Họ và tên sinh viên: Huỳnh Lê Minh - 22162026,

Chống Lệ Vân – 22162055

1. **Phân công nhiệm vụ từng thành viên:**

- Làm client.c: Lê Minh
- Làm server.c : Lệ Vân

2. **Hướng dẫn setup môi trường thực thi**

- Cài đặt MSYS2:
  - Tải từ trang chủ MSYS2.
  - Cập nhật packages: pacman -Syu.
  - Cài đặt compiler và libssh: pacman -S mingw-w64-x86_64-gcc mingw-w64-x86_64-libssh.
- Tạo SSH Key Pair
  - ssh-keygen -t rsa -b 4096 -f ./id_rsa
- Copy public key (id_rsa.pub) vào Server (C:\Van\Lap_Trinh_Mang\kt\.ssh)

3. **Trình bày giải pháp thực hiện theo tiếp cận Top-down refinement.**

Mô tả giải pháp

Chúng ta sẽ chia quá trình gửi file có xác thực thành các bước lớn và tiếp tục phân rã chúng thành các bước chi tiết hơn.

Bước 1: Xác định chức năng chính của chương trình

Chương trình có nhiệm vụ:

Xác thực client bằng RSA (Digital Signature).

Kết nối TCP đến server.

Gửi file đến server sau khi xác thực thành công.

Phân rã bài toán → Chia chương trình thành 3 thành phần chính:

Thiết lập kết nối với server

Xác thực client bằng chữ ký số (RSA-SHA256)

Gửi file qua mạng TCP

Bước 2: Thiết lập kết nối TCP

Chức năng:

Mở socket TCP.

Kết nối đến server.

Kiểm tra kết nối có thành công không.

Phân rã chi tiết

Khởi tạo Winsock (trên Windows).

Tạo socket (SOCKET sock).

Thiết lập địa chỉ IP và cổng (sockaddr_in server).

Gọi connect() để kết nối server.

Nhận phản hồi xác nhận kết nối từ server.

Bước 3: Xác thực client bằng chữ ký số

Chức năng:

Nhận challenge từ server.

Ký challenge bằng RSA-SHA256.

Gửi chữ ký số đến server.

Nhận phản hồi xác thực từ server.

Phân rã chi tiết:

Nhận challenge (recv(sock, challenge, ...)).

Đọc khóa riêng id_rsa.pem.

Ký challenge (EVP_DigestSignFinal()).

Gửi chữ ký số đến server.

Nhận phản hồi từ server (recv(sock, response, 4, 0)).

Bước 4: Gửi file đến server

Chức năng:

Kiểm tra file có tồn tại không.

Gửi tên file đích và kích thước file.

Đọc file theo từng phần nhỏ và gửi qua socket.

Hiển thị tiến trình gửi file.

Phân rã chi tiết:

Mở file nguồn (fopen()).

Xác định kích thước file (fseek()).

Gửi metadata: đường dẫn file đích + kích thước file.

Gửi nội dung file theo từng khối nhỏ (send()).

Tính toán và hiển thị tiến trình (% hoàn thành).

Bước 5: Dọn dẹp và kết thúc

Chức năng:

Giải phóng bộ nhớ.

Đóng file.

Đóng socket.

Phân rã chi tiết:

Đóng file (fclose()).

Giải phóng chữ ký (free(sig)).

Giải phóng OpenSSL (EVP_MD_CTX_free()).

Đóng socket (closesocket()).

4. **Mô tả các bước thực hiện code.**

Server:

- Nhận file khoá công khai của client (authorized_keys.pem)
- Viết chương trình lắng nghe kết nối của client
- Viết challenge và challenge cho client
- Dùng khoá công khai của client để giải mã và xác thực challenge
- Nếu xác thực thành công challenge, thiết lập session key mã hoá
- Nhận file truyền từ client

Client :

- Xác thực bằng chữ ký số RSA sử dụng OpenSSL.

- Sau khi xác thực thành công, chương trình gửi file từ máy client đến server.

5. **Giải thích các phần code quan trọng.**

Client:

1. Đọc khóa riêng :

- Mở file chứa **khóa riêng** (id_rsa.pem).
- Đọc dữ liệu và trích xuất khóa **EVP_PKEY** từ định dạng **PEM**.
- Nếu mở file thất bại, báo lỗi và trả về NULL.

```
EVP_PKEY* read_private_key(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        perror("Error opening private key file");
        return NULL;
    }
    EVP_PKEY* pkey = PEM_read_PrivateKey(file, NULL, NULL, NULL);
    fclose(file);
    return pkey;
}
```

2.Khởi tạo kết nối TCP

- Khởi tạo **Winsock** trên Windows (WSAStartup).
- Tạo **socket TCP** (socket(AF_INET, SOCK_STREAM, 0)).
- Thiết lập **địa chỉ IP và cổng**.
- Gọi connect() để kết nối đến server.

```
WSADATA wsa;
SOCKET sock;
struct sockaddr_in server;

WSAStartup(MAKEWORD(2, 2), &wsa);  // Khởi tạo Winsock (Windows)
sock = socket(AF_INET, SOCK_STREAM, 0);  // Tạo socket TCP
server.sin_addr.s_addr = inet_addr(SERVER_IP);  // Địa chỉ server
server.sin_family = AF_INET;
server.sin_port = htons(PORT);

connect(sock, (struct sockaddr*)&server, sizeof(server)); // Kết nối server
printf("Connected to server %s:%d\n", SERVER_IP, PORT);
```

3. Nhận challenge từ server

- Nhận một **chuỗi thử thách 32 byte** từ server.
- Challenge này sẽ được ký bằng **RSA-SHA256** để xác thực client.

```
unsigned char challenge[32];
recv(sock, (char*)challenge, sizeof(challenge), 0);
```

4. Ký thử thách bằng RSA-SHA256

- Đọc khóa riêng id_rsa.pem.
- Tạo **EVP_MD_CTX** để khởi tạo hàm băm SHA-256.
- **EVP_DigestSignUpdate()**: Băm dữ liệu **challenge**.
- **EVP_DigestSignFinal()**: Ký dữ liệu bằng **RSA** và lấy chữ ký số (sig).

```
EVP_PKEY* private_key = read_private_key("id_rsa.pem");
if (!private_key) {
    closesocket(sock);
    WSACleanup();
    return 1;
}

EVP_MD_CTX* md_ctx = EVP_MD_CTX_new();
EVP_PKEY_CTX* ctx;
EVP_DigestSignInit(md_ctx, &ctx, EVP_sha256(), NULL, private_key);
EVP_PKEY_CTX_set_rsa_padding(ctx, RSA_PKCS1_PADDING);

size_t sig_len;
EVP_DigestSignUpdate(md_ctx, challenge, sizeof(challenge));
EVP_DigestSignFinal(md_ctx, NULL, &sig_len);

unsigned char* sig = (unsigned char*)malloc(sig_len);
EVP_DigestSignFinal(md_ctx, sig, &sig_len);
```

5. Gửi chữ ký số đến server

- Chuyển đổi kích thước chữ ký thành **big-endian** (htonl).
- Gửi **kích thước chữ ký** trước.
- Gửi **chữ ký số** đến server.

```
uint32_t sig_len_net = htonl((uint32_t)sig_len);
send(sock, (char*)&sig_len_net, sizeof(sig_len_net), 0);
send(sock, (char*)sig, sig_len, 0);
```

6. Nhận phản hồi tới server

- Nhận phản hồi từ server.
- Nếu **"OK"**, xác thực thành công → tiếp tục gửi file.
- Nếu không, chương trình thoát.

```
char response[5] = {0};
recv(sock, response, 4, 0);

if (strcmp(response, "OK") == 0) {
    // Tiếp tục gửi file
}
```

7.Gửi thông tin file

- Mở file nguồn **(source file)**.
- **Lấy kích thước file** bằng ftell().
- Gửi thông tin:

1. **Tên file đích** (dest_path).
2. **Kích thước file** (file_size).

```
FILE* file = fopen(source_path, "rb");
if (!file) {
    perror("Error opening file to send");
    closesocket(sock);
    return 1;
}

fseek(file, 0, SEEK_END);
uint64_t file_size = ftell(file);
rewind(file);

send(sock, dest_path, 256, 0); // Gửi đường dẫn file đích
send(sock, (char*)&file_size, sizeof(file_size), 0); // Gửi kích thước file
```

8.Gửi dữ liệu file

- Đọc file thành từng khối **(BUFFER_SIZE = 1024 bytes)**.
- Gửi từng phần bằng **send()**.
- Hiển thị tiến trình truyền file **(Progress %)**.
- Tính thời gian truyền file.

```c
char buffer[BUFFER_SIZE];
size_t bytes_read;
uint64_t sent = 0;
time_t start_time = time(NULL);

while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
    send(sock, buffer, bytes_read, 0);
    sent += bytes_read;
    printf("\rProgress: %.2f%%", (double)sent / file_size * 100);
}

printf("\nFile sent successfully! Time elapsed: %ld seconds\n", time(NULL) - start_ti
```

9.Dọn dẹp và kết thúc

- **Giải phóng bộ nhớ** (free()).
- **Đóng file** (fclose()).
- **Giải phóng OpenSSL context** (EVP_MD_CTX_free()).
- **Đóng socket** (closesocket()).
- **Dọn dẹp Winsock** (WSACleanup()).

```c
fclose(file);
free(sig);
EVP_MD_CTX_free(md_ctx);
EVP_PKEY_free(private_key);
closesocket(sock);
WSACleanup();
return 0;
```

Server:

- Nhận file khoá công khai của client (authorized_keys.pem)

```
EVP_PKEY* read_public_key(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        perror("Error opening public key file");
        return NULL;
    }
    EVP_PKEY* pkey = PEM_read_PUBKEY(file, NULL, NULL, NULL);
    fclose(file);
    return pkey;
}
```

- Hàm read_public_key() mở file chứa khóa công khai của client (authorized_keys.pem), đọc nội dung và chuyển thành đối tượng EVP_PKEY* để sử dụng trong xác thực.
- Nếu mở file thất bại, chương trình in lỗi và trả về NULL.

- Viết chương trình lắng nghe kết nối của client

```
WSADATA wsa;
SOCKET server_fd, new_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE] = {0};

WSAStartup(MAKEWORD(2, 2), &wsa);
server_fd = socket(AF_INET, SOCK_STREAM, 0);
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

bind(server_fd, (struct sockaddr*)&address, sizeof(address));
listen(server_fd, 3);

printf("Server is listening on port %d...\n", PORT);
```

- Server khởi tạo Winsock (WSAStartup()), tạo socket, và bind socket tới cổng 8080.
- Sau đó, gọi listen() để chờ kết nối từ client.

- Viết challenge và challenge cho client

```
new_socket = accept(server_fd, (struct sockaddr*)&address, &addrlen);
printf("\nClient connected from %s:%d\n", inet_ntoa(address.sin_addr), ntohs(address.sin_port));

// Gửi challenge cho client
unsigned char challenge[32];
RAND_bytes(challenge, sizeof(challenge));
send(new_socket, (char*)challenge, sizeof(challenge), 0);
```

  - Khi client kết nối, server tạo một challenge ngẫu nhiên (RAND_bytes()).
  - Challenge này giúp xác thực client bằng khóa công khai.
- Dùng khoá công khai của client để giải mã và xác thực challenge

```
// Nhận chữ ký từ client
uint32_t sig_len_net;
recv(new_socket, (char*)&sig_len_net, sizeof(sig_len_net), 0);
unsigned int sig_len = ntohl(sig_len_net);

unsigned char* sig = (unsigned char*)malloc(sig_len);
recv(new_socket, (char*)sig, sig_len, 0);

// Đọc khóa công khai để xác thực
EVP_PKEY* public_key = read_public_key("authorized_keys.pem");
if (!public_key) {
    closesocket(new_socket);
    free(sig);
    continue;
}

EVP_MD_CTX* md_ctx = EVP_MD_CTX_new();
EVP_PKEY_CTX* ctx;
EVP_DigestVerifyInit(md_ctx, &ctx, EVP_sha256(), NULL, public_key);
EVP_PKEY_CTX_set_rsa_padding(ctx, RSA_PKCS1_PADDING);
```

```
int verify_result = EVP_DigestVerify(md_ctx, sig, sig_len, challenge, sizeof(challenge));

free(sig);
EVP_MD_CTX_free(md_ctx);
EVP_PKEY_free(public_key);

if (verify_result != 1) {
    send(new_socket, "FAIL", 4, 0);
    closesocket(new_socket);
    continue;
}

send(new_socket, "OK", 2, 0);
```

  - Nhận chữ ký số từ client (sig) và kiểm tra chữ ký này có hợp lệ không.
  - Nếu xác thực thất bại (verify_result != 1), server từ chối kết nối.
- Nếu xác thực thành công challenge, thiết lập session key mã hoá

- Nhận file truyền từ client

```
// Nhận thông tin file từ client
char file_path[256];
uint64_t file_size;
recv(new_socket, file_path, sizeof(file_path), 0);
recv(new_socket, (char*)&file_size, sizeof(file_size), 0);

printf("Receiving file: %s (Size: %llu bytes)\n", file_path, file_size);

// Mở file để ghi dữ liệu
FILE* file = fopen(file_path, "wb");
if (!file) {
    perror("Error opening file for writing");
    closesocket(new_socket);
    continue;
}

// Nhận và ghi dữ liệu vào file
uint64_t received = 0;
int bytes;
time_t start_time = time(NULL);
while ((bytes = recv(new_socket, buffer, BUFFER_SIZE, 0)) > 0) {
    fwrite(buffer, 1, bytes, file);
    received += bytes;
    printf("\rProgress: %.2f%%", (double)received / file_size * 100);
}

fclose(file);  // Đóng file sau khi ghi xong

printf("\nFile saved to: %s\n", file_path);
printf("Time elapsed: %ld seconds\n", time(NULL) - start_time);

closesocket(new_socket);
```

  - Server nhận file_path và file_size từ client.
  - Mở file để ghi (fopen(file_path, "wb")).
  - Nhận dữ liệu từ client và ghi vào file.

6. Kết quả biên dịch và chạy chương trình.

Server

Biên dịch: gcc -o server server.c -lssl -lcrypto -lws2_32

Thực hiện chương trình: ./server

```
dinhk@Van MSYS /c/Van/Lap_Trinh_Mang/kt
# ./server
Server is listening on port 8080...

Client connected from 10.107.2.235:63624
Receiving file: C:\Van\Lap_Trinh_Mang\kt\dest_dir (Size: 20 bytes)
Error opening file for writing: Permission denied
```

Client:

Biên dịch: gcc -o scp_client scp_client.c -lssl -lcrypto -lws2_32

Thực hiện chương trình: ./scp_client "D:\ssh\textssh\text.txt" "C:\Van\Desktop"

```
hlm10@ElmMinh UCRT64 /d/ssh
# ./scp_client "D:\ssh\textssh\text.txt" "C:\Van\Desktop"
Connected to server 10.107.2.208:8080
Progress: 100.00%
File sent successfully! Time elapsed: 0 seconds

hlm10@ElmMinh UCRT64 /d/ssh
#
```