FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
Department of Computer Engineering

Course , Subject & Experiment Details

Academic Year	**2024-25**	**Estimated Time**	**02 - Hours**
Course & Semester	**T.E. (CMPN)- Sem VI**	**Subject Name & Code**	**CSS – CSC602**
Module No.	**02 – Mapped to CO-2**	**Chapter Title**	**Key Management Techniques**

Practical No:	**2**
Title:	**Implementation of Diffie- Hellman Key exchange algorithm and Simulation of Man In the Middle attack**
Date of Performance:	20/02/2025
Date of Submission:	26/02/2025
Roll No:	9913
Name of the Student:	Mark Lopes

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher:

Date:

Title: Implementation of Diffie- Hellman Key exchange algorithm and Simulation of Man In the Middle attack.

Lab Objective :

This lab provides insight into:

· The working of Diffie – Hellman Key Exchange Protocol.

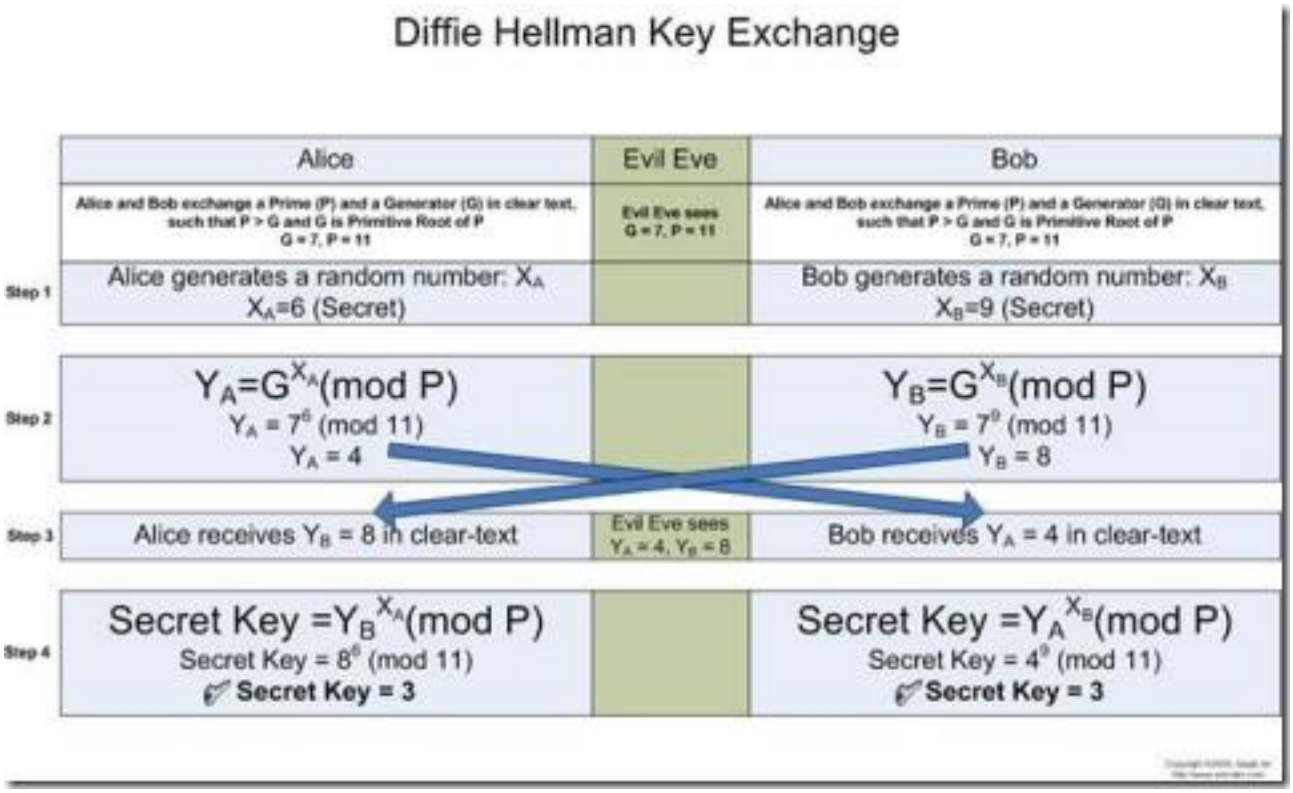
Reference : "Cryptography and Network Security" B. A. Forouzan
"Cryptography and Network Security" Atul Kahate

Prerequisite: Any programming Language and Knowledge of Symmetric Key cryptography.

Theory:

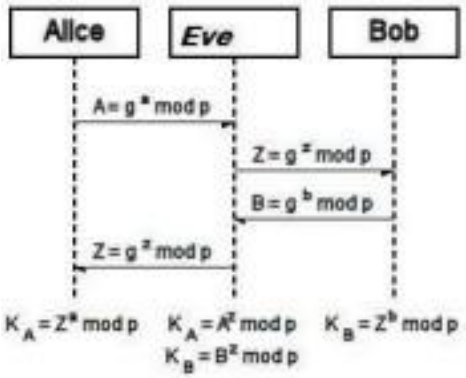
Diffie-Hellman is a way of *generating* a shared secret between two people in such a way that the secret can't be seen by observing the communication.

This is particularly useful because you can use this technique to create an encryption key with someone, and then start encrypting your traffic with that key. And even if the traffic is recorded and later analyzed, there's absolutely no way to figure out what the key was, even though the exchanges that created it may have been visible.



Man – In – The –Middle Attack

Let us take the example illustrated by Diffie-Hellman to discuss the Man-in-the-Middle Attack. Let us that Eve is in the middle of Alice and Bob. Eve does not need the value of x or y to attack the protocol. She can fool both Alice and Bob by the following process.



1. Alice choose a, calculate A=g^a mod p
2. Eve, the intruder, interpret A, she chooses z, calculate Z=g^z mod p, and sends Z to both Alice and Bob.
3. Bob choose b, calculate B=g^b mod p, and sends B to Alice; B is interpreted by Eve and

never reaches Alice.

4. Alice and Eve calculate the same key g^az mod p, which become a shared key between Alice and Eve. Alice however think that it is a key shared between Bob and herself.

5. Eve and Bob calculate the same key g^bz mod p, which become a shared key between Eve and Bob. Bob, however, thinks that it is a key shared between Alice and himself.

This situation is called man-in-the-middle attack.

Practical and Real Time Applications

- Used as a method of exchanging cryptography keys for **use** in symmetric encryption algorithms like AES
- Public key encryption schemes based on DF – ElGamal encryption
- Password-authenticated key agreement
- public key infrastructure - It is possible to use DF as part of PKI

Conclusion:
The program was tested for different sets of inputs. Program is working SATISFACTORY NOT SATISFACTORY (Tick appropriate outcome)

Post Lab Assignment:
1. In the Diffie- Hellman protocol , what happens if x and y have the same value, that is, Alice and Bob have accidentally chosen the same number? Are A and B (values exchanged by Alice and Bob to each other) the same? Do the session keys calculated by Alice and Bob have the same value? Use an example to prove your claims. 2. How to secure Diffie-Hellman from Man-in –the –Middle attack?

Client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <gmp.h>

#pragma comment(lib, "ws2_32.lib")

#define BUFFER_SIZE 1024

int main() {
    WSADATA wsaData;
    SOCKET sock = INVALID_SOCKET;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};
    char message[BUFFER_SIZE] = {0};
    int result;

    // Initialize Winsock
    result = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        printf("WSAStartup failed: %d\n", result);
        return 1;
    }

    // Initialize random number generator
    srand((unsigned int)time(NULL));

    // Create socket
    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock == INVALID_SOCKET) {
        printf("Socket creation error: %d\n", WSAGetLastError());
        WSACleanup();
```

```
        return 1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(9001);

    // Convert IPv4 address from text to binary form
    result = inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
    if (result <= 0) {
        printf("Invalid address / Address not supported\n");
        closesocket(sock);
        WSACleanup();
        return 1;
    }

    // Connect to server
    result = connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr));
    if (result == SOCKET_ERROR) {
        printf("Connection failed: %d\n", WSAGetLastError());
        closesocket(sock);
        WSACleanup();
        return 1;
    }
    printf("Connected to server!\n");

    // Initialize GMP variables for big integer operations
    mpz_t p, g, private_key, public_key, server_public_key, shared_secret;
    mpz_init(p);
    mpz_init(g);
    mpz_init(private_key);
    mpz_init(public_key);
    mpz_init(server_public_key);
    mpz_init(shared_secret);

    // Get prime numbers from user
```

```
    printf("Enter the two prime numbers (p and g): ");
    fgets(message, BUFFER_SIZE, stdin);
    // Remove newline if present
    message[strcspn(message, "\n")] = 0;

    // Send p and g to server
    send(sock, message, (int)strlen(message), 0);

    // Parse p and g
    char *token = strtok(message, " ");
    mpz_set_str(p, token, 10);
    token = strtok(NULL, " ");
    mpz_set_str(g, token, 10);

    // Generate private key (random number between 1 and 20)
    mpz_set_ui(private_key, rand() % 20 + 1);
    gmp_printf("Private key: %Zd\n", private_key);

    // Compute public key: g^private_key mod p
    mpz_powm(public_key, g, private_key, p);

    // Receive server's public key
    result = recv(sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
        mpz_set_str(server_public_key, buffer, 10);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        closesocket(sock);
        WSACleanup();
        mpz_clear(p);
        mpz_clear(g);
        mpz_clear(private_key);
        mpz_clear(public_key);
        mpz_clear(server_public_key);
        mpz_clear(shared_secret);
```

```
        return 1;
    }

    // Send client's public key
    gmp_printf("CLients public key : %Zd\n", public_key);
    memset(message, 0, BUFFER_SIZE);
    gmp_sprintf(message, "%Zd", public_key);
    send(sock, message, (int)strlen(message), 0);

    // Compute shared secret: (server_public_key)^private_key mod p
    mpz_powm(shared_secret, server_public_key, private_key, p);
    gmp_printf("Shared secret from client: %Zd\n", shared_secret);

    // Clean up
    mpz_clear(p);
    mpz_clear(g);
    mpz_clear(private_key);
    mpz_clear(public_key);
    mpz_clear(server_public_key);
    mpz_clear(shared_secret);

    closesocket(sock);
    WSACleanup();

    return 0;
}
```

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <gmp.h>
```

```c
#pragma comment(lib, "ws2_32.lib")

#define BUFFER_SIZE 1024

int main() {
    WSADATA wsaData;
    SOCKET server_fd = INVALID_SOCKET;
    SOCKET client_sock = INVALID_SOCKET;
    struct sockaddr_in address;
    char buffer[BUFFER_SIZE] = {0};
    char message[BUFFER_SIZE] = {0};
    int addrlen = sizeof(address);
    int result;

    // Initialize Winsock
    result = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        printf("WSAStartup failed: %d\n", result);
        return 1;
    }

    // Initialize random number generator
    srand((unsigned int)time(NULL));

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (server_fd == INVALID_SOCKET) {
        printf("Socket creation failed: %d\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(9000);
```

```
    // Bind socket
    result = bind(server_fd, (struct sockaddr *)&address, sizeof(address));
    if (result == SOCKET_ERROR) {
        printf("Bind failed: %d\n", WSAGetLastError());
        closesocket(server_fd);
        WSACleanup();
        return 1;
    }

    // Listen for connections
    result = listen(server_fd, SOMAXCONN);
    if (result == SOCKET_ERROR) {
        printf("Listen failed: %d\n", WSAGetLastError());
        closesocket(server_fd);
        WSACleanup();
        return 1;
    }

    printf("Server is waiting for client connection...\n");

    // Accept connection
    client_sock = accept(server_fd, (struct sockaddr *)&address, &addrlen);
    if (client_sock == INVALID_SOCKET) {
        printf("Accept failed: %d\n", WSAGetLastError());
        closesocket(server_fd);
        WSACleanup();
        return 1;
    }
    printf("Client connected!\n");

    // Initialize GMP variables
    mpz_t p, g, private_key, public_key, client_public_key, shared_secret;
    mpz_init(p);
    mpz_init(g);
    mpz_init(private_key);
    mpz_init(public_key);
```

```
    mpz_init(client_public_key);
    mpz_init(shared_secret);

    // Read message from client (p and g)
    result = recv(client_sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
        printf("Client says: %s\n", buffer);

        // Parse p and g
        char *token = strtok(buffer, " ");
        mpz_set_str(p, token, 10);
        token = strtok(NULL, " ");
        mpz_set_str(g, token, 10);

        gmp_printf("p = %Zd\n", p);
        gmp_printf("g = %Zd\n", g);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        closesocket(client_sock);
        closesocket(server_fd);
        WSACleanup();
        mpz_clear(p);
        mpz_clear(g);
        mpz_clear(private_key);
        mpz_clear(public_key);
        mpz_clear(client_public_key);
        mpz_clear(shared_secret);
        return 1;
    }

    // Generate private key (random number between 1 and 20)
    mpz_set_ui(private_key, rand() % 20 + 1);
    gmp_printf("Private key: %Zd\n", private_key);

    // Compute public key: g^private_key mod p
```

```
    mpz_powm(public_key, g, private_key, p);

    // Send public key to client
    gmp_printf("Servers public key : %Zd\n", public_key);
    memset(message, 0, BUFFER_SIZE);
    gmp_sprintf(message, "%Zd", public_key);
    send(client_sock, message, (int)strlen(message), 0);

    // Receive client's public key
    memset(buffer, 0, BUFFER_SIZE);
    result = recv(client_sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
        mpz_set_str(client_public_key, buffer, 10);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        closesocket(client_sock);
        closesocket(server_fd);
        WSACleanup();
        mpz_clear(p);
        mpz_clear(g);
        mpz_clear(private_key);
        mpz_clear(public_key);
        mpz_clear(client_public_key);
        mpz_clear(shared_secret);
        return 1;
    }

    // Compute shared secret: (client_public_key)^private_key mod p
    mpz_powm(shared_secret, client_public_key, private_key, p);
    gmp_printf("Shared secret by server: %Zd\n", shared_secret);

    // Clean up
    mpz_clear(p);
    mpz_clear(g);
    mpz_clear(private_key);
```

```
    mpz_clear(public_key);
    mpz_clear(client_public_key);
    mpz_clear(shared_secret);

    closesocket(client_sock);
    closesocket(server_fd);
    WSACleanup();

    return 0;
}
```

Mim.c(Man in middle)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <gmp.h>

#pragma comment(lib, "ws2_32.lib")

#define BUFFER_SIZE 1024

int main() {
    WSADATA wsaData;
    SOCKET mim_server_sock = INVALID_SOCKET;
    SOCKET client_sock = INVALID_SOCKET;
    SOCKET real_server_sock = INVALID_SOCKET;
    struct sockaddr_in mim_server_addr, real_server_addr, client_addr;
    char buffer[BUFFER_SIZE] = {0};
    char message[BUFFER_SIZE] = {0};
    int addrlen = sizeof(client_addr);
    int result;

    // Initialize Winsock
```

```
    result = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        printf("WSAStartup failed: %d\n", result);
        return 1;
    }

    // Initialize random number generator
    srand((unsigned int)time(NULL));

    // Create MiM server socket
    mim_server_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (mim_server_sock == INVALID_SOCKET) {
        printf("MiM server socket creation failed: %d\n",
WSAGetLastError());
        WSACleanup();
        return 1;
    }

    mim_server_addr.sin_family = AF_INET;
    mim_server_addr.sin_addr.s_addr = INADDR_ANY;
    mim_server_addr.sin_port = htons(9001);

    // Bind MiM server socket
    result = bind(mim_server_sock, (struct sockaddr *)&mim_server_addr,
sizeof(mim_server_addr));
    if (result == SOCKET_ERROR) {
        printf("MiM server bind failed: %d\n", WSAGetLastError());
        closesocket(mim_server_sock);
        WSACleanup();
        return 1;
    }

    // Listen for client connections
    result = listen(mim_server_sock, SOMAXCONN);
    if (result == SOCKET_ERROR) {
        printf("MiM server listen failed: %d\n", WSAGetLastError());
```

```
        closesocket(mim_server_sock);
        WSACleanup();
        return 1;
    }

    // Create socket to connect to real server
    real_server_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (real_server_sock == INVALID_SOCKET) {
        printf("Real server socket creation failed: %d\n",
WSAGetLastError());
        closesocket(mim_server_sock);
        WSACleanup();
        return 1;
    }

    real_server_addr.sin_family = AF_INET;
    real_server_addr.sin_port = htons(9000);

    // Convert IPv4 address to binary
    result = inet_pton(AF_INET, "127.0.0.1", &real_server_addr.sin_addr);
    if (result <= 0) {
        printf("Invalid address / Address not supported\n");
        closesocket(mim_server_sock);
        closesocket(real_server_sock);
        WSACleanup();
        return 1;
    }

    // Connect to real server
    result = connect(real_server_sock, (struct sockaddr *)&real_server_addr,
sizeof(real_server_addr));
    if (result == SOCKET_ERROR) {
        printf("Connection to real server failed: %d\n", WSAGetLastError());
        closesocket(mim_server_sock);
        closesocket(real_server_sock);
        WSACleanup();
```

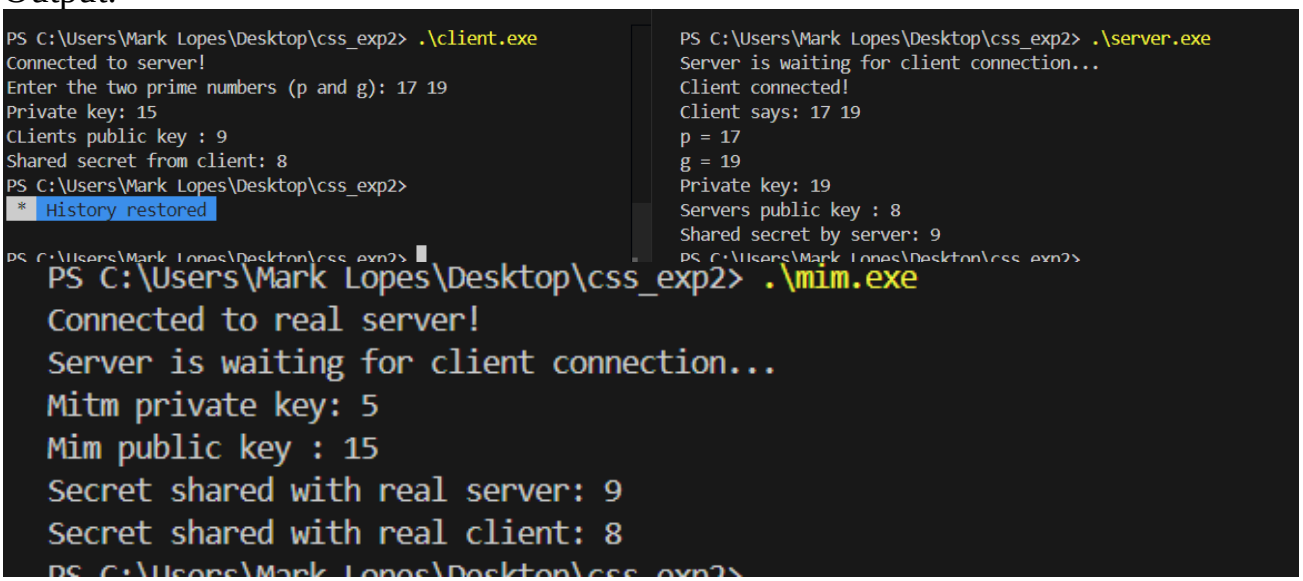
```
        return 1;
    }
    printf("Connected to real server!\n");
    printf("Server is waiting for client connection...\n");

    // Accept connection from real client
    client_sock = accept(mim_server_sock, (struct sockaddr *)&client_addr,
&addrlen);
    if (client_sock == INVALID_SOCKET) {
        printf("Accept failed: %d\n", WSAGetLastError());
        closesocket(mim_server_sock);
        closesocket(real_server_sock);
        WSACleanup();
        return 1;
    }

    // Initialize GMP variables
    mpz_t p, g, mim_private_key, mim_public_key;
    mpz_t real_server_public_key, real_client_public_key;
    mpz_t secret_shared_with_server, secret_shared_with_client;

    mpz_init(p);
    mpz_init(g);
    mpz_init(mim_private_key);
    mpz_init(mim_public_key);
    mpz_init(real_server_public_key);
    mpz_init(real_client_public_key);
    mpz_init(secret_shared_with_server);
    mpz_init(secret_shared_with_client);

    // Main MiM logic
    // Receive p,g from real client
    memset(buffer, 0, BUFFER_SIZE);
    result = recv(client_sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
```

```
        // Forward p,g to real server
        send(real_server_sock, buffer, result, 0);

        // Parse p and g
        char *token = strtok(buffer, " ");
        mpz_set_str(p, token, 10);
        token = strtok(NULL, " ");
        mpz_set_str(g, token, 10);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        goto cleanup;
    }

    // Generate MiM private key
    mpz_set_ui(mim_private_key, rand() % 20 + 1);
    gmp_printf("Mitm private key: %Zd\n", mim_private_key);

    // Compute MiM public key
    mpz_powm(mim_public_key, g, mim_private_key, p);
    gmp_printf("Mim public key : %Zd\n", mim_public_key);

    // Get real server's public key
    memset(buffer, 0, BUFFER_SIZE);
    result = recv(real_server_sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
        mpz_set_str(real_server_public_key, buffer, 10);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        goto cleanup;
    }

    // Send MiM public key to real server
    memset(message, 0, BUFFER_SIZE);
    gmp_sprintf(message, "%Zd", mim_public_key);
```

```
    send(real_server_sock, message, (int)strlen(message), 0);

    // Send MiM public key to real client
    send(client_sock, message, (int)strlen(message), 0);

    // Get real client's public key
    memset(buffer, 0, BUFFER_SIZE);
    result = recv(client_sock, buffer, BUFFER_SIZE, 0);
    if (result > 0) {
        buffer[result] = '\0'; // Ensure null termination
        mpz_set_str(real_client_public_key, buffer, 10);
    } else {
        printf("Receive failed: %d\n", WSAGetLastError());
        goto cleanup;
    }

    // Compute shared secrets
    mpz_powm(secret_shared_with_server, real_server_public_key,
mim_private_key, p);
    mpz_powm(secret_shared_with_client, real_client_public_key,
mim_private_key, p);

    gmp_printf("Secret shared with real server: %Zd\n",
secret_shared_with_server);
    gmp_printf("Secret shared with real client: %Zd\n",
secret_shared_with_client);

cleanup:
    // Clean up
    mpz_clear(p);
    mpz_clear(g);
    mpz_clear(mim_private_key);
    mpz_clear(mim_public_key);
    mpz_clear(real_server_public_key);
    mpz_clear(real_client_public_key);
    mpz_clear(secret_shared_with_server);
```

```
    mpz_clear(secret_shared_with_client);


    closesocket(client_sock);
    closesocket(mim_server_sock);
    closesocket(real_server_sock);
    WSACleanup();


    return 0;
}
```

Output:-



Postlab

TE-COMPS A
9913 - Mark Lopes

26/02/25 CSS - Exp 2 - postlab

Q.1 If Alice and Bob choose the same value for x and y, then the calculated value of A and B will be same.
The session keys will still remain same regardless of A and B being equal

Ex:-
n = 17, g = 3

x = y = 5

Alice finds A:-

$A = g^x \bmod n$
$= 3^5 \bmod 17$
$= 5$

Bob finds B:-

$B = g^y \bmod n$
$= 3^5 \% 17$
$= 5$

Alice computes session key K1 with B from Bob

$K1 = B^x \bmod n$
$= 5^5 \% 17$
$= 13$

Bob computes session key K2 with A from Alice

$K2 = A^x \% n$
$= 5^5 \% 17$
$= 13$

∴ Value of A and B will be same.

Q.2 To protect against man-in-middle attacker, we can use some techniques :-

1. We authenticated Diffie-helman by digital signature

- Each party will sign their public key using private key from a public-key ecosystem (RSA). The other party verifies the signature using sender public key.

- This prevents MIM because attacker cannot forge signatures without access to private keys

2. We Password-Authenticated Key-Exchange (PAKE)

- PAKE protocols (SRP) combine DH with a password.
- Even if an attacker intercepts key exchange, they cannot derive a session key without knowing password.