CRYPTOGRAPHY AND NETWORK SECURITY LAB

DIGITAL ASSIGNMENT -4

NAME-ADRIJA MONDAL

REGISTRATION NUMBER-22BCB0156

Design of a Network Security Models

1. Secure Multiparty Key Exchange and Man-in-the-Middle (MITM) Attack

Simulation Using Diffie-Hellman
A group of three users—Alice, Bob, and Charlie—want to securely establish a shared secret key using a multiparty Diffie-Hellman key exchange protocol over
an insecure channel. However, an attacker Mallory attempts a Man-in-the-Middle (MITM) attack to intercept and manipulate their key exchange process.
Implement a Multiparty Diffie-Hellman Key Exchange:
• Choose a large prime number (p) and a primitive root (g).
• Each participant (Alice, Bob, Charlie) selects their private key and computes their public key.
• They progressively exchange values to compute a shared secret key among all three parties.
Simulate a Man-in-the-Middle Attack (MITM) by Mallory:
• Mallory intercepts and modifies the public keys exchanged between participants.
• Instead of allowing Alice, Bob, and Charlie to compute a common shared key, Mallory establishes separate shared keys with each party.
• Verify that Alice, Bob, and Charlie do not share the same key, but instead, Mallory has established different keys with each of them.

AIM

To implement a **Multiparty Diffie-Hellman Key Exchange** protocol for three users (Alice, Bob, and Charlie) to securely establish a shared secret key over an insecure channel and simulate a **Man-in-the-Middle (MITM) attack** by Mallory to intercept and manipulate the key exchange process.

PROCEDURE

Step 1: Input Selection
The users choose a large prime number (p) and a primitive root (g) as the base of the Diffie-Hellman Key Exchange.

Each participant (Alice, Bob, and Charlie) selects their private key and computes their public key using the formula:

Public Key

=
g
private key
m
o
d

p
Public Key=g
private key
modp
The computed public keys are then exchanged between the participants.

Step 2: Multiparty Diffie-Hellman Key Exchange (Without MITM)
The participants progressively compute a shared secret key using the received public keys. The final shared key for all three parties should be the same.

Step 3: Simulating the MITM Attack
Mallory, the attacker, intercepts the public keys exchanged between Alice, Bob, and Charlie.

Instead of allowing a proper key exchange, Mallory replaces the public keys with fake values and sends different public keys to each participant.

Each participant unknowingly computes a shared secret key with Mallory instead of the actual group.

As a result:

Alice and Mallory share one secret key.

Bob and Mallory share another secret key.

Charlie and Mallory share yet another secret key.

Alice, Bob, and Charlie do not share the same key anymore.

Step 4: Verification of the Attack
The program verifies whether Alice, Bob, and Charlie have different shared keys due to the MITM attack.

The program outputs:

The expected shared key if there was no MITM attack.

The incorrect shared keys that result due to Mallory's attack.

Confirmation that the attack was successful.

Step 5: Program Execution & Testing
The program should be executed multiple times with different values of p, g, and private keys to verify correctness.

Ensure the MITM attack works by intercepting and altering the public keys**.**

CODE
Alice Code

```
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p, g, private_key, public_key;

    cout << "Enter prime number (p): ";
    cin >> p;
    cout << "Enter primitive root (g): ";
    cin >> g;

    cout << "Enter Alice's private key: ";
    cin >> private_key;

    public_key = power(g, private_key, p);

    ofstream file("alice_key.txt");
    file << private_key << " " << public_key;
    file.close();
```

```
    cout << "Alice's public key: " << public_key << endl;
    cout << "Keys saved to 'alice_key.txt'\n";
    return 0;
}
```

Bob Code

```
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p, g, private_key, public_key;

    cout << "Enter prime number (p): ";
    cin >> p;
    cout << "Enter primitive root (g): ";
    cin >> g;

    cout << "Enter Bob's private key: ";
    cin >> private_key;

    public_key = power(g, private_key, p);

    ofstream file("bob_key.txt");
    file << private_key << " " << public_key;
    file.close();

    cout << "Bob's public key: " << public_key << endl;
    cout << "Keys saved to 'bob_key.txt'\n";
    return 0;
}
```

Charlie Code

```
#include <iostream>
```

```
#include <fstream>
#include <cmath>

using namespace std;

long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p, g, private_key, public_key;

    cout << "Enter prime number (p): ";
    cin >> p;
    cout << "Enter primitive root (g): ";
    cin >> g;

    cout << "Enter Charlie's private key: ";
    cin >> private_key;

    public_key = power(g, private_key, p);

    ofstream file("charlie_key.txt");
    file << private_key << " " << public_key;
    file.close();

    cout << "Charlie's public key: " << public_key << endl;
    cout << "Keys saved to 'charlie_key.txt'\n";
    return 0;
}
Compute with Mallory Key
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

long long power(long long base, long long exp, long long mod) {
    long long result = 1;
```

```
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p, g;
    long long alice_priv, alice_pub, bob_priv, bob_pub, charlie_priv,
charlie_pub;
    long long mallory_priv, mallory_pub_a, mallory_pub_b, mallory_pub_c;

    // Read prime and root from Alice (same for all)
    ifstream alice_file("alice_key.txt");
    alice_file >> alice_priv >> alice_pub;
    alice_file.close();

    ifstream bob_file("bob_key.txt");
    bob_file >> bob_priv >> bob_pub;
    bob_file.close();

    ifstream charlie_file("charlie_key.txt");
    charlie_file >> charlie_priv >> charlie_pub;
    charlie_file.close();

    cout << "Enter prime number (p): ";
    cin >> p;
    cout << "Enter primitive root (g): ";
    cin >> g;

    cout << "Enter Mallory's private key: ";
    cin >> mallory_priv;

    // Mallory intercepts and replaces public keys
    mallory_pub_a = power(g, mallory_priv, p);
    mallory_pub_b = power(g, mallory_priv, p);
    mallory_pub_c = power(g, mallory_priv, p);

    cout << "\n[MITM ATTACK SIMULATION]\n";
    cout << "Mallory sends fake public keys instead of real ones:\n";
    cout << "Mallory to Alice: " << mallory_pub_a << "\n";
    cout << "Mallory to Bob: " << mallory_pub_b << "\n";
    cout << "Mallory to Charlie: " << mallory_pub_c << "\n\n";
```

```
    // Each computes a "shared key" but it's with Mallory
    long long shared_mallory_alice = power(alice_pub, mallory_priv, p);
    long long shared_mallory_bob = power(bob_pub, mallory_priv, p);
    long long shared_mallory_charlie = power(charlie_pub, mallory_priv, p);

    cout << "Shared Key (Alice & Mallory): " << shared_mallory_alice << "\n";
    cout << "Shared Key (Bob & Mallory): " << shared_mallory_bob << "\n";
    cout << "Shared Key (Charlie & Mallory): " << shared_mallory_charlie <<
"\n";

    // Verify that Alice, Bob, and Charlie do NOT share the same key
    long long real_shared_key = power(bob_pub, alice_priv, p);
    cout << "\n[Expected Correct Shared Key Without MITM]: " <<
real_shared_key << endl;

    if (shared_mallory_alice != shared_mallory_bob || shared_mallory_bob !=
shared_mallory_charlie) {
        cout << "\n[MITM SUCCESS]: Mallory has established separate keys
with each party!\n";
    } else {
        cout << "\n[MITM FAILED]: All parties share the same key, no
interception occurred.\n";
    }

    return 0;
}
```

```
./compute
Enter prime number (p): 23
Enter primitive root (g): 5
Enter Alice's private key: 6
Alice's public key: 8
Keys saved to 'alice_key.txt'
Enter prime number (p): 23
Enter primitive root (g): 5
Enter Bob's private key: 15
Bob's public key: 19
Keys saved to 'bob_key.txt'
Enter prime number (p): 23
Enter primitive root (g): 5
Enter Charlie's private key: 13
Charlie's public key: 21
Keys saved to 'charlie_key.txt'
Enter prime number (p): 23
Enter primitive root (g): 5
Enter Mallory's private key: 10

[MITM ATTACK SIMULATION]
Mallory sends fake public keys instead of real ones:
Mallory to Alice: 9
Mallory to Bob: 9
Mallory to Charlie: 9
```

Test Case 2

```
adrijam@LAPTOP-7V5JSQPE:~$ ./alice
Enter prime number (p): 37
Enter primitive root (g): 7
Enter Alice's private key: 10
Alice's public key: 7
Keys saved to 'alice_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ ./bob
Enter prime number (p): 37
Enter primitive root (g): 7
Enter Bob's private key: 11
Bob's public key: 12
Keys saved to 'bob_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ ./charlie
Enter prime number (p): 37
Enter primitive root (g): 7
Enter Charlie's private key: 11
Charlie's public key: 12
Keys saved to 'charlie_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ ./compute
Enter prime number (p): 37
Enter primitive root (g): 7
Enter Mallory's private key: 12

[MITM ATTACK SIMULATION]
Mallory sends fake public keys instead of real ones:
Mallory to Alice: 10
```

```
Mallory sends fake public keys instead of real ones:
Mallory to Alice: 10
Mallory to Bob: 10
Mallory to Charlie: 10

Shared Key (Alice & Mallory): 10
Shared Key (Bob & Mallory): 26
Shared Key (Charlie & Mallory): 26

[Expected Correct Shared Key Without MITM]: 12
```

Test Case where Man in Middle attack doesn’t happen

```
adrijam@LAPTOP-7V5JSQPE:~$ ./alice
Enter prime number (p): 31
Enter primitive root (g): 5
Enter Alice's private key: 12
Alice's public key: 1
Keys saved to 'alice_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ ./bob
Enter prime number (p): 31
Enter primitive root (g): 5
Enter Bob's private key: 11
Bob's public key: 25
Keys saved to 'bob_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ 31
31: command not found
adrijam@LAPTOP-7V5JSQPE:~$ ./charlie
Enter prime number (p): 31
Enter primitive root (g): 5
Enter Charlie's private key: 10
Charlie's public key: 5
Keys saved to 'charlie_key.txt'
adrijam@LAPTOP-7V5JSQPE:~$ ./compute
Enter prime number (p): 31
Enter primitive root (g): 5
Enter Mallory's private key: 15

[MITM ATTACK SIMULATION]
```

```
[MITM ATTACK SIMULATION]
Mallory sends fake public keys instead of real ones:
Mallory to Alice: 1
Mallory to Bob: 1
Mallory to Charlie: 1

Shared Key (Alice & Mallory): 1
Shared Key (Bob & Mallory): 1
Shared Key (Charlie & Mallory): 1

[Expected Correct Shared Key Without MITM]: 1

[MITM FAILED]: All parties share the same key, no interception occurred.
```

RESULT: Diffie Hellman and Man in the middle attack implemented successfully.

2. Secure Client-Server Communication Using SSL Sockets

A financial services company wants to securely transmit sensitive customer data between its client application (used by customers) and its server (handling transactions). To achieve this, they must implement a secure client-server communication model using SSL/TLS sockets, ensuring encryption, authentication, and integrity of transmitted data.

Implement a Secure SSL Client-Server Communication Model

• Create a server application that listens for incoming connections using SSL sockets.
• The server must authenticate itself with an SSL/TLS certificate to establish trust.
• The client application should securely connect to the server over SSL and transmit a confidential message (e.g., login credentials or banking details).
• The server should decrypt and respond with an acknowledgment message.

Aim

To implement a **Secure SSL Client-Server Communication Model** using OpenSSL in C++, ensuring **encryption, authentication, and data integrity** while transmitting sensitive financial data over a network.

Algorithm

1. Server Side (ssl_server1.cpp)

1. **Initialize OpenSSL** by loading error strings and algorithms.
2. **Create an SSL Context** using the TLS server method.
3. **Configure SSL Certificates** by loading a **server certificate (server.pem)** and **private key (key.pem)**.
4. **Create a TCP socket** and bind it to **port 12345**.
5. **Listen for incoming connections** and accept a client.
6. **Establish an SSL handshake** with the client.
7. **Receive encrypted data** from the client and decrypt it.
8. **Send an acknowledgment response** back to the client.
9. **Close the connection** and clean up SSL resources.

2. Client Side (ssl_client1.cpp)

1. **Initialize OpenSSL** by loading error strings and algorithms.
2. **Create an SSL Context** using the TLS client method.
3. **Create a TCP socket** and connect to the server at 127.0.0.1:12345.
4. **Wrap the socket with SSL** and initiate an SSL handshake with the server.
5. **Send encrypted data** (e.g., banking credentials) to the server.
6. **Receive and display the server's acknowledgment** message.
7. **Close the connection** and clean up SSL resources.

CODE

Sender Side

```
#include <iostream>

#include <openssl/ssl.h>

#include <openssl/err.h>

#include <sys/socket.h>

#include <arpa/inet.h>
```

```
#include <unistd.h>

#define PORT 12345

void init_openssl() {
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}

SSL_CTX* create_server_context() {
    const SSL_METHOD* method = TLS_server_method();
    SSL_CTX* ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        exit(EXIT_FAILURE);
    }
    return ctx;
}

void configure_server_context(SSL_CTX* ctx) {
    if (SSL_CTX_use_certificate_file(ctx, "server.pem", SSL_FILETYPE_PEM) <=
0 ||
        SSL_CTX_use_PrivateKey_file(ctx, "key.pem", SSL_FILETYPE_PEM) <= 0)
{
        perror("Failed to load SSL certificate or key");
        exit(EXIT_FAILURE);
    }
}

int main() {
    init_openssl();
```

```
SSL_CTX* ctx = create_server_context();
configure_server_context(ctx);

int server_fd = socket(AF_INET, SOCK_STREAM, 0);
sockaddr_in server_addr = {AF_INET, htons(PORT), INADDR_ANY};

bind(server_fd, (sockaddr*)&server_addr, sizeof(server_addr));
listen(server_fd, 1);
std::cout << "[*] SSL Server listening on port " << PORT << "...\n";

int client_fd = accept(server_fd, nullptr, nullptr);
SSL* ssl = SSL_new(ctx);
SSL_set_fd(ssl, client_fd);

if (SSL_accept(ssl) <= 0) {
    perror("SSL handshake failed");
} else {
    std::cout << "[+] Secure connection established\n";
    char buffer[1024] = {0};
    SSL_read(ssl, buffer, sizeof(buffer));
    std::cout << "[*] Received: " << buffer << std::endl;
    SSL_write(ssl, "Server: Data received securely.", 30);
}

SSL_shutdown(ssl);
SSL_free(ssl);
close(client_fd);
close(server_fd);
SSL_CTX_free(ctx);
EVP_cleanup();
```

```
    return 0;
}
```

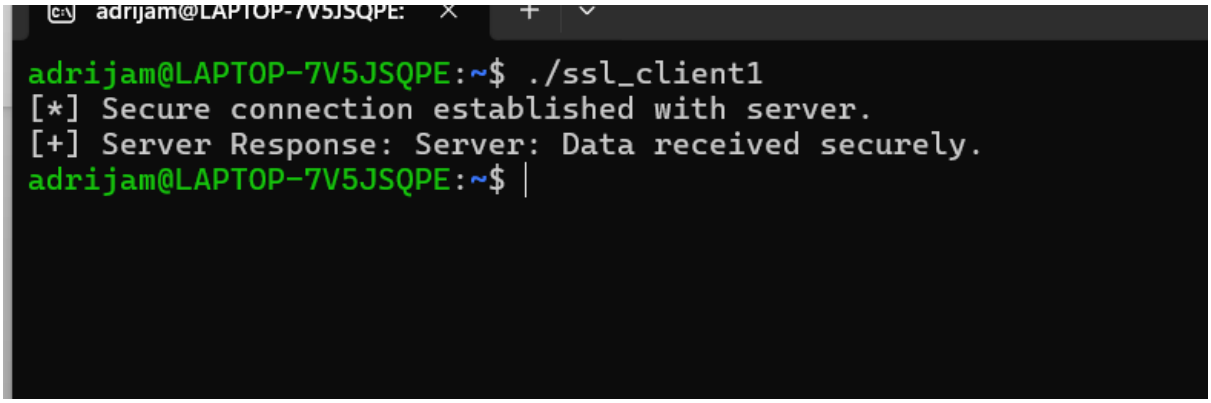
Client Side

```
#include <iostream>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#define SERVER_IP "127.0.0.1"
#define PORT 12345

void init_openssl() {
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}

SSL_CTX* create_client_context() {
    const SSL_METHOD* method = TLS_client_method();
    SSL_CTX* ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        exit(EXIT_FAILURE);
    }
    return ctx;
}

int main() {
    init_openssl();
```

```
    SSL_CTX* ctx = create_client_context();
    SSL* ssl = SSL_new(ctx);

    int client_fd = socket(AF_INET, SOCK_STREAM, 0);
    sockaddr_in server_addr = {AF_INET, htons(PORT)};
    inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr);

    if (connect(client_fd, (sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    SSL_set_fd(ssl, client_fd);
    if (SSL_connect(ssl) <= 0) {
        perror("SSL connection failed");
    } else {
        std::cout << "[*] Secure connection established with server.\n";
        SSL_write(ssl, "Client: Confidential Data - User: Alice, Password:
P@ssw0rd", 60);
        char buffer[1024] = {0};
        SSL_read(ssl, buffer, sizeof(buffer));
        std::cout << "[+] Server Response: " << buffer << std::endl;
    }

    SSL_shutdown(ssl);
    SSL_free(ssl);
    close(client_fd);
    SSL_CTX_free(ctx);
    EVP_cleanup();
    return 0;
}
```

Input/Output

```
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Jharkhand
Locality Name (eg, city) []:Jamshedpur
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TS
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:ADRIJA
Email Address []:adrijam482@gmail.com
adrijam@LAPTOP-7V5JSQPE:~$ vi ssl_server.cpp
adrijam@LAPTOP-7V5JSQPE:~$ vi ssl_server1.cpp
adrijam@LAPTOP-7V5JSQPE:~$ vi ssl_client1.cpp
adrijam@LAPTOP-7V5JSQPE:~$ g++ ssl_server1.cpp -o ssl_server1 -lssl -lcrypto
g++ ssl_client1.cpp -o ssl_client1 -lssl -lcrypto
adrijam@LAPTOP-7V5JSQPE:~$ ./ssl_server1
[*] SSL Server listening on port 12345...
[+] Secure connection established
[*] Received: Client: Confidential Data - User: Alice, Password: P@ssw0rd
```



Result: Secure Client Server communication using Sockets implemented successfully.