**CRYPTOGRAPHY AND NETWORK SECURITY LAB EXERCISE – 5**

**Man in the middle attack**

**21BCE1672 Adittya Narayan**

**Server.java:**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class server {

    public static void main(String[] args) {

        try {

            // Wait for connections from ClientBob and ClientEve

            ServerSocket serverSocket = new ServerSocket(1900);

            System.out.println("ServerAlice is running. Waiting for connections...");

            // Connect with ClientBob

            Socket bobSocket = serverSocket.accept();

            System.out.println("Connected to Bob.");

            DataInputStream bobDis = new DataInputStream(bobSocket.getInputStream());

            DataOutputStream bobDos = new DataOutputStream(bobSocket.getOutputStream());

            // Connect with ClientEve

            Socket eveSocket = serverSocket.accept();

            System.out.println("Connected to Eve.");

            DataInputStream eveDis = new DataInputStream(eveSocket.getInputStream());

            DataOutputStream eveDos = new DataOutputStream(eveSocket.getOutputStream());

            int p = 227;

            int g = 14;

            // Receive p and g from Bob

            int bobP = bobDis.readInt();

            int bobG = bobDis.readInt();

            System.out.println("Received p from Bob: " + bobP);

            System.out.println("Received g from Bob: " + bobG);

            // Receive p and g from Eve

            int eveP = eveDis.readInt();

            int eveG = eveDis.readInt();

            System.out.println("Received p from Eve: " + eveP);

            System.out.println("Received g from Eve: " + eveG);

            if (bobP == p && bobG == g && eveP == p && eveG == g) {

                // Send p and g to Bob

                bobDos.writeInt(p);

                bobDos.writeInt(g);

                // Send p and g to Eve

                eveDos.writeInt(p);

                eveDos.writeInt(g);

                // Receive b and d from Bob and Eve

                int b = bobDis.readInt();

                int d = eveDis.readInt();

                System.out.println("Received b from Bob: " + b);

                System.out.println("Received d from Eve: " + d);

                while (true) {

                    // Receive ga from Bob

                    int ga = bobDis.readInt();

                    System.out.println("Received ga from Bob: " + ga);

                    // Send ga to Eve

                    eveDos.writeInt(ga);

                    // Receive gb from Eve

                    int gb = eveDis.readInt();

                    System.out.println("Received gb from Eve: " + gb);

                    // Send gb to Bob

                    bobDos.writeInt(gb);

                    // Compute secret keys

                    int secretKeyA = computeSecretKey(gb, b, p);

                    int secretKeyB = computeSecretKey(ga, d, p);

                    System.out.println("Computed secret key for Bob (S2): " + secretKeyA);

                    System.out.println("Computed secret key for Alice (S1): " + secretKeyB);

                    // Send secret keys to Bob and Eve

                    bobDos.writeInt(secretKeyB);

                    eveDos.writeInt(secretKeyA);

                }

            } else {

                // If p or g doesn't match, close the connections

                System.out.println("Invalid p or g value. Closing connections.");

                bobSocket.close();

                eveSocket.close();

                serverSocket.close();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static int computeSecretKey(int g, int x, int p) {

        return (int) Math.pow(g, x) % p;

    }

}

**ClientA.java:**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import java.util.Scanner;

public class clientA {

    public static void main(String[] args) {

        try {

            // Connect to ServerAlice

            Socket aliceSocket = new Socket("localhost", 1900);

            System.out.println("Connected to ServerAlice.");

            DataInputStream aliceDis = new DataInputStream(aliceSocket.getInputStream());

            DataOutputStream aliceDos = new

            DataOutputStream(aliceSocket.getOutputStream());

            // Connect to ClientEve

            Socket eveSocket = new Socket("localhost", 2020);

            System.out.println("Connected to ClientEve.");

            DataInputStream eveDis = new DataInputStream(eveSocket.getInputStream());

            DataOutputStream eveDos = new DataOutputStream(eveSocket.getOutputStream());

            int p = 227;

            int g = 14;

            // Send p and g to ServerAlice

            aliceDos.writeInt(p);

            aliceDos.writeInt(g);

            // Send p and g to ClientEve

            eveDos.writeInt(p);

            eveDos.writeInt(g);

            // Receive p and g from ServerAlice

            int aliceP = aliceDis.readInt();

            int aliceG = aliceDis.readInt();

            System.out.println("Received p from ServerAlice: " + aliceP);

            System.out.println("Received g from ServerAlice: " + aliceG);

            // Receive p and g from ClientEve

            int eveP = eveDis.readInt();

            int eveG = eveDis.readInt();

            System.out.println("Received p from ClientEve: " + eveP);

            System.out.println("Received g from ClientEve: " + eveG);

            if (aliceP == p && aliceG == g && eveP == p && eveG == g) {

                System.out.print("Enter the private key (b): ");

                Scanner scanner = new Scanner(System.in);

                int b = scanner.nextInt();

                // Send b to ServerAlice

                aliceDos.writeInt(b);

                // Send b to ClientEve

                eveDos.writeInt(b);

                while (true) {

                    // Receive ga from ServerAlice

                    int ga = aliceDis.readInt();

                    System.out.println("Received ga from ServerAlice: " + ga);

                    // Send ga to ClientEve

                    eveDos.writeInt(ga);

                    // Receive gb from ClientEve

                    int gb = eveDis.readInt();

                    System.out.println("Received gb from ClientEve: " + gb);

                    // Send gb to ServerAlice

                    aliceDos.writeInt(gb);

                    // Compute secret keys

                    int secretKeyB = computeSecretKey(ga, b, p);

                    int secretKeyA = computeSecretKey(gb, b, p);

                    System.out.println("Computed secret key for Alice (S1): " + secretKeyA);

                    System.out.println("Computed secret key for Bob (S2): " + secretKeyB);

                    // Send secret keys to ServerAlice and ClientEve

                    aliceDos.writeInt(secretKeyB);

                    eveDos.writeInt(secretKeyA);

                }

            } else {

                // If p or g doesn't match, close the connections

                System.out.println("Invalid p or g value. Closing connections.");

                aliceSocket.close();

                eveSocket.close();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static int computeSecretKey(int g, int x, int p) {

        return (int) Math.pow(g, x) % p;

    }

}

**Mtm.java:**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Scanner;

public class mtm {

    public static void main(String[] args) {

        try {

            // connect to server alice

            Socket aliceSocket = new Socket("localhost", 1900);

            System.out.println("Connected to server alice");

            DataInputStream aliceDis = new DataInputStream(aliceSocket.getInputStream());

            DataOutputStream aliceDos = new

            DataOutputStream(aliceSocket.getOutputStream());

            ServerSocket serverSocket = new ServerSocket(2020);

            System.out.println("Eve is running, waiting for connections");

            // Connect with ClientBob

            Socket bobSocket = serverSocket.accept();

            System.out.println("Connected to Bob.");

            DataInputStream bobDis = new DataInputStream(bobSocket.getInputStream());

            DataOutputStream bobDos = new DataOutputStream(bobSocket.getOutputStream());

            int p = 227;

            int g = 14;

            // send p and g to Bob

            bobDos.writeInt(p);

            bobDos.writeInt(g);

            //send to alice

            aliceDos.writeInt(p);

            aliceDos.writeInt(g);

            int bobP = bobDis.readInt();

            int bobG = bobDis.readInt();

            System.out.println("Received p from Bob: " + bobP);

            System.out.println("Received g from Bob: " + bobG);

            // Receive p and g from Eve

            int aliceP = aliceDis.readInt();

            int aliceG = aliceDis.readInt();

            System.out.println("Received p from Alice: " + aliceP);

            System.out.println("Received g from Alice: " + aliceG);

            if (bobP == p && bobG == g && aliceP == p && aliceG == g) {

                // Send p and g to Bob

                int d = (int)(Math.random() \* 100);

                bobDos.writeInt(d);

                // Send d to ServerAlice

                aliceDos.writeInt(d);

                while (true) {

                    // Receive ga from ClientBob

                    int ga = bobDis.readInt();

                    System.out.println("Received ga from ClientBob: " + ga);

                    // Send ga to ServerAlice

                    aliceDos.writeInt(ga);

                    // Receive gb from ServerAlice

                    int gb = aliceDis.readInt();

                    System.out.println("Received gb from ServerAlice: " + gb);

                    // Send gb to ClientBob

                    bobDos.writeInt(gb);

                    // Compute secret keys

                    int secretKeyD = computeSecretKey(gb, d, p);

                    int secretKeyA = computeSecretKey(ga, d, p);

                    System.out.println("Computed secret key for ClientBob (S2): " +

                        secretKeyD);

                    System.out.println("Computed secret key for Eve (S1): " + secretKeyA);

                    // Send secret keys to ClientBob and ServerAlice

                    bobDos.writeInt(secretKeyA);

                    aliceDos.writeInt(secretKeyD);

                }

            } else {

                // If p or g doesn't match, close the connections

                System.out.println("Invalid p or g value. Closing connections.");

                aliceSocket.close();

                serverSocket.close();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static int computeSecretKey(int g, int x, int p) {

        return (int) Math.pow(g, x) % p;

    }

}

**Output:**





