|  |  |
| --- | --- |
| Ex.No.7  29.08.24 | **IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE** |

|  |
| --- |
| **AIM:** |

To simulate the working of Diffie Hellman in Virtual lab environment and to implement the same in Client/ Server model using Java/Python

|  |
| --- |
| **THEORY:** |

**Steps Involved:**

1. **SELECTION OF PRIME NUMBERS:**

Two communicating parties agree on:

* A large prime number q.
* A primitive root alpha of q.

**2. PRIVATE KEYS:**

Each party selects a private key:

* One party chooses a private key a.
* The other party chooses a private key b.

**3.PUBILC KEYS:**

Both parties compute public keys:

* The first party computes their public key as A=g a mod p and sends it.
* The second party computes their public key as B=g b mod p and sends it.
* These public keys are shared between the two parties.

**4. Exchange and Computation of Shared Secret:**

After exchanging public keys, each party calculates the shared secret:

* The first party computes S=B^a mod p.
* The second party S = A ^b mod p.

|  |
| --- |
| **ALGORITHM:** |

**1.KEY EXCHANGE:**

* **Server:**
* Initializes a socket to listen for a connection.
* Sends the public values p and g.
* Generates a private key a and computes the public key A=g^ a mod p.
* Receives the client’s public key B.
* Sends its public key A to the client.
* Computes the shared secret S = B^a mod p.
* **Client:**
* Connects to the server.
* Receives the public values p and g.
* Generates a private key b and computes the public key B=g^ b mod p.
* Sends its public key B to the server.
* Receives the server’s public key A.
* Computes the shared secret S= A^ b mod p.

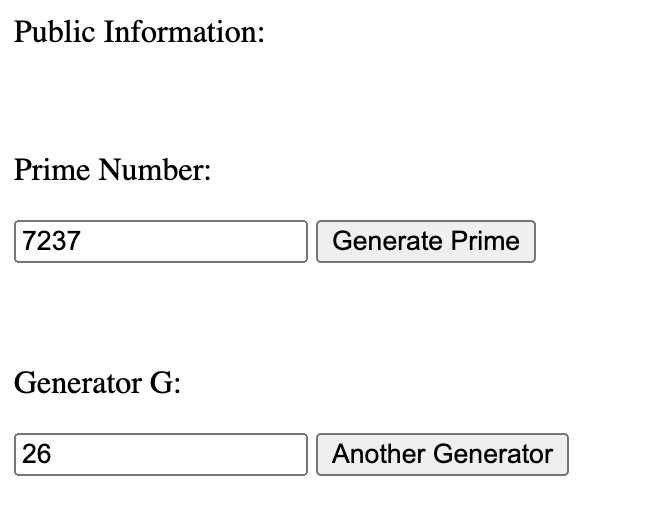
**2.MAN IN THE MIDDLE ATTACK:**

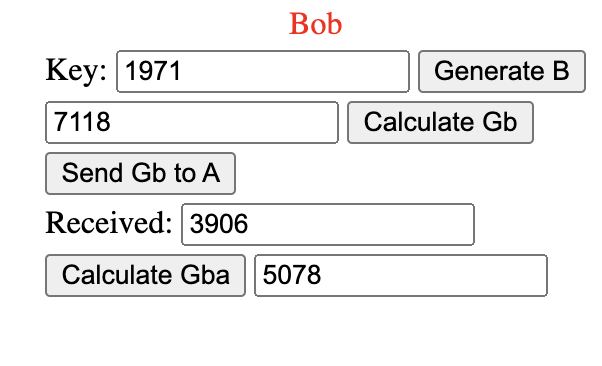
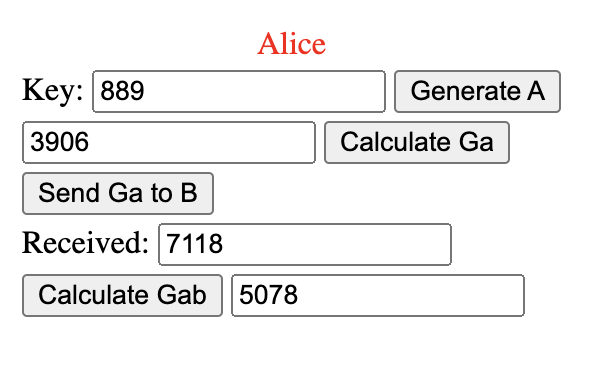
* **Server (Believing it's communicating directly with Client):**
* Initializes a socket to listen for a connection.
* Sends the public values p and g to the **attacker** (thinking it’s the client).
* Generates a private key a and computes the public key A = g^a mod p.
* Receives the attacker's public key C (thinking it’s from the client).
* Sends its public key A to the attacker.
* Computes the shared secret S = C^a mod p (thinking it's a secure key shared with the client).
* **Client (Believing it's communicating directly with Server):**
* Connects to the **attacker** (thinking it’s the server).
* Receives the public values p and g from the attacker (thinking it's from the server).
* Generates a private key b and computes the public key B = g^b mod p.
* Sends its public key B to the attacker.
* Receives the attacker's public key C (thinking it’s from the server).
* Computes the shared secret S = C^b mod p (thinking it's a secure key shared with the server).
* **Attacker:**
* **Intercepts** the initial connection between the server and the client.
* When the server sends p and g, the attacker **forwards** these values to the client.
* When the client sends its public key B, the attacker intercepts it and **does not forward** it directly to the server.
* Instead, the attacker generates its own private key c and computes its public key C = g^c mod p.
* Sends C to the server as if it's from the client and sends C to the client as if it's from the server.
* The attacker now computes:

**1.** Shared secret with the server: S\_server = A^c mod p.

**2.** Shared secret with the client: S\_client = B^c mod p.

|  |
| --- |
| **Screen Shots of simulation in Virtual labs** |





|  |
| --- |
| **Coding** |

**1.KEY EXCHANGE:**

**UserA.java (SERVER):**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.Random;

public class UserA {

public static void main(String[] args) throws IOException {

ServerSocket ss = new ServerSocket(2729);

System.out.println("Server (User A) is Running... Waiting For UserB to connect.");

Socket s = ss.accept();

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

BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

PrintWriter out = new PrintWriter(s.getOutputStream(), true); // true for auto-flushing

BigInteger q = new BigInteger("162259276829213363391578010288127");

BigInteger alpha = new BigInteger("5");

out.println(q);

out.println(alpha);

System.out.println("q:"+q);

System.out.println("Alpha:"+alpha);

BigInteger Xa;

Random rand = new Random();

do {

Xa = new BigInteger(q.bitLength(), rand);

} while (Xa.compareTo(q) >= 0 || Xa.equals(BigInteger.ZERO));

System.out.println("UserA's private Key: " + Xa);

BigInteger Ya = alpha.modPow(Xa, q);

System.out.println("Computed UserA's Public Key: " + Ya);

out.println(Ya);

BigInteger Yb = new BigInteger(in.readLine());

BigInteger K = Yb.modPow(Xa, q);

System.out.println("UserA's Shared Secret Key: " + K);

in.close();

out.close();

s.close();

ss.close();

}

}

**UserB.java (CLIENT):**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.Socket;

import java.util.Random;

public class UserB {

public static void main(String[] args) throws IOException {

Socket socket = new Socket("localhost", 2729);

System.out.println("Connected To Server (UserA)...");

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true); // true for auto-flushing

BigInteger q = new BigInteger(in.readLine());

BigInteger alpha = new BigInteger(in.readLine());

BigInteger Ya = new BigInteger(in.readLine());

System.out.println("Received q:"+q);

System.out.println("Received alpha:"+alpha);

System.out.println("Received UserA's Public Key:"+Ya);

Random rand = new Random();

BigInteger Xb;

do {

Xb = new BigInteger(q.bitLength(), rand);

} while (Xb.compareTo(q) >= 0 || Xb.equals(BigInteger.ZERO));

System.out.println("UserB's Private Key: " + Xb);

BigInteger Yb = alpha.modPow(Xb, q);

System.out.println("Computed UserB's Public Key: " + Yb);

out.println(Yb);

BigInteger K = Ya.modPow(Xb, q);

System.out.println("UserB's Shared Secret Key: " + K);

in.close();

out.close();

socket.close();

}

}

**2.MAN IN THE MIDDLE ATTACK:**

**UA.java (UserA):**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.Socket;

import java.security.SecureRandom;

public class UA {

public static void main(String[] args) throws IOException {

Socket socket=new Socket("localhost",2729);

System.out.println("Connected To Attacker (Believing Its User B)...");

BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));

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

BigInteger q = new BigInteger("162259276829213363391578010288127");

BigInteger alpha = new BigInteger("5");

System.out.println("q:"+q);

System.out.println("Alpha:"+alpha);

out.println(q);

out.println(alpha);

System.out.println("q and alpha are sent to the Attacker (Believing Its User B)...");

BigInteger Xa;

SecureRandom rand=new SecureRandom();

do{

Xa=new BigInteger(q.bitLength(),rand);

}while(Xa.compareTo(q)>=0||Xa.equals(BigInteger.ZERO));

System.out.println("Generated UserA's Private Key:"+Xa);

BigInteger Ya=alpha.modPow(Xa,q);

System.out.println("Computed UserA's Public Key:"+Ya);

out.println(Ya);

System.out.println("Public Key is Sent To Attcker (Believing Its User B)...");

BigInteger Yb=new BigInteger(in.readLine());

System.out.println("Received Attacker's Public Key (Believing Its User B):"+Yb);

BigInteger Ka=Yb.modPow(Xa,q);

System.out.println("Shared Secret Key Of User A:"+Ka);

socket.close();

in.close();

out.close();

}

}

**UB.java (UserB):**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.Socket;A

import java.security.SecureRandom;

public class UB {

public static void main(String[] args) throws IOException {

Socket socket=new Socket("localhost",2728);

System.out.println("Connected To Attacker (Believing Its User A)...");

BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));

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

BigInteger q=new BigInteger(in.readLine());

BigInteger alpha=new BigInteger(in.readLine());

System.out.println("Received q:"+q);

System.out.println("Received Alpha:"+alpha);

BigInteger Xb;

SecureRandom rand=new SecureRandom();

do{

Xb=new BigInteger(q.bitLength(),rand);

}while(Xb.compareTo(q)>=0||Xb.equals(BigInteger.ZERO));

System.out.println("Generated UserB's Private Key:"+Xb);

BigInteger Yb=alpha.modPow(Xb,q);

System.out.println("Computed UserB's Public Key:"+Yb);

out.println(Yb);

System.out.println("Public Key is Sent To Attcker (Believing Its User A)...");

BigInteger Ya=new BigInteger(in.readLine());

System.out.println("Received Attacker's Public Key (Believing Its User A):"+Ya);

BigInteger Kb=Ya.modPow(Xb,q);

System.out.println("Shared Secret Key Of User A:"+Kb);

socket.close();

in.close();

out.close();

}

}

**Attacker.java:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.math.BigInteger;

import java.net.ServerSocket;

import java.net.Socket;

import java.security.SecureRandom;

public class Attacker {

public static void main(String[] args) throws IOException {

ServerSocket ssA=new ServerSocket(2729);

ServerSocket ssB=new ServerSocket(2728);

System.out.println("Attacker Is Waiting To Interupt A Exchange Between UserA And UserB...");

Socket socketA=ssA.accept();

Socket socketB=ssB.accept();

System.out.println("Attacker Succesffuly interupted UserA And UserB.");

BufferedReader inA=new BufferedReader(new InputStreamReader(socketA.getInputStream()));

PrintWriter outA=new PrintWriter(socketA.getOutputStream(),true);

BufferedReader inB=new BufferedReader(new InputStreamReader(socketB.getInputStream()));

PrintWriter outB=new PrintWriter(socketB.getOutputStream(),true);

BigInteger q=new BigInteger(inA.readLine());

BigInteger alpha=new BigInteger(inA.readLine());

System.out.println("Received q:"+q);

System.out.println("Received Alpha:"+alpha);

outB.println(q);

outB.println(alpha);

System.out.println("q and alpha are sent to UserB...");

BigInteger Xc;

SecureRandom rand=new SecureRandom();

do{

Xc=new BigInteger(q.bitLength(),rand);

}while(Xc.compareTo(q)>=0||Xc.equals(BigInteger.ZERO));

System.out.println("Generated Attacker's Private Key:"+Xc);

BigInteger Yc=alpha.modPow(Xc,q);

System.out.println("Computed Attacker's Public Key:"+Yc);

outA.println(Yc);

outB.println(Yc);

System.out.println("Attacker's Public Key is Sent To Both UserA And UserB...");

BigInteger Ya=new BigInteger(inA.readLine());

System.out.println("Received UserA's Public Key By interupting:"+Ya);

BigInteger Yb=new BigInteger(inB.readLine());

System.out.println("Received UserB's Public Key By interupting:"+Yb);

BigInteger Kb=Yb.modPow(Xc,q);

BigInteger Ka=Ya.modPow(Xc,q);

System.out.println("Shared Secret Key Of UserA:"+Ka);

System.out.println("Shared Secret Key Of UserB:"+Kb);

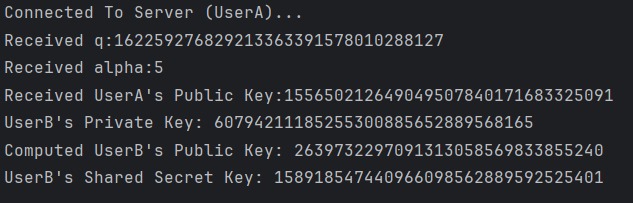
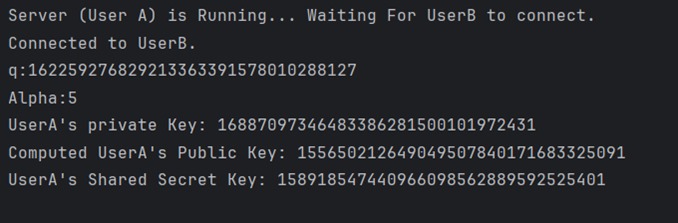
ssA.close();

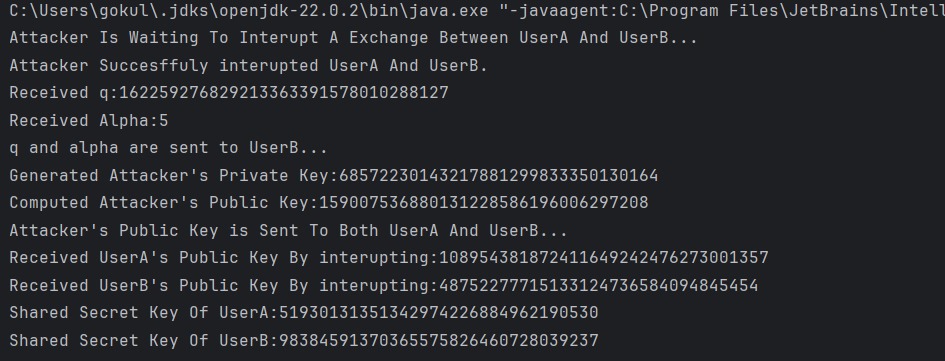
ssB.close();

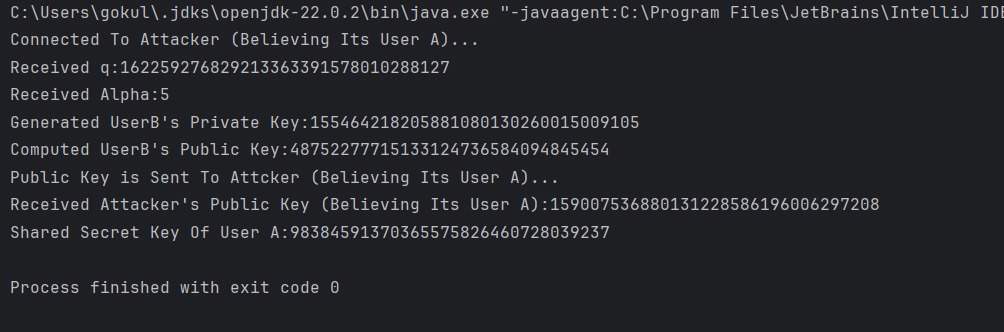
}}

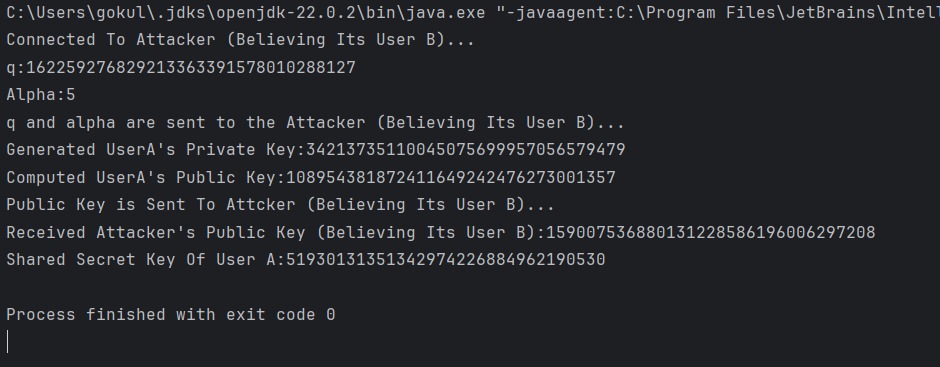
|  |
| --- |
| **SCREEN SHOTS:** |

**1.KEY EXCHANGE:**



**2.MAN IN THE MIDDLE ATTACK:** 





|  |
| --- |
| **RESULT:** |

Thus, simulated the working of Diffie Hellman in Virtual lab environment and implemented the same in Client/ Server model using Java.

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Uniqueness of the Code | 40 |  |
| Completion of experiment on time | 5 |  |
| Documentation | 10 |  |
| Simulation in Vlabs | 20 |  |
| Total | 75 |  |
| Signature of the faculty with Date |  |  |