

**NAME: ABISHEK K G****REG NO: 23BCE1739****DATE: 13/02/2026****SLOT: L29+L30****Multi-Party Diffie–Hellman Key Exchange and MITM Attack Simulation****1. Introduction**

Secure communication over insecure networks is a fundamental requirement in distributed systems. One of the foundational methods used to establish a shared secret over an open channel is the Diffie–Hellman key exchange algorithm. This protocol allows multiple participants to agree upon a common secret key without directly transmitting the key itself.

In this experiment, a multi-party Diffie–Hellman key exchange was implemented using socket programming. Additionally, a Man-in-the-Middle (MITM) attack was simulated to demonstrate the vulnerability of the protocol when authentication mechanisms are not present.

---

**2. Multi-Party Diffie–Hellman Key Exchange****2.1 Concept Overview**

The Diffie–Hellman key exchange is based on modular arithmetic and the mathematical difficulty of solving the discrete logarithm problem.

It works using:

- A publicly known prime number  $p$
- A primitive root (generator)  $g$
- Private keys selected independently by each participant
- Public keys computed using modular exponentiation

Each participant:

1. Chooses a private key.
2. Computes a public key using the formula:

$$PublicKey = g^{privateKey} \bmod p$$

3. Shares the public key openly.

4. Uses the received public keys to compute the shared secret.

Even though public keys are exchanged openly, the shared secret cannot be easily computed without knowing the private keys.

---

## 2.2 Multi-Party Extension

In a two-party scenario, both participants compute:

$$SharedSecret = (OtherPublicKey)^{privateKey} \bmod p$$

For multiple participants, the process extends using sequential modular exponentiation. Each participant applies their private key to the received public keys to derive a final shared secret.

In normal operation:

- All participants derive the same final shared key.
  - The shared key is never transmitted directly.
  - Only public keys are exchanged.
- 

## 2.3 Observations in Normal Mode

- All clients successfully generated private and public keys.
- The server relayed public keys correctly.
- Each participant independently computed the same shared secret.
- Encrypted messages were successfully decrypted using the shared key.
- Secure communication was achieved without directly transmitting the secret.

This demonstrates the correctness of the Diffie–Hellman mechanism.

---

## 3. Man-in-the-Middle (MITM) Attack

### 3.1 Concept Overview

Although Diffie–Hellman allows secure key agreement, it does not provide authentication.

This means:

- Participants do not verify the identity of the sender of public keys.

- An attacker can intercept and modify public keys during exchange.

A Man-in-the-Middle attack exploits this weakness.

---

### 3.2 How the Attack Works

In the simulated MITM scenario:

1. The attacker intercepts the public keys exchanged between clients.
2. The attacker replaces the original public keys with their own public key.
3. Each client unknowingly establishes a shared key with the attacker instead of the intended participant.
4. The attacker now possesses:
  - One shared key with Client A
  - A different shared key with Client B

The attacker can then:

- Decrypt messages from Client A.
- Modify the message.
- Re-encrypt it using the key shared with Client B.
- Forward it without detection.

Both clients believe they are communicating securely with each other.

---

### 3.3 Observations in MITM Mode

- Clients derived different shared secrets.
- The attacker derived separate keys with each client.
- The attacker successfully decrypted intercepted messages.
- The attacker modified and re-encrypted messages.
- The receiving client decrypted the modified message without detecting tampering.

This clearly demonstrates that Diffie–Hellman alone does not guarantee secure communication.

---

## 4. Message Encryption Mechanism

To simulate secure communication, a simple XOR-based encryption was implemented.

Encryption Process:

- Convert message into byte array.
- XOR each byte with the shared secret.
- Send encrypted data.

Decryption Process:

- XOR encrypted bytes again with the same shared secret.
- Original message is restored.

Although XOR encryption is not secure for real-world applications, it was sufficient to demonstrate:

- Correct key establishment.
  - Successful encryption/decryption.
  - MITM message interception and modification.
- 

## 5. Security Analysis

### 5.1 Strength of Diffie–Hellman

- Secure against passive eavesdropping.
- Secret key is never transmitted.
- Security based on discrete logarithm hardness.

### 5.2 Weakness Identified

- No authentication mechanism.
- Vulnerable to active attacks.
- Public keys can be replaced without detection.

The MITM simulation clearly proves this vulnerability.

---

## 6. Security Recommendations

To prevent MITM attacks, authentication mechanisms must be added. Recommended solutions include:

1. Digital Signatures
  - Sign public keys using private signing keys.
  - Verify signatures before accepting public keys.
2. Public Key Infrastructure (PKI)
  - Use certificates issued by trusted authorities.
  - Validate identity before key exchange.
3. Authenticated Diffie–Hellman
  - Combine DH with digital certificates (e.g., TLS handshake).
4. Hash-Based Key Confirmation
  - Exchange hashed verification values of derived keys.

Without authentication, Diffie–Hellman remains vulnerable.

---

## 7. Conclusion

This experiment demonstrated both the strength and weakness of the Diffie–Hellman key exchange protocol.

In normal mode:

- All participants derived the same shared secret.
- Secure encrypted communication was achieved.

In MITM mode:

- The attacker successfully intercepted and altered communication.
- Clients unknowingly shared keys with the attacker.
- Message integrity was compromised.

The experiment highlights a critical cybersecurity principle:

Key exchange alone is not sufficient. Authentication is essential for secure communication.

### Code implementation:

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

public class MultiPartyDH {

    // Public DH parameters
    static final BigInteger prime = BigInteger.valueOf(23);
    static final BigInteger generator = BigInteger.valueOf(5);
    static final int serverPort = 9876;
    static final int attackerPort = 9877;
    static final int partyCount = 3;

    // XOR encryption
    static byte[] xorEncryptDecrypt(byte[] input, BigInteger key) {
        int keyByte = key.mod(BigInteger.valueOf(256)).intValue();
        byte[] output = new byte[input.length];
        for (int i = 0; i < input.length; i++) {
            output[i] = (byte) (input[i] ^ keyByte);
        }
        return output;
    }
}
```

**// Encrypt message with key**

```
static String encryptMsg(String msg, BigInteger key) {  
    byte[] encrypted = xorEncryptDecrypt(msg.getBytes(), key);  
    StringBuilder sb = new StringBuilder();  
    for (byte b : encrypted) {  
        sb.append(String.format("%02X", b));  
    }  
    return sb.toString();  
}
```

**// Decrypt message with key**

```
static String decryptMsg(String hexCipher, BigInteger key) {  
    byte[] encrypted = new byte[hexCipher.length() / 2];  
    for (int i = 0; i < encrypted.length; i++) {  
        encrypted[i] = (byte) Integer.parseInt(hexCipher.substring(2 * i, 2 * i + 2), 16);  
    }  
    byte[] decrypted = xorEncryptDecrypt(encrypted, key);  
    return new String(decrypted);  
}
```

**// Print section header**

```
static void printSection(String title) {  
    System.out.println("\n" + "=".repeat(60));  
    System.out.println(" " + title);  
    System.out.println("=".repeat(60));  
}
```

**// Print subsection**

```
static void printSubSection(String title) {  
    System.out.println("\\n--- " + title + " ---");  
}
```

**// Server for normal mode**

```
static class PartyServer implements Runnable {  
    private final int port;  
    private final int expectedCount;  
    private final CountDownLatch readySignal;  
    private final Map<Integer, BigInteger> partyKeys =  
Collections.synchronizedMap(new LinkedHashMap<>());  
    private final List<Socket> partySockets = Collections.synchronizedList(new  
ArrayList<>());
```

```
    PartyServer(int port, int expectedCount, CountDownLatch readySignal) {  
        this.port = port;  
        this.expectedCount = expectedCount;  
        this.readySignal = readySignal;  
}
```

**@Override**

```
public void run() {  
    try (ServerSocket serverSocket = new ServerSocket(port)) {  
        serverSocket.setReuseAddress(true);  
        printSection("SERVER STARTED on port " + port);  
        System.out.println(" Waiting for " + expectedCount + " parties...");  
        readySignal.countDown();
```

```

// Accept connections and collect keys
for (int i = 0; i < expectedCount; i++) {
    Socket sock = serverSocket.accept();
    partySockets.add(sock);

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

    String line = in.readLine();

    String[] parts = line.split(":");

    int partyId = Integer.parseInt(parts[1]);

    BigInteger pubKey = new BigInteger(parts[2]);

    partyKeys.put(partyId, pubKey);

    System.out.println(" [Server] Got key from Party " + partyId + " : " +
pubKey);
}

printSubSection("SERVER: All keys collected");

// Send all keys to each party
StringBuilder keyList = new StringBuilder("ALLKEYS:");

int idx = 0;

for (Map.Entry<Integer, BigInteger> entry : partyKeys.entrySet()) {
    if (idx > 0) keyList.append(",");

    keyList.append(entry.getKey()).append("=").append(entry.getValue());

    idx++;
}

String broadcast = keyList.toString();

for (Socket sock : partySockets) {

```

```

        PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
        out.println(broadcast);
    }

    System.out.println(" [Server] Sent all keys to parties.");

    // Relay encrypted messages

    Thread.sleep(500);

    printSubSection("SERVER: Message Relay");

    Map<Integer, String> messages = new LinkedHashMap<>();

    for (int i = 0; i < partySockets.size(); i++) {

        Socket sock = partySockets.get(i);

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

        String msg = in.readLine();

        if (msg != null && msg.startsWith("MSG:")) {

            String[] parts = msg.split(":", 3);

            int senderId = Integer.parseInt(parts[1]);

            String encryptedHex = parts[2];

            messages.put(senderId, encryptedHex);

            System.out.println(" [Server] Got encrypted msg from Party " + senderId +
" : " + encryptedHex);

        }

    }

    // Send all messages to each party

    StringBuilder msgBroadcast = new StringBuilder("MESSAGES:");

    idx = 0;

    for (Map.Entry<Integer, String> entry : messages.entrySet()) {

```

```

        if (idx > 0) msgBroadcast.append("|");
        msgBroadcast.append(entry.getKey()).append("=").append(entry.getValue());
        idx++;
    }

    String msgStr = msgBroadcast.toString();
    for (Socket sock : partySockets) {
        PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
        out.println(msgStr);
    }

    System.out.println(" [Server] Relayed all encrypted messages to clients.");

    // Allow clients time to process before closing
    Thread.sleep(500);

    for (Socket sock : partySockets) {
        sock.close();
    }

} catch (Exception e) {
    System.err.println(" [Server] Error: " + e.getMessage());
}

}

}

```

// Party client

```

static class PartyClient implements Runnable {
    private final int partyId;
    private final int port;
    private final CountDownLatch readySignal;

```

```
private BigInteger mySecret;  
private BigInteger myPublic;  
private BigInteger sharedKey;  
private final String msgToSend;
```

```
PartyClient(int partyId, int port, CountDownLatch readySignal, String msg) {  
    this.partyId = partyId;  
    this.port = port;  
    this.readySignal = readySignal;  
    this.msgToSend = msg;  
}
```

*@Override*

```
public void run() {  
    try {  
        readySignal.await();  
        Thread.sleep(100 * partyId);  
        Socket socket = new Socket("localhost", port);  
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
        BufferedReader in = new BufferedReader(new  
InputStreamReader(socket.getInputStream()));  
  
        // Generate keys  
        Random rand = new Random();  
        mySecret = BigInteger.valueOf(rand.nextInt(20) + 1);  
        myPublic = generator.modPow(mySecret, prime);  
        System.out.println(" [Party " + partyId + "] Secret = " + mySecret + ", Public =  
" + myPublic);  
    }  
}
```

```

out.println("PUBKEY:" + partyId + ":" + myPublic);

// Receive all keys
String response = in.readLine();
if (response != null && response.startsWith("ALLKEYS:")) {
    String data = response.substring(8);
    String[] entries = data.split(",");
    List<BigInteger> others = new ArrayList<>();
    for (String entry : entries) {
        String[] kv = entry.split("=");
        int otherId = Integer.parseInt(kv[0]);
        BigInteger otherPub = new BigInteger(kv[1]);
        if (otherId != partyId) {
            others.add(otherPub);
        }
    }

    // Compute shared key
    sharedKey = BigInteger.ONE;
    for (BigInteger otherPub : others) {
        BigInteger partial = otherPub.modPow(mySecret, prime);
        sharedKey = sharedKey.multiply(partial).mod(prime);
    }

    System.out.println(" [Party " + partyId + "] Shared key = " + sharedKey);
}

// Send encrypted message
Thread.sleep(300);

```

```

    String encrypted = encryptMsg(msgToSend, sharedKey);

    out.println("MSG:" + partyId + ":" + encrypted);

    System.out.println(" [Party " + partyId + "] Sent encrypted msg: \"" +
msgToSend + "\" -> " + encrypted);

// Receive messages

String msgResp = in.readLine();

if (msgResp != null && msgResp.startsWith("MESSAGES:")) {

    String msgData = msgResp.substring(9);

    String[] msgEntries = msgData.split("\\|");

    for (String me : msgEntries) {

        String[] kv = me.split("=", 2);

        int senderId = Integer.parseInt(kv[0]);

        if (senderId != partyId) {

            String decrypted = decryptMsg(kv[1], sharedKey);

            System.out.println(" [Party " + partyId + "] Decrypted msg from Party "
+ senderId + " : \"" + decrypted + "\"");

        }

    }

    socket.close();

} catch (Exception e) {

    System.err.println(" [Party " + partyId + "] Error: " + e.getMessage());

}

}

}

// Attacker (MITM)

```

```

static class Attacker implements Runnable {
    private final int listenPort;
    private final int realServerPort;
    private final int partyCount;
    private final CountDownLatch attackerReady;
    private final CountDownLatch serverReady;
    private final BigInteger mySecret;
    private final BigInteger myPublic;
    private final Map<Integer, BigInteger> partySecrets =
Collections.synchronizedMap(new LinkedHashMap<>());
    private final Map<Integer, BigInteger> originalKeys =
Collections.synchronizedMap(new LinkedHashMap<>());

    Attacker(int listenPort, int realServerPort, int partyCount, CountDownLatch
attackerReady, CountDownLatch serverReady) {
        this.listenPort = listenPort;
        this.realServerPort = realServerPort;
        this.partyCount = partyCount;
        this.attackerReady = attackerReady;
        this.serverReady = serverReady;
        Random rand = new Random();
        this.mySecret = BigInteger.valueOf(rand.nextInt(20) + 1);
        this.myPublic = generator.modPow(mySecret, prime);
    }

    @Override
    public void run() {
        try (ServerSocket attackerSocket = new ServerSocket(listenPort)) {

```

```
attackerSocket.setReuseAddress(true);  
printSection("ATTACKER STARTED on port " + listenPort);  
System.out.println(" Attacker secret = " + mySecret + ", public = " + myPublic);  
attackerReady.countDown();  
serverReady.await();
```

```
List<Socket> partyConns = new ArrayList<>();  
List<Socket> serverConns = new ArrayList<>();  
List<BufferedReader> partyReaders = new ArrayList<>();  
List<PrintWriter> partyWriters = new ArrayList<>();  
List<BufferedReader> serverReaders = new ArrayList<>();  
List<PrintWriter> serverWriters = new ArrayList<>();  
List<Integer> partyIds = new ArrayList<>();
```

```
for (int i = 0; i < partyCount; i++) {  
    Socket partyConn = attackerSocket.accept();  
    partyConns.add(partyConn);  
    BufferedReader cReader = new BufferedReader(new  
InputStreamReader(partyConn.getInputStream()));  
    PrintWriter cWriter = new PrintWriter(partyConn.getOutputStream(), true);  
    partyReaders.add(cReader);  
    partyWriters.add(cWriter);  
    String line = cReader.readLine();  
    String[] parts = line.split(":");  
    int partyId = Integer.parseInt(parts[1]);  
    BigInteger partyPub = new BigInteger(parts[2]);  
    partyIds.add(partyId);
```

```

        originalKeys.put(partyId, partyPub);

        printSubSection("MITM: Intercepted Party " + partyId); // Intercepted party
        System.out.println(" [Attacker] Got key from Party " + partyId + " : " +
partyPub);

        BigInteger secretWithParty = partyPub.modPow(mySecret, prime); // Compute
shared secret

        partySecrets.put(partyId, secretWithParty);

        System.out.println(" [Attacker] Shared secret with Party " + partyId + " = " +
secretWithParty);

        Socket serverConn = new Socket("localhost", realServerPort);

        serverConns.add(serverConn);

        BufferedReader sReader = new BufferedReader(new
InputStreamReader(serverConn.getInputStream()));

        PrintWriter sWriter = new PrintWriter(serverConn.getOutputStream(), true);

        serverReaders.add(sReader);

        serverWriters.add(sWriter);

        String fakeKey = "PUBKEY:" + partyId + ":" + myPublic; // Replace with
attacker's key

        sWriter.println(fakeKey);

        System.out.println(" [Attacker] Sent FAKE key to server for Party " + partyId
+ " : " + myPublic);
    }

    printSubSection("MITM: Intercepting Key Broadcasts"); // Intercept key
broadcasts

    for (int i = 0; i < partyCount; i++) {

        String serverResp = serverReaders.get(i).readLine();

        System.out.println(" [Attacker] Got key list from server for slot " + i + " : " +
serverResp);

        if (serverResp != null && serverResp.startsWith("ALLKEYS:")) {

```

```

    String data = serverResp.substring(8);
    String[] entries = data.split(",");
    StringBuilder fakeList = new StringBuilder("ALLKEYS:");
    int idx = 0;
    for (String entry : entries) {
        String[] kv = entry.split("=");
        int eid = Integer.parseInt(kv[0]);
        if (idx > 0) fakeList.append(",");
        fakeList.append(eid).append("=").append(myPublic);
        idx++;
    }
    String fakeResp = fakeList.toString();
    partyWriters.get(i).println(fakeResp);
    System.out.println(" [Attacker] Sent FAKE key list to Party " +
partyIds.get(i) + " : " + fakeResp); // Send fake key list
    }
}

```

```

Thread.sleep(800);
printSubSection("MITM: Intercepting Messages"); // Intercept messages
Map<Integer, String> interceptedMsgs = new LinkedHashMap<>();
for (int i = 0; i < partyCount; i++) {
    String msgLine = partyReaders.get(i).readLine();
    if (msgLine != null && msgLine.startsWith("MSG:")) {
        String[] parts = msgLine.split(":", 3);
        int senderId = Integer.parseInt(parts[1]);
        String encryptedHex = parts[2];
    }
}

```

```

        interceptedMsgs.put(senderId, encryptedHex);

        BigInteger keyWithSender = partySecrets.get(senderId);

        String decrypted = decryptMsg(encryptedHex, keyWithSender); // Decrypt
        System.out.println(" [Attacker] Got msg from Party " + senderId + " :
cipher=" + encryptedHex);

        System.out.println(" [Attacker] Decrypted: \"" + decrypted + "\"");

        String modified = "[TAMPERED] " + decrypted; // Tamper

        System.out.println(" [Attacker] Modified: \"" + modified + "\"");

        String reEncrypted = encryptMsg(modified, keyWithSender); // Re-encrypt

        String fakeMsg = "MSG:" + senderId + ":" + reEncrypted;

        serverWriters.get(i).println(fakeMsg);

        System.out.println(" [Attacker] Sent tampered msg to server for Party " +
senderId); // Forward tampered
    }

}

    printSubSection("MITM: Forwarding Tampered Messages"); // Forward
tampered messages

    for (int i = 0; i < partyCount; i++) {

        String msgResp = serverReaders.get(i).readLine();

        if (msgResp != null && msgResp.startsWith("MESSAGES:")) {

            System.out.println(" [Attacker] Got message broadcast from server for slot
" + i);

            partyWriters.get(i).println(msgResp);

            System.out.println(" [Attacker] Sent tampered broadcast to Party " +
partyIds.get(i)); // Send tampered broadcast
        }

    }
}

```

```

        printSubSection("MITM SUMMARY"); // Print summary

        System.out.println(" Attacker secret: " + mySecret);

        System.out.println(" Attacker public: " + myPublic);

        for (Map.Entry<Integer, BigInteger> entry : partySecrets.entrySet()) {

            System.out.println(" Shared secret with Party " + entry.getKey() + " : " +
entry.getValue());

        }

        System.out.println(" Attacker can read and modify all messages.");

        Thread.sleep(500);

        for (Socket s : partyConns) s.close();

        for (Socket s : serverConns) s.close();

    } catch (Exception e) {

        System.err.println(" [Attacker] Error: " + e.getMessage());

        e.printStackTrace();

    }

}

}

```

```
//
```

```
=====
==
```

```
// NORMAL MODE: Multi-party DH key exchange without attacker
```

```
//
```

```
=====
==
```

```
static void runNormalMode() throws Exception { // Normal mode
```

```
    printSection("DIFFIE-HELLMAN EXCHANGE (Normal Mode)");
```

```
    System.out.println(" p = " + prime + ", g = " + generator);
```

```
    System.out.println(" Parties: " + partyCount);
```

```

    CountDownLatch serverReady = new CountDownLatch(1);

    Thread serverThread = new Thread(new PartyServer(serverPort, partyCount,
serverReady));

    serverThread.start();

    printSubSection("Starting Parties");

    String[] messages = {
        "Hello from Party 1!",
        "Greetings from Party 2!",
        "Hi from Party 3!"
    };

    List<Thread> partyThreads = new ArrayList<>();
    for (int i = 1; i <= partyCount; i++) {
        Thread t = new Thread(new PartyClient(i, serverPort, serverReady, messages[i -
1]));

        partyThreads.add(t);

        t.start();
    }

    serverThread.join();

    for (Thread t : partyThreads) t.join();

    printSection("NORMAL MODE DONE");

    System.out.println(" All parties derived the same shared key and exchanged
messages.");
}

//
=====

// MITM MODE: Multi-party DH with Man-in-the-Middle attacker

```

```
//
=====

static void runMitmMode() throws Exception { // MITM mode

    printSection("DIFFIE-HELLMAN WITH MITM ATTACK");

    System.out.println(" p = " + prime + ", g = " + generator);

    System.out.println(" Parties: " + partyCount);

    System.out.println(" Attacker will intercept and tamper!");

    CountDownLatch serverReady = new CountDownLatch(1);

    CountDownLatch attackerReady = new CountDownLatch(1);

    Thread serverThread = new Thread(new PartyServer(serverPort, partyCount,
serverReady));

    serverThread.start();

    Thread attackerThread = new Thread(new Attacker(attackerPort, serverPort,
partyCount, attackerReady, serverReady));

    attackerThread.start();

    attackerReady.await();

    Thread.sleep(200);

    printSubSection("Starting Parties (connecting to Attacker)");

    String[] messages = {

        "Secret msg from Party 1",

        "Confidential from Party 2",

        "Private data from Party 3"

    };

    List<Thread> partyThreads = new ArrayList<>();

    for (int i = 1; i <= partyCount; i++) {

        Thread t = new Thread(new PartyClient(i, attackerPort, attackerReady, messages[i
- 1]));

        partyThreads.add(t);
    }
}
```

```

        t.start();
    }
    serverThread.join();
    attackerThread.join();
    for (Thread t : partyThreads) t.join();
    printSection("MITM MODE DONE");
    System.out.println(" Attacker intercepted, replaced keys, and tampered messages.");
}

//
=====

// MAIN ENTRY POINT
//
=====

public static void main(String[] args) {
    if (args.length < 1) {
        System.out.println("Usage: java MultiPartyDH <normal|mitm>");
        return;
    }
    try {
        switch (args[0].toLowerCase()) {
            case "normal":
                runNormalMode();
                break;
            case "mitm":
                runMitmMode();

```

```

        break;

    default:

        System.out.println("Unknown mode: " + args[0]);

        System.out.println("Usage: java MultiPartyDH <normal|mitm>");

    }

} catch (Exception e) {

    System.err.println("Fatal error: " + e.getMessage());

    e.printStackTrace();

}

}

}

```

## Output screenshots:

```

PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 5> java MultiPartyDH normal

=====
DIFFIE-HELLMAN EXCHANGE (Normal Mode)
=====
p = 23, g = 5
Parties: 3

--- Starting Parties ---

=====
SERVER STARTED on port 9876
=====
Waiting for 3 parties...
[Party 1] Secret = 2, Public = 2
[Server] Got key from Party 1 : 2
[Party 2] Secret = 11, Public = 22
[Server] Got key from Party 2 : 22
[Party 3] Secret = 3, Public = 10
[Server] Got key from Party 3 : 10

--- SERVER: All keys collected ---
[Server] Sent all keys to parties.
[Party 1] Shared key = 8
[Party 2] Shared key = 22
[Party 3] Shared key = 15
[Party 3] Sent encrypted msg: "Hi from Party 3!" -> 47662F697D60622F5F6E7D7B762F3C2E
[Party 1] Sent encrypted msg: "Hello from Party 1!" -> 406D646467286E7A67652858697A7C71283929
[Party 2] Sent encrypted msg: "Greetings from Party 2!" -> 51647373627F787165367064797B36467764626F362437

--- SERVER: Message Relay ---
[Server] Got encrypted msg from Party 1 : 406D646467286E7A67652858697A7C71283929
[Server] Got encrypted msg from Party 2 : 51647373627F787165367064797B36467764626F362437
[Server] Got encrypted msg from Party 3 : 47662F697D60622F5F6E7D7B762F3C2E
[Server] Relayed all encrypted messages to clients.
[Party 2] Decrypted msg from Party 1 : "V{rrq>xlqs>Nljg>/?"
[Party 1] Decrypted msg from Party 2 : "Yl{[[jwpym>xlqs>Nljg>?"
[Party 3] Decrypted msg from Party 1 : "Obkkh'auhj'wfus-'66"
[Party 2] Decrypted msg from Party 3 : "Qp9kvt9Ixm'9+8"
[Party 1] Decrypted msg from Party 3 : "On'auhj'wfus-'45"
[Party 3] Decrypted msg from Party 2 : ""k||mpw~j9kvt9Ixm'9+8"

=====
NORMAL MODE DONE
=====
All parties derived the same shared key and exchanged messages.
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 5>

```

```
Windows PowerShell
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 5> java MultiPartyOH mitm

=====
DIFFIE-HELLMAN WITH MITM ATTACK
=====
p = 23, g = 5
Parties: 3
Attacker will intercept and tamper!

=====
SERVER STARTED on port 9876
=====

ATTACKER STARTED on port 9877
=====

Waiting for 3 parties...
Attacker secret = 3, public = 10

--- Starting Parties (connecting to Attacker) ---
[Party 1] Secret = 13, Public = 21

--- MITM: Intercepted Party 1 ---
[Attacker] Got key from Party 1 : 21
[Attacker] Shared secret with Party 1 = 15
[Attacker] Sent FAKE key to server for Party 1 : 10
[Server] Got key from Party 1 : 10
[Party 2] Secret = 10, Public = 9

--- MITM: Intercepted Party 2 ---
[Attacker] Got key from Party 2 : 9
[Attacker] Shared secret with Party 2 = 16
[Attacker] Sent FAKE key to server for Party 2 : 10
[Server] Got key from Party 2 : 10
[Party 3] Secret = 10, Public = 9

--- MITM: Intercepted Party 3 ---
[Attacker] Got key from Party 3 : 9
[Attacker] Shared secret with Party 3 = 16
[Attacker] Sent FAKE key to server for Party 3 : 10
[Server] Got key from Party 3 : 10

--- MITM: Intercepting Key Broadcasts ---

--- SERVER: All keys collected ---
[Server] Sent all keys to parties.
[Attacker] Got key list from server for slot 0 : ALLKEYS:1=10,2=10,3=10
[Attacker] Sent FAKE key list to Party 1 : ALLKEYS:1=10,2=10,3=10
[Attacker] Got key list from server for slot 1 : ALLKEYS:1=10,2=10,3=10
```

```
Windows PowerShell

[Party 1] Shared key = 18
[Attacker] Sent FAKE key list to Party 2 : ALLKEYS:1=10,2=10,3=10
[Attacker] Got key list from server for slot 2 : ALLKEYS:1=10,2=10,3=10
[Party 2] Shared key = 3
[Attacker] Sent FAKE key list to Party 3 : ALLKEYS:1=10,2=10,3=10
[Party 3] Shared key = 3
[Party 3] Sent encrypted msg: "Private data from Party 3" -> 53716A7562776623676277622365716C6E23536271777A2330
[Party 2] Sent encrypted msg: "Confidential from Party 2" -> 486C6D656A67666D776A626F2365716C6E23536271777A2331
[Party 1] Sent encrypted msg: "Secret msg from Party 1" -> 417771687766327F61753274687D7F3242736866683223

--- SERVER: Message Relay ---

--- MITM: Intercepting Messages ---
[Attacker] Got msg from Party 1 : cipher=417771687766327F61753274687D7F3242736866683223
[Attacker] Decrypted: "Nx~oxi~pnz={orp=M}oid=,"
[Attacker] Modified: "[TAMPERED] Nx~oxi~pnz={orp=M}oid=,"
[Server] Got encrypted msg from Party 1 : 545B4E425F4A5D4A4B522F417771687766327F61753274687D7F3242736866683223
[Attacker] Sent tampered msg to server for Party 1
[Attacker] Got msg from Party 2 : cipher=486C6D656A67666D776A626F2365716C6E23536271777A2331
[Attacker] Decrypted: "p|juzwvlgz3ua|~3Cragj3!"
[Attacker] Modified: "[TAMPERED] p|juzwvlgz3ua|~3Cragj3!"
[Attacker] Sent tampered msg to server for Party 2
[Server] Got encrypted msg from Party 2 : 4B44515D48554255544D3B486C6D656A67666D776A626F2365716C6E23536271777A2331
[Attacker] Got msg from Party 3 : cipher=53716A7562776623676277622365716C6E23536271777A2330
[Attacker] Decrypted: "Cazergv3wrg3ua|~3Cragj3 "
[Attacker] Modified: "[TAMPERED] Cazergv3wrg3ua|~3Cragj3 "
[Attacker] Sent tampered msg to server for Party 3

--- MITM: Forwarding Tampered Messages ---
[Server] Got encrypted msg from Party 3 : 4B44515D48554255544D3B486C6D656A67666D776A626F2365716C6E23536271777A2330
[Server] Relayed all encrypted messages to clients.
[Attacker] Got message broadcast from server for slot 0
[Attacker] Sent tampered broadcast to Party 1
[Attacker] Got message broadcast from server for slot 1
[Attacker] Sent tampered broadcast to Party 2
[Attacker] Got message broadcast from server for slot 2
[Attacker] Sent tampered broadcast to Party 3

--- MITM SUMMARY ---
Attacker secret: 3
Attacker public: 10
Shared secret with Party 1 : 15
Shared secret with Party 2 : 16
Shared secret with Party 3 : 16
Attacker can read and modify all messages.
[Party 1] Decrypted msg from Party 2 : "YVCORGPGEF~"R~wxutexpj|wlc~|1Apceh1#"
[Party 1] Decrypted msg from Party 3 : "YVCORGPGEF~"Acxgpetlupeplwlc~|1Apceh1""
[Party 3] Decrypted msg from Party 1 : "WXMA~I~IHQ,8trctel|bv|wlc~|1Apceh1 "
[Party 2] Decrypted msg from Party 1 : "WXMA~I~IHQ,8trctel|bv|wlc~|1Apceh1 "

[Party 3] Decrypted msg from Party 2 : "HGR^CVAVMN3Confidential from Party 2"
[Party 2] Decrypted msg from Party 3 : "HGR^CVAVMN3Private data from Party 3"

=====
MITM MODE DONE
=====

Attacker intercepted, replaced keys, and tampered messages.
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 5> |
```