();
        }
    }
}
```

- **Testing and Results_**

The system was tested by running the server and both clients. Messages were successfully encrypted, transmitted, and decrypted by the intended recipients. The server was able to intercept and decrypt the messages, demonstrating the MITM attack scenario.

Test Cases

Test Case 1: Client 1 sends a message to Client 2.

Expected Result: Client 2 receives and decrypts the message correctly.

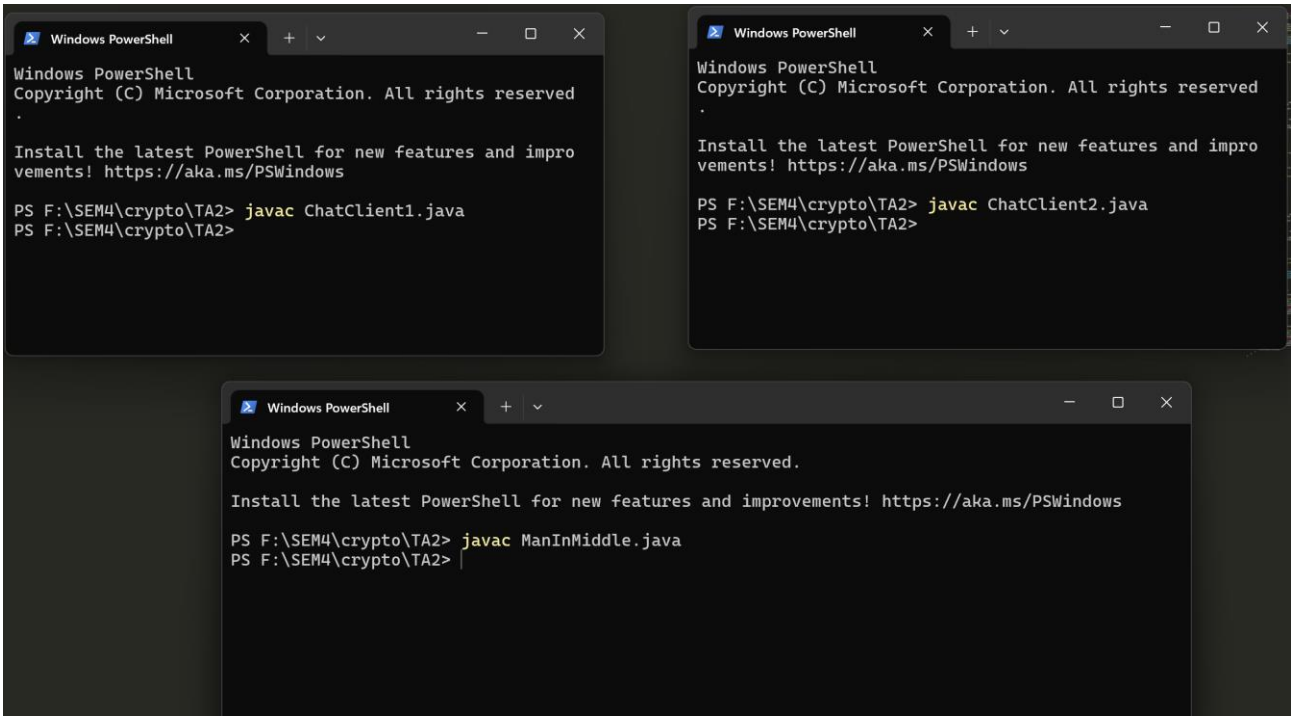
Test Case 2: Client 2 sends a message to Client 1.

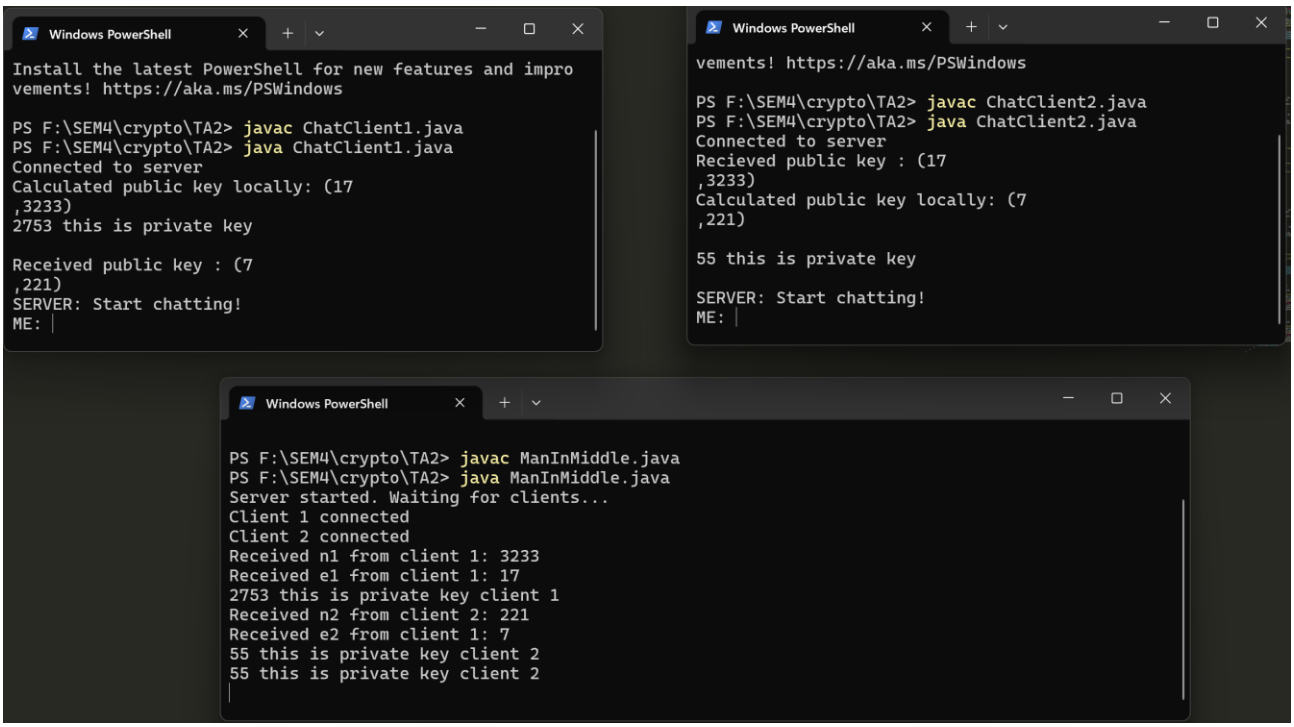
Expected Result: Client 1 receives and decrypts the message correctly.

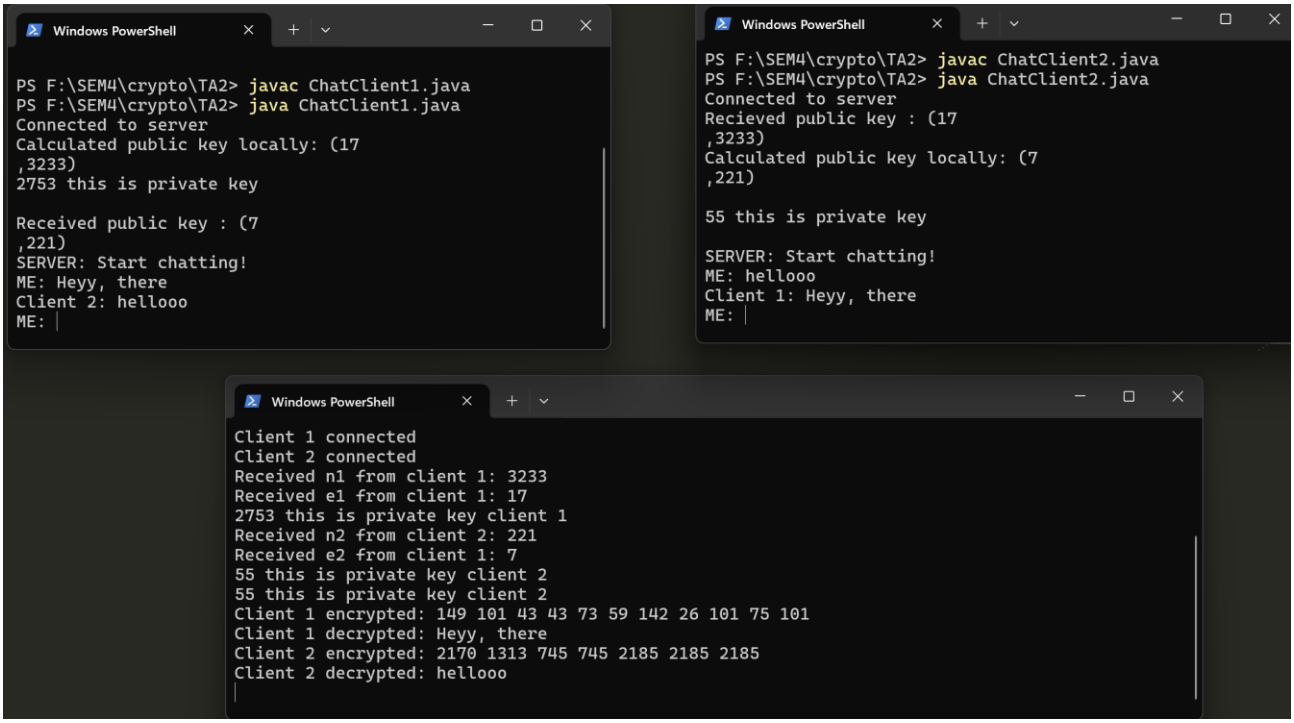
Test Case 3: Server intercepts and decrypts messages.

Expected Result: Server decrypts and displays the intercepted messages correctly.

ScreenShots of Output:







- **Challenges and Solutions_**

Challenges

- Ensuring accurate key generation and encryption/decryption processes.
- Managing simultaneous communication between multiple clients and the server.
- Simulating a realistic MITM attack.

Solutions

- Implemented rigorous testing and debugging to ensure accurate RSA operations.
- Used multi-threading to handle multiple clients simultaneously.
- Calculated private keys on the server to simulate decryption of intercepted messages

- **Future Work_**

Implementing stronger security measures to prevent MITM attacks.

Adding authentication mechanisms to verify the identity of clients.

Extending the system to support more clients and different types of encryption algorithms.

• Conclusion_

This project successfully demonstrates secure communication between two clients using the RSA algorithm with a server acting as a MITM. The implementation showcases the importance of secure key exchange and highlights potential vulnerabilities in cryptographic systems.

• References_

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

Stallings, W. (2016). Cryptography and Network Security: Principles and Practice (7th Edition). Pearson.

Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd Edition). John Wiley & Sons.

This report covers the detailed implementation and testing of the RSA algorithm in a networked environment, including a simulation of a MITM attack