BCSE309P - Cryptography and Network Security Laboratory

Name: Priyanshu Dayaramani Reg No.: 21BCE5259

Date: -28-02-2024

Implementation of Diffie-Hellman Key Exchange Protocol

Aim: Design a Diffie Hellman multiparty key exchange protocol and perform a Man-in-the-Middle Attack.

Algorithm:

A malicious Malory, that has a MitM (man in the middle) position, can manipulate the communications between Alice and Bob, and break the security of the key exchange.

Step by Step explanation of this process:

Step 1: Selected public numbers p and g, p is a prime number, called the “modulus” and g is called the base.

Step 2: Selecting private numbers.

let Alice pick a private random number a and let Bob pick a private random number b, Malory picks 2 random numbers c and d.

Step 3: Intercepting public values,

Malory intercepts Alice’s public value (ga(mod p)), block it from reaching Bob, and instead sends Bob her own public value (gc(modp)) and Malory intercepts Bob’s public value (gb(mod p)), block it from reaching Alice, and instead sends Alice her own public value (gd (modp))

Step 4: Computing secret key

Alice will compute a key S1=gda(mod p), and Bob will compute a different key, S2=gcb(mod p)

Step 5: If Alice uses S1 as a key to encrypt a later message to Bob, Malory can decrypt it, re-encrypt it using S2, and send it to Bob. Bob and Alice won’t notice any problem and may assume their communication is encrypted, but in reality, Malory can decrypt, read, modify, and then re-encrypt all their conversations.

Code 1: (for deffie hellman key exchange)

Server:

Output:

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

long power(long a, long b, long p) {
    long result = 1;
    a = a % p; // Take modulo of base if it is more than or equal to p

    while (b > 0) {
        if (b % 2 == 1)
            result = (result * a) % p;

        b = b / 2;
        a = (a * a) % p;
    }

    return result;
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    long P = 23; // Example prime number
    long G = 9; // Example primitive root
    long b, y, x, ka;
```

```
// Create socket
server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == -1) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}

// Server address setup
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(12345); // Port number 12345

// Bind socket
if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
    perror("Binding failed");
    exit(EXIT_FAILURE);
}

// Listen for connections
if (listen(server_socket, 5) == -1) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

printf("Server is listening...\n");

// Accept connection
socklen_t client_addr_len = sizeof(client_addr);
client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len);
```

```
if (client_socket == -1) {
    perror("Accept failed");
    exit(EXIT_FAILURE);
}

// Send prime number P and primitive root G to client
send(client_socket, &P, sizeof(P), 0);
send(client_socket, &G, sizeof(G), 0);

// Receive x from client
recv(client_socket, &x, sizeof(x), 0);

printf("Received x: %ld\n", x);

printf("Enter the private key of Bob: ");
scanf("%ld", &b);

y = power(G, b, P);

// Send y to client
send(client_socket, &y, sizeof(y), 0);

// Calculate shared secret key
ka = power(x, b, P);

printf("The secret key of Alice is: %ld\n", ka);

// Close sockets
close(client_socket);
close(server_socket);
```

```
    return 0;
}
```

Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

long power(long a, long b, long p) {
    long result = 1;
    a = a % p; // Take modulo of base if it is more than or equal to p

    while (b > 0) {
        if (b % 2 == 1)
            result = (result * a) % p;

        b = b / 2;
        a = (a * a) % p;
    }

    return result;
}

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    long P, G, x, a, y, b, kb;

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
```

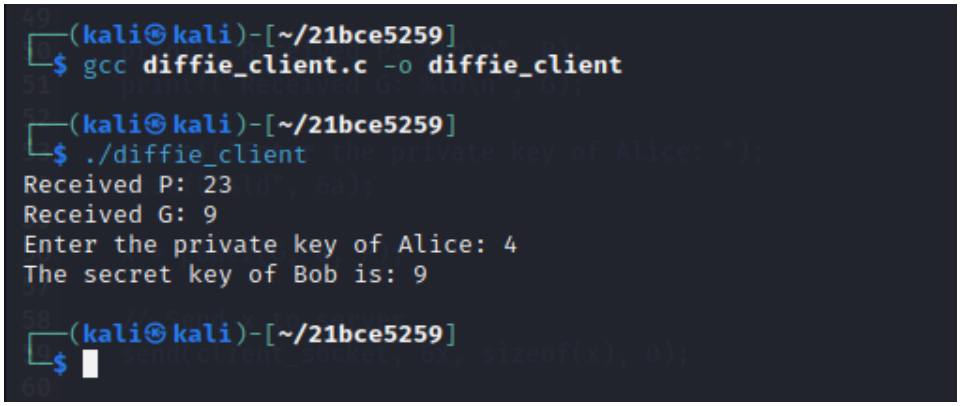
```
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Server address setup
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(12345); // Assuming server is running on port 12345
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr); // Assuming server is running on
localhost

    // Connect to server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Receive prime number P and primitive root G from server
    recv(client_socket, &P, sizeof(P), 0);
    recv(client_socket, &G, sizeof(G), 0);

    printf("Received P: %ld\n", P);
    printf("Received G: %ld\n", G);

    printf("Enter the private key of Alice: ");
    scanf("%ld", &a);

    x = power(G, a, P);

    // Send x to server
```

```
    send(client_socket, &x, sizeof(x), 0);

    // Receive y from server
    recv(client_socket, &y, sizeof(y), 0);

    // Calculate shared secret key
    kb = power(y, a, P);

    printf("The secret key of Bob is: %ld\n", kb);

    // Close socket
    close(client_socket);

    return 0;
}
```

Output:

Server:

Client:


┌──(kali㉿kali)-[~/21bce5259]
└─$ gcc diffie_client.c -o diffie_client
┌──(kali㉿kali)-[~/21bce5259]
└─$ ./diffie_client
Received P: 23
Received G: 9
Enter the private key of Alice: 4
The secret key of Bob is: 9
┌──(kali㉿kali)-[~/21bce5259]
└─$

Code 2: (for man in middle attack)

Server:

```
#include <iostream>
#include <string>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

using namespace std;

// Function to calculate modular exponentiation
unsigned long long modPow(unsigned long long base, unsigned long long exp, unsigned long long mod) {
    unsigned long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp & 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}

int main() {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
```

```
int addrlen = sizeof(address);

unsigned long long p, g, a, A;

cout << "Enter value of p: ";
cin >> p;
cout << "Enter value of g: ";
cin >> g;
cout << "Enter value of a (Alice's private key): ";
cin >> a;

// Alice generates her public key
A = modPow(g, a, p);

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
}

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(8080);

// Binding socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
```

```
    perror("bind failed");
    exit(EXIT_FAILURE);
}

// Listening for connections
if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
}

cout << "Waiting for client connection..." << endl;

if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
    perror("accept");
    exit(EXIT_FAILURE);
}

cout << "Client connected" << endl;

// Sending parameters to the client
send(new_socket, &p, sizeof(p), 0);
send(new_socket, &g, sizeof(g), 0);
send(new_socket, &A, sizeof(A), 0);

// Closing server socket
close(server_fd);

// Alice's secret key
unsigned long long keyA = modPow(A, a, p);
cout << "Darth's secret key with Alice:" << keyA << endl;
```

```
    return 0;
}
```

Client:

```
#include <iostream>
#include <string>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

using namespace std;

// Function to calculate modular exponentiation
unsigned long long modPow(unsigned long long base, unsigned long long exp, unsigned long long mod) {
    unsigned long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp & 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
```

```
unsigned long long p, g, A;

// Creating socket file descriptor
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    cout << "Socket creation error" << endl;
    return -1;
}

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(8080);

// Convert IPv4 and IPv6 addresses from text to binary form
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
    cout << "Invalid address/ Address not supported" << endl;
    return -1;
}

// Connecting to server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    cout << "Connection Failed" << endl;
    return -1;
}

// Receiving parameters from server
recv(sock, &p, sizeof(p), 0);
recv(sock, &g, sizeof(g), 0);
recv(sock, &A, sizeof(A), 0);

unsigned long long b, keyB;

cout << "Enter value of b (Bob's private key): ";
```

```
    cin >> b;

    // Calculating Bob's secret key
    keyB = modPow(A, b, p);
    cout << "Darth's secret key with Bob:" << keyB << endl;

    // Closing client socket
    close(sock);

    return 0;
}
```

Output:

Server:

Client: