Government College of Engineering, Jalgaon
(An Autonomous Institute of Govt. of Maharashtra)

Name: **PRN:**

Class: L.Y **Semester:** VII

Batch: B

Date of Performance: _____ **Date of Completion:** _____

Subject: CO407U CNSL **Subject Teacher:** Ms. Shrutika Mahajan

Practical – 9

Aim: Demonstrate how Diffie-Hellman key exchange works with Man-In-The-Middle attack.

Software Used:

Operating system: ubuntu
Programming Language: Python3

Theory:

Diffie-Hellman Key Exchange (DHKE)

Overview: The Diffie-Hellman Key Exchange (DHKE) is a method used to securely exchange cryptographic keys over a public channel. It was proposed by Whitfield Diffie and Martin Hellman in 1976 and is foundational in establishing secure communication. Unlike symmetric or asymmetric encryption schemes, DHKE is specifically for securely sharing a secret key between two parties (e.g., Alice and Bob) without transmitting the key itself.

How It Works:

1. **Public Parameters:** Alice and Bob agree on a large prime number ppp and a generator ggg (where g<pg < pg<p), both of which are public and can be used by any party.
2. **Private Keys:**
   - Alice selects a private key aaa (a random integer) and computes A=gamod pA = g^a \mod pA=gamodp, which she sends to Bob.
   - Bob selects a private key bbb and computes B=gbmod pB = g^b \mod pB=gbmodp, which he sends to Alice.
3. **Shared Secret Key:**
   - Alice computes the shared key as K=Bamod pK = B^a \mod pK=Bamodp.
   - Bob computes the shared key as K=Abmod pK = A^b \mod pK=Abmodp.

- Both calculations yield the same result: K=gabmod pK = g^{ab} \mod pK=gabmodp, which becomes their shared secret.

This key exchange allows Alice and Bob to share a secret key without exposing it to potential eavesdroppers.

Security Basis: The security of DHKE relies on the **Discrete Logarithm Problem**, which states that given ggg, ppp, and gamod pg^a \mod pgamodp, it is computationally infeasible to determine aaa.

Man-In-The-Middle (MITM) Attack

Overview: A Man-in-the-Middle (MITM) attack is a type of cyberattack where an attacker intercepts and potentially alters the communication between two parties without their knowledge. In the context of the Diffie-Hellman Key Exchange, an MITM attack compromises the security of the key exchange process.

How a MITM Attack Works in DHKE:

1. **Interception:**
   - Assume Alice and Bob want to establish a shared secret key using DHKE.
   - An attacker (Mallory) intercepts the public keys that Alice and Bob send to each other.
2. **Altered Key Exchange:**
   - Mallory sends her own public key M1=gm1mod pM_1 = g^{m_1} \mod pM1 =gm1modp to Alice, pretending it is Bob's key.
   - Similarly, Mallory sends another key M2=gm2mod pM_2 = g^{m_2} \mod pM2=gm2modp to Bob, pretending it is Alice's key.
3. **Separate Keys:**
   - Alice computes KAM=M1amod pK_{AM} = M_1^a \mod pKAM=M1amodp, thinking it's shared with Bob.
   - Bob computes KBM=M2bmod pK_{BM} = M_2^b \mod pKBM=M2bmodp, thinking it's shared with Alice.
   - Mallory, knowing m1m_1m1 and m2m_2m2, can compute KAMK_{AM}KAM and KBMK_{BM}KBM to decrypt or modify the messages exchanged between Alice and Bob.
4. **Compromised Communication:**
   - Alice and Bob believe they have established a shared secret, but Mallory can read and potentially modify any communication between them.

Implications: The MITM attack shows that while DHKE itself is mathematically sound, it lacks inherent authentication. Without verifying that the public keys actually belong to the intended parties, an attacker can easily compromise the exchange.

Mitigation Strategies:

To prevent MITM attacks, protocols incorporating DHKE must include:

- **Authentication Mechanisms:** Such as digital signatures or certificates, to verify the identities of the parties.

- **Public Key Infrastructure (PKI):** Ensures the integrity of the keys by using trusted certificate authorities.

Example Scenario of DHKE with MITM Attack:

1. Alice and Bob agree on public values ppp and ggg.
2. Alice sends A=gamod pA = g^a \mod pA=gamodp to Bob, but Mallory intercepts it and sends M1=gm1mod pM_1 = g^{m_1} \mod pM1=gm1modp to Bob.
3. Bob sends B=gbmod pB = g^b \mod pB=gbmodp to Alice, but Mallory intercepts it and sends M2=gm2mod pM_2 = g^{m_2} \mod pM2=gm2modp to Alice.
4. Alice calculates KAM=M1amod pK_{AM} = M_1^a \mod pKAM=M1amodp, and Bob calculates KBM=M2bmod pK_{BM} = M_2^b \mod pKBM=M2bmodp.
5. Mallory computes both KAMK_{AM}KAM and KBMK_{BM}KBM and can decrypt and modify messages as needed.

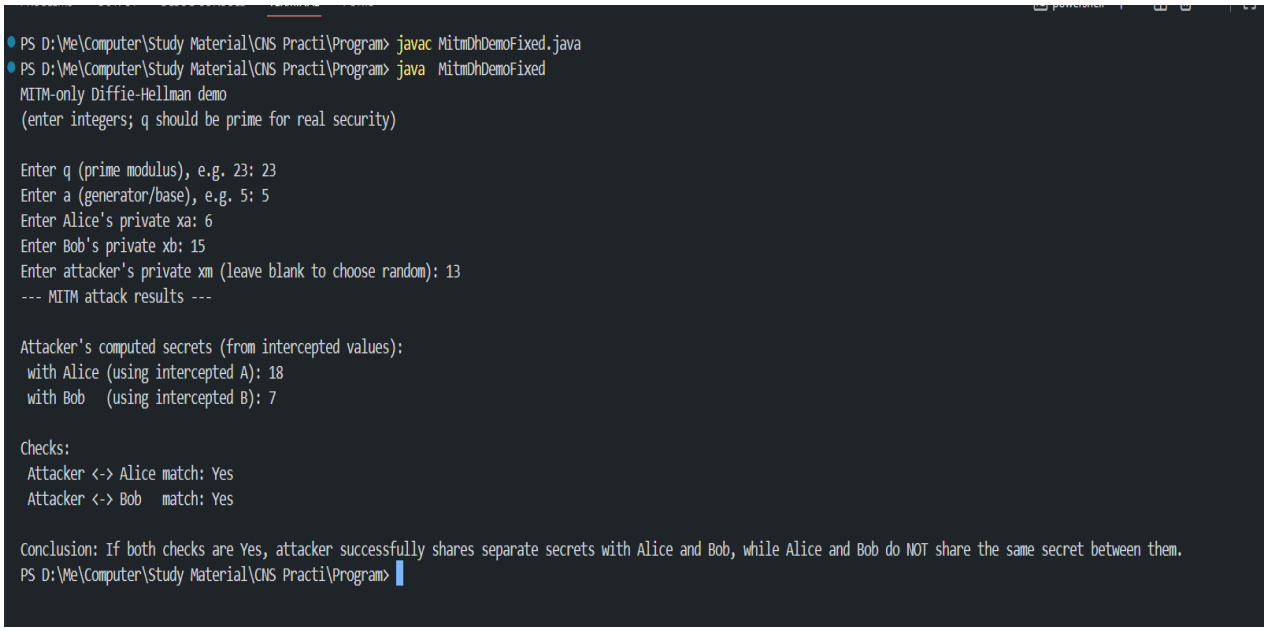
Code:

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;

public class MitmDhDemo {

    private static final SecureRandom random = new SecureRandom();

    // Read an integer BigInteger from user, retry on invalid input
    private static BigInteger getBigInt(Scanner sc, String prompt) {
        while (true) {
            System.out.print(prompt);
            String s = sc.nextLine().trim();
            try {
                return new BigInteger(s);
            } catch (Exception e) {
                System.out.println("Invalid integer, try again.");
            }
        }
    }

    // Compute a^x mod q
    private static BigInteger pub(BigInteger a, BigInteger x, BigInteger q) {
        return a.modPow(x, q);
    }

    // Compute shared: pubOther^x mod q
    private static BigInteger shared(BigInteger pubOther, BigInteger x, BigInteger
    q) {
        return pubOther.modPow(x, q);
    }
```

```
private static BigInteger randomInRangeTwoToQminus2(BigInteger q) {
    BigInteger two = BigInteger.valueOf(2);
    BigInteger max = q.subtract(two); // q - 2 + 0? we'll add 0..(max-2) below
    if (max.compareTo(two) < 0) {
        // small q; fall back
        return two;
    }
    // choose r in [0, max-2] then add 2 -> yields [2, max]
    BigInteger bound = max.subtract(two).add(BigInteger.ONE); // (max-
2)+1 = max-1
    BigInteger r;
    do {
        r = new BigInteger(bound.bitLength(), random);
    } while (r.compareTo(bound) >= 0);
    return r.add(two);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("MITM-only Diffie-Hellman demo\n(enter integers; q
should be prime for real security)\n");
    BigInteger q = getBigInt(sc, "Enter q (prime modulus), e.g. 23: ");
    BigInteger a = getBigInt(sc, "Enter a (generator/base), e.g. 5: ");
    BigInteger xa = getBigInt(sc, "Enter Alice's private xa: ");
    BigInteger xb = getBigInt(sc, "Enter Bob's private xb: ");
    System.out.print("Enter attacker's private xm (leave blank to choose
random): ");
    String xmInput = sc.nextLine().trim();
    BigInteger xm;
    if (xmInput.isEmpty()) {
        try {
            xm = randomInRangeTwoToQminus2(q);
        } catch (Exception e) {
            // fallback small random
            xm = BigInteger.valueOf(2 + random.nextInt(Math.max(1,
q.intValue() - 3)));
        }
        System.out.println("Attacker chooses xm = " + xm + " (random)\n");
    } else {
        try {
            xm = new BigInteger(xmInput);
        } catch (Exception e) {
            xm = randomInRangeTwoToQminus2(q);
            System.out.println("Invalid xm provided: choosing xm = " + xm +
" (random)\n");
        }
    }
```

```
        // MITM attack simulation
        BigInteger A_pub = pub(a, xa, q);
        BigInteger B_pub = pub(a, xb, q);
        BigInteger M_pub = pub(a, xm, q);

        BigInteger S_alice = shared(M_pub, xa, q);
        BigInteger S_bob   = shared(M_pub, xb, q);

        BigInteger S_attacker_with_alice = shared(A_pub, xm, q);
        BigInteger S_attacker_with_bob   = shared(B_pub, xm, q);

        System.out.println("--- MITM attack results ---");
        System.out.println("Alice's original public (intercepted by attacker): " +
A_pub);
        System.out.println("Bob's   original public (intercepted by attacker): " +
B_pub);
        System.out.println("Attacker sends substituted public to both: " + M_pub
+ "\n");

        System.out.println("Alice computes (thinks shared with Bob): " + S_alice);
        System.out.println("Bob   computes (thinks shared with Alice): " +
S_bob + "\n");

        System.out.println("Attacker's computed secrets (from intercepted
values):");
        System.out.println("  with Alice (using intercepted A): " +
S_attacker_with_alice);
        System.out.println("  with Bob   (using intercepted B): " +
S_attacker_with_bob + "\n");

        System.out.println("Checks:");
        System.out.println("  Attacker <-> Alice match: " +
(S_alice.equals(S_attacker_with_alice) ? "Yes" : "No"));
        System.out.println("  Attacker <-> Bob   match: " +
(S_bob.equals(S_attacker_with_bob) ? "Yes" : "No"));

        System.out.println("\nConclusion: If both checks are Yes, attacker
successfully shares separate secrets with Alice and Bob, while Alice and Bob
do NOT share the same secret between them.");
        sc.close();
    }
}
```

Output:



Conclusion:

While the Diffie-Hellman Key Exchange provides a powerful way to establish a shared secret over an untrusted channel, it is vulnerable to MITM attacks without additional measures like authentication. This vulnerability highlights the importance of securing the key exchange with methods that confirm the identities of the communicating parties.

Course Teacher
Ms. Shrutika Mahajan