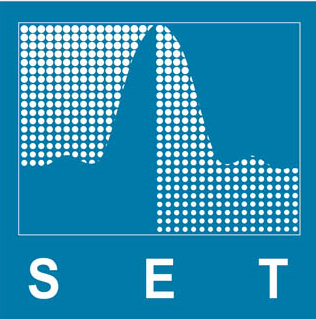
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN ĐIỆN TỬ - VIỄN THÔNG**

--🙢🕮🙠--



**BÁO CÁO BÀI TẬP LỚN  
MÔN: LÝ THUYẾT MẬT MÃ**

***Đề tài:***

***Lập trình một ứng dụng chat bảo mật sử dụng SSL/TLS hoặc RSA để mã hóa tin nhắn giữa hai client.***

GVHD: Hà Duyên Trung   
Nhóm bao gồm

1. Phạm Văn Hoàn 20213929  
2. Đào Đức Hiệp 20213911

*Hà Nội, 2025*

# Mục lục

[Mục lục 2](#_Toc200562263)

[LỜI MỞ ĐẦU 4](#_Toc200562264)

[CHƯƠNG 1. TỔNG QUAN VỀ CÁC GIAO THỨC MẠNG VÀ BẢO MẬT 5](#_Toc200562265)

[1.1. Mạng Client-Server 5](#_Toc200562266)

[1.1.1. Khái niệm 5](#_Toc200562267)

[1.1.2. Cách thức hoạt động 5](#_Toc200562268)

[1.1.3. Ưu điểm và nhược điểm của mô hình Client-Server 5](#_Toc200562269)

[1.2. Các nguy cơ bảo mật khi truyền thông 6](#_Toc200562270)

[1.3. Giới thiệu mã hóa 7](#_Toc200562271)

[CHƯƠNG 2: GIAO THỨC SSL/TLS VÀ ỨNG DỤNG 10](#_Toc200562272)

[2.1 Tổng quan về SSL/TLS 10](#_Toc200562273)

[2.2 Handshake và Trao đổi Khóa trong SSL/TLS 10](#_Toc200562274)

[2.2.1 Mô phỏng một quá trình Handshake cơ bản (ví dụ cho Server Authentication) 11](#_Toc200562275)

[2.2.2 Mutual TLS (mTLS) - Xác thực hai chiều 12](#_Toc200562276)

[2.3. Ứng dụng Chứng chỉ X.509 và Khóa RSA trong phần mềm chat 12](#_Toc200562277)

[2.4. So sánh Mã hóa Kết nối (SSL/TLS) và Mã hóa End-to-End 13](#_Toc200562278)

[2.4.1. Mã hóa Kết nối (SSL/TLS - Hop-by-Hop Encryption) 13](#_Toc200562279)

[2.4.2. Mã hóa End-to-End (E2EE) 13](#_Toc200562280)

[CHƯƠNG 3. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG CHAT CLIENT-SERVER VỚI SSL/TLS 15](#_Toc200562281)

[3.1 Cấu trúc và thành phần hệ thống 15](#_Toc200562282)

[3.2 Quy trình hoạt động 18](#_Toc200562283)

[3.3 Giao diện người dùng 21](#_Toc200562284)

[CHƯƠNG 4: ĐÁNH GIÁ VÀ KẾT QUẢ 24](#_Toc200562285)

[4.1. Kiểm thử bảo mật 24](#_Toc200562286)

[4.1.1. Tình huống 1: sai chứng chỉ client 24](#_Toc200562287)

[4.1.2. Tình huống 2: Tấn công Man-in-the-Middle (MITM) 24](#_Toc200562288)

[4.1.3. Tình huống 3: Bắt gói bằng Wireshark 25](#_Toc200562289)

[4.2. Đánh giá 26](#_Toc200562290)

[KẾT LUẬN 28](#_Toc200562291)

[PHỤ LỤC 28](#_Toc200562292)

**LỜI MỞ ĐẦU**

Ngày nay, với sự phát triển nhanh chóng của công nghệ thông tin, đa số các hoạt động đều thực hiện qua mạng internet thì vấn đề bảo mật và an toàn thông tin ngày càng trở nên quan trọng hơn bao giờ hết. Đặc biệt, trong bối cảnh giao tiếp trực tuyến trở thành một phần thiết yếu của cuộc sống và công việc, nhu cầu về một kênh truyền thông an toàn, bảo mật và đáng tin cậy là không thể thiếu.

Trong lĩnh vực an toàn thông tin, các giao thức mật mã đóng vai trò nền tảng để bảo vệ dữ liệu khỏi các mối đe dọa như nghe lén, giả mạo hay thay đổi. Trong số đó, giao thức SSL/TLS (Secure Sockets Layer/Transport Layer Security) là một chuẩn mực vàng cho việc thiết lập các kênh truyền thông an toàn trên Internet. SSL/TLS không chỉ cung cấp khả năng mã hóa dữ liệu để đảm bảo tính bảo mật mà còn thực hiện xác thực các bên tham gia, bảo vệ tính toàn vẹn của thông tin truyền tải. Mặc dù có nhiều giao thức bảo mật khác, nhưng SSL/TLS vẫn duy trì được vị thế quan trọng nhờ khả năng ứng dụng rộng rãi và độ tin cậy cao.

Chính vì những lý do trên mà nhóm chúng em đã lựa chọn nghiên cứu đề tài: “Ứng dụng mã hóa SSL/TLS cho hệ thống Chat Client-Server”. Nội dung đề tài bao gồm:

Chương 1: Tổng quan về các giao thức mạng và bảo mật   
Chương 2: Giao thức SSL/TLS và ứng dụng   
Chương 3: Thiết kế và triển khai hệ thống Chat Client-Server với SSL/TLS   
Chương 4: Đánh giá và kết quả

Chúng em mong nhận được sự góp ý, chỉ bảo của thầy để đề tài của chúng em được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

# CHƯƠNG 1. TỔNG QUAN VỀ CÁC GIAO THỨC MẠNG VÀ BẢO MẬT

* 1. Mạng Client-Server
     1. Khái niệm

Mô hình Client-Server là một kiến trúc ứng dụng phân tán, trong đó một máy chủ (Server) cung cấp tài nguyên, dữ liệu hoặc dịch vụ cho các máy khách (Client) thông qua mạng máy tính. Client và Server hoạt động như hai thực thể riêng biệt: Client khởi tạo các yêu cầu dịch vụ, còn Server lắng nghe các yêu cầu đó, xử lý chúng và gửi lại phản hồi.

* + 1. Cách thức hoạt động

Trong mô hình Client-Server, quá trình truyền thông diễn ra theo các bước cơ bản như sau:

* Bước 1: Client gửi yêu cầu: Khi một Client cần một dịch vụ (ví dụ: yêu cầu kết nối, gửi tin nhắn, lấy danh sách người dùng…), nó sẽ tạo một yêu cầu và gửi đến Server. Yêu cầu này thường bao gồm thông tin về dịch vụ mong muốn và dữ liệu liên quan.
* Bước 2: Server xử lý yêu cầu: Server sẽ “lắng nghe” các kết nối đến từ Client trên một cổng (còn gọi là Port) cụ thể. Khi nhận được yêu cầu, Server sẽ phân tích, xử lý logic (ví dụ: tạo kết nối, lưu tin nhắn, cập nhật trạng thái người dùng) và chuẩn bị phản hồi.
* Bước 3: Server phản hồi dữ liệu: Sau khi xử lý, Server gửi dữ liệu hoặc thông báo kết quả trở lại Client. Client nhận phản hồi và xử lý tiếp/phản hồi cho người dùng.
  + 1. Ưu điểm và nhược điểm của mô hình Client-Server
* Ưu điểm:

+ Tập trung hóa: mô hình sẽ mang tính tập trung hóa, dữ liệu và tài nguyên được quản lý tập trung trên Server, giúp dễ dàng kiểm soát, sao lưu và bảo mật.

+ Mở rộng: mô hình sẽ dễ dàng mở rộng bằng cách thêm nhiều Client mới mà không ảnh hưởng đến Server, hoặc nâng cấp Server để phục vụ nhiều Client hơn.

+ Dễ dàng bảo trì: mô hình sẽ dễ dàng bảo trì hoặc cập nhật chỉ bằng những tao tác thực hiện trên Server.

+ Chia sẻ tài nguyên: với mô hình này, Nhiều Client có thể cùng lúc truy cập và chia sẻ tài nguyên trên Server.

* Nhược điểm:

+ Điểm lỗi duy nhất/Single Point of Failure: Chỉ cần server gặp sự cố, toàn bộ hệ thống sẽ bị gián đoạn.

+ Tắc nghẽn mạng: khi có lượng truy cập đồng thời lớn, tắc nghẽn Server hoặc mạng có thể xảy ra.

+ Chi phí: Yêu cầu phần cứng và phần mềm Server chuyên dụng, có thể tốn kém, hoặc với những dự án nhỏ sẽ phải cài đặt một vài ứng dụng cụ thể gây rắc rối.

+ Bảo mật: Server và những kết nối đến Server là mục tiêu chính của các cuộc tấn công, yêu cầu các biện pháp bảo mật mạnh mẽ.

* 1. Các nguy cơ bảo mật khi truyền thông

Trong môi trường mạng Client-Server, dữ liệu được trao đổi liên tục giữa các bên. Nếu không có biện pháp bảo vệ phù hợp, thông tin có thể đối mặt với nhiều nguy cơ bảo mật nghiêm trọng.

* Nguy cơ bị nghe lén (Evaesdropping): là hành vi thu thập trái phép thông tin đang được truyền tải qua mạng. Kẻ tấn công có thể "nghe" hoặc "đánh hơi" (sniffing) các gói tin đi qua mạng mà không cần truy cập trực tiếp vào máy Client hay Server. Nếu dữ liệu không được mã hóa, kẻ tấn công có thể đọc được nội dung tin nhắn, thông tin đăng nhập, hoặc bất kỳ dữ liệu nhạy cảm nào khác.
* Nguy cơ bị giả mạo danh tính (Spoofing): Giả mạo danh tính là hành vi kẻ tấn công đóng giả làm một người dùng hoặc một hệ thống hợp pháp để lừa đảo các bên khác hoặc để giành quyền truy cập trái phép. Kẻ giả mạo có thể gửi tin nhắn dưới danh nghĩa là người khác, lừa đảo người dùng hoặc hệ thống, hoặc thực hiện các hành động độc hại.
* Nguy cơ bị tấn công trung gian (Man-in-the-Middle - MitM): Tấn công MitM xảy ra khi kẻ tấn công “chèn mình” vào giữa hai bên đang giao tiếp, đánh chặn và có khả năng đọc và thay đổi các thông tin được trao đổi mà không bị phát hiện. Kẻ tấn công có thể giả mạo cả hai phía (Client và Server) để lừa dối họ tin rằng họ đang giao tiếp trực tiếp với nhau.
* Các nguy cơ khác: tấn công từ chối dịch vụ (DoS/DDoS), tấn công tiêm mã (Injection Attacks), tấn công Relay…
  1. Giới thiệu mã hóa

Để đối phó với các nguy cơ bảo mật nêu trên, mã hóa là một trong những công cụ rất quan trọng. Mã hóa là quá trình biến đổi thông tin (văn bản gốc/Plaintext) thành một dạng không thể đọc được (văn bản mã hóa/Ciphertext) mà không có khóa giải mã phù hợp.

Có thể kể đến hai loại mã hóa sau:

* Mã hóa đối xứng (Symmetric Encryption):

+ Khái niệm: Mã hóa đối xứng là phương pháp mã hóa mà trong đó, cùng một khóa bí mật (secret key) được sử dụng cho cả quá trình mã hóa và giải mã dữ liệu. Cả người gửi và người nhận đều phải có khóa này.

+ Thuật toán mã hóa: AES (Advanced Encryption Standard), DES (Data Encryption Standard), Triple DES.

+ Ưu/Nhược điểm: Tốc độ mã hóa và giải mã rất nhanh, phù hợp cho việc mã hóa một lượng lớn dữ liệu. Nhưng vấn đề chính là làm thế nào để trao đổi khóa bí mật một cách an toàn giữa các bên mà không bị lộ. Nếu khóa bị lộ, tính bảo mật của toàn bộ dữ liệu mã hóa sẽ bị phá vỡ.

+ Ứng dụng: Mã hóa dữ liệu lưu trữ trên đĩa cứng, mã hóa luồng dữ liệu lớn trong giao tiếp (sau khi đã thiết lập khóa an toàn).

* Mã hóa bất đối xứng (Asymmetric Encryption / Public-key Cryptography)

+ Khái niệm: Mã hóa bất đối xứng sử dụng một cặp khóa: một khóa công khai (public key) và một khóa riêng (private key). Khóa công khai có thể được chia sẻ rộng rãi, trong khi khóa riêng phải được giữ bí mật tuyệt đối. Nếu dữ liệu được mã hóa bằng khóa công khai của người nhận, chỉ khóa riêng tương ứng của người đó mới có thể giải mã. Điều này đảm bảo tính bảo mật; Nếu dữ liệu được ký bằng khóa riêng của người gửi, bất kỳ ai có khóa công khai của người gửi đều có thể xác minh được chữ ký, đảm bảo tính xác thực và chống chối bỏ.

+ Thuật toán mã hóa: RSA (Rivest-Shamir-Adleman), ECC (Elliptic Curve Cryptography), Diffie-Hellman (cho trao đổi khóa).

+ Ưu/Nhược điểm: mã hóa bất đối xứng có ưu điểm là giải quyết bài toán trao đổi khóa an toàn của mã hóa đối xứng. Khóa công khai có thể được phân phối công khai mà không làm giảm bảo mật; Nhưng nhược điểm là tốc độ mã hóa và giải mã chậm hơn đáng kể so với mã hóa đối xứng, không phù hợp để mã hóa lượng lớn dữ liệu.

+ Ứng dụng: Trao đổi khóa bí mật cho các thuật toán mã hóa đối xứng; Ký số (Digital Signatures) để xác thực danh tính và đảm bảo tính toàn vẹn; Mã hóa một lượng nhỏ dữ liệu nhạy cảm (ví dụ: thông tin của phiên làm việc).

* Bảng so sánh hai loại mã hóa:

| **Tiêu chí** | **Mã hóa đối xứng** | **Mã hóa bất đối xứng** |
| --- | --- | --- |
| **Số lượng khóa** | 1 khóa (khóa bí mật) | 2 khóa (khóa công khai và khóa riêng) |
| **Tốc độ** | Nhanh | Chậm |
| **Sử dụng** | Mã hóa dữ liệu lớn | Trao đổi khóa, ký số, xác thực, mã hóa dữ liệu nhỏ |
| **Vấn đề chính** | Trao đổi khóa an toàn | Hiệu suất |
| **Khả năng bảo vệ** | Bảo mật (Confidentiality) | Bảo mật, Xác thực (Authentication), Chống chối bỏ (Non-repudiation) |

# CHƯƠNG 2: GIAO THỨC SSL/TLS VÀ ỨNG DỤNG

## 2.1 Tổng quan về SSL/TLS

SSL (Secure Sockets Layer) và TLS (Transport Layer Security) là các giao thức mật mã được thiết kế để cung cấp bảo mật truyền thông qua mạng máy tính. Về bản chất, TLS là phiên bản kế nhiệm và được phát triển từ SSL, nhưng thuật ngữ SSL vẫn thường được sử dụng phổ biến. Các giao thức này hoạt động ở tầng Transport của mô hình TCP/IP (hoặc tầng Session/Presentation của mô hình OSI), nằm giữa tầng Ứng dụng (Application Layer) và tầng Giao vận (Transport Layer). Vai trò chính của SSL/TLS là thiết lập một kênh truyền thông an toàn, bảo vệ dữ liệu khỏi các mối đe dọa như nghe lén, giả mạo, và thay đổi dữ liệu trong quá trình truyền tải. Khi một kết nối SSL/TLS được thiết lập, mọi dữ liệu trao đổi giữa Client và Server sẽ được mã hóa, đảm bảo tính bảo mật, toàn vẹn và xác thực danh tính của các bên.

Các tính năng chính của SSL/TLS như sau:

- Mã hóa dữ liệu (Confidentiality): Toàn bộ dữ liệu truyền qua lại giữa Client và Server được mã hóa, ngăn chặn việc nghe lén và đọc trộm thông tin. Đây là chức năng cốt lõi, bảo vệ nội dung các tin nhắn chat khỏi kẻ tấn công.

- Xác thực danh tính (Authentication): SSL/TLS cho phép Client xác minh danh tính của Server thông qua chứng chỉ số (X.509 certificate). Điều này giúp Client đảm bảo rằng nó đang kết nối với Server hợp pháp, tránh các cuộc tấn công giả mạo Server. Trong các trường hợp nâng cao như hệ thống của chúng em, SSL/TLS còn cho phép Server xác thực danh tính của Client (Mutual TLS - mTLS).

- Bảo toàn vẹn dữ liệu (Integrity): các giao thức sử dụng các hàm băm mật mã (ví dụ: SHA-256) và mã xác thực thông điệp (MAC) để đảm bảo rằng dữ liệu không bị sửa đổi trong quá trình truyền tải. Nếu bất kỳ bit nào của dữ liệu bị thay đổi, điều đó sẽ được phát hiện và kết nối có thể bị chấm dứt.

## 2.2 Handshake và Trao đổi Khóa trong SSL/TLS

Quá trình Handshake (bắt tay) là giai đoạn quan trọng nhất của SSL/TLS, nơi Client và Server thương lượng các tham số bảo mật và thiết lập một khóa phiên bí mật để mã hóa dữ liệu.

### 2.2.1 Mô phỏng một quá trình Handshake cơ bản (ví dụ cho Server Authentication)

Ảnh có chứa văn bản, ảnh chụp màn hình

Nội dung do AI tạo ra có thể không chính xác.

Quá trình diễn ra theo các bước như trên hình :

- Bước 1: Client Hello**:** Client gửi phiên bản TLS cao nhất được hỗ trợ, số ngẫu nhiên, và danh sách cipher suites (danh sách các bộ mã hóa).

- Bước 2: Server Hello**:** Server chọn phiên bản TLS, cipher suite, gửi số ngẫu nhiên và chứng chỉ số (X.509).

- Bước 3: Xác thực Server**:** Client kiểm tra tính hợp lệ của chứng chỉ (CA, tên miền, thời hạn). Nếu chứng chỉ không hợp lệ, kết nối sẽ bị hủy.

- Bước 4: Trao đổi khóa**:** Client dùng khóa công khai từ chứng chỉ để mã hóa Pre-Master Secret và gửi cho Server.

- Bước 5: Sinh khóa phiên**:** Cả hai bên tạo Master Secret từ Pre-Master + số ngẫu nhiên, rồi sinh Session Keys đối xứng.

- Bước 6: Kết thúc Handshake**:** Hai bên xác nhận sử dụng khóa mới → thiết lập kênh bảo mật.

### 2.2.2 Mutual TLS (mTLS) - Xác thực hai chiều

Trong hệ thống chat, chúng em triển khai Mutual TLS (mTLS), một dạng nâng cao của Handshake, nơi cả Server và Client đều xác thực lẫn nhau. Sau bước Server gửi chứng chỉ của mình, Server sẽ yêu cầu Client gửi chứng chỉ của nó.

Server sẽ yêu cầu Client Certificate bằng cách gửi thông điệp "Certificate Request" đến Client. Sau đó, Client gửi chứng chỉ X.509 của mình cho Server. Server nhận chứng chỉ của Client và xác minh tính hợp lệ của nó, tương tự như cách Client xác thực Server. mTLS cung cấp mức độ bảo mật cao hơn vì nó đảm bảo cả hai phía của kết nối đều là các nguồn đáng tin cậy.

## 2.3. Ứng dụng Chứng chỉ X.509 và Khóa RSA trong phần mềm chat

Dự án chat Client-Server của chúng em tận dụng tối đa các thành phần của SSL/TLS, đặc biệt là chứng chỉ X.509 và các cặp khóa RSA, để thiết lập một kênh truyền thông bảo mật hai chiều (Mutual TLS). Đây là bảng mô tả chi tiết các thành phần chính và vai trò của chúng trong hệ thống:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thành phần** | File | Loại | Vai trò | Liên kết với code |
| Server Certificate | Cert.pem | X.509 Certificate | Chứa khóa công khai và thông tin định danh của Server. Gửi cho Client để xác minh danh tính Server trong quá trình TLS Handshake. | **Server:** context.load\_cert\_chain(certfile="cert.pem", keyfile="key.pem") **Client:** context.load\_verify\_locations(cafile="../Server/cert.pem") |
| Server Private Key | Key.pem | RSA Private Key | |  | | --- | |  |  |  | | --- | | Giữ bí mật. Dùng để: - Giải mã Pre-Master Secret từ Client - Ký số các thông điệp trong Handshake  nhằm xác thực Server | | **Server:** Tải cùng với cert.pem qua context.load\_cert\_chain(...) |
| Client Certificate | client\_cert.pem | X.509 Certificate | Chứa khóa công khai của Client. Gửi cho Server (trong Mutual TLS) để xác minh danh tính Client, đảm bảo chỉ Client hợp lệ được truy cập Server. | **Server:** context.load\_verify\_locations(cafile="../Client/client\_cert.pem") **Client:** context.load\_cert\_chain(certfile="client\_cert.pem", keyfile="client\_key.pem") |
| Client Private Key | client\_key.pem | RSA private key | Giữ bí mật. Dùng để: - Ký số các thông điệp khi Server yêu cầu xác thực - (Tùy chọn) Giải mã dữ liệu được Server mã hóa bằng khóa công khai của Client | **Client:** Tải cùng với client\_cert.pem qua context.load\_cert\_chain(...) |

## 2.4. So sánh Mã hóa Kết nối (SSL/TLS) và Mã hóa End-to-End

Trong lĩnh vực bảo mật truyền thông, có hai khái niệm quan trọng là mã hóa kết nối (Hop-by-Hop Encryption) và mã hóa End-to-End (E2EE). Trong ứng dụng của chúng em, chúng em áp dụng mã hóa kết nối, nhưng vẫn có sự đảm bảo nhất định về thông tin được chuyển đi, sẽ được đề cấp đến ở những chương sau

### 2.4.1. Mã hóa Kết nối (SSL/TLS - Hop-by-Hop Encryption)

- Lớp mã hóa: Mã hóa kết nối hoạt động ở tầng Transport, như SSL/TLS. Khi dữ liệu được mã hóa bằng SSL/TLS, nó sẽ được mã hóa trên đường truyền giữa hai điểm cuối trực tiếp kết nối với nhau (ví dụ: Client và Server, hoặc Server và một Proxy).

- Phương pháp mã hóa: Sau quá trình Handshake, SSL/TLS sử dụng các thuật toán mã hóa đối xứng mạnh mẽ (ví dụ: AES-256-GCM trong TLS 1.3 - đây là bộ mã hóa phổ biến và mạnh mẽ nhất hiện nay và cũng là bộ mã hóa được sử dụng trong ứng dụng) để mã hóa dữ liệu thực tế.

- Ai sẽ đọc được dữ liệu: Dữ liệu được mã hóa trên đường truyền, nhưng khi đến điểm cuối (ví dụ: Server), nó sẽ được giải mã. Điều này có nghĩa là Server có thể đọc được nội dung tin nhắn. Tương tự, nếu có một Proxy trung gian được cấu hình để "giải mã và kiểm tra", Proxy đó cũng có thể đọc được dữ liệu.

### 2.4.2. Mã hóa End-to-End (E2EE)

- Lớp mã hóa: Mã hóa End-to-End hoạt động ở tầng Ứng dụng (Application Layer). Dữ liệu được mã hóa tại nguồn gửi và chỉ được giải mã tại đích nhận cuối cùng.

- Phương pháp mã hóa: Thường sử dụng kết hợp cả mã hóa đối xứng (để mã hóa nội dung) và mã hóa bất đối xứng (để trao đổi khóa đối xứng). Ví dụ: mỗi người dùng có một cặp khóa riêng, và khóa phiên được tạo ra chỉ giữa hai người dùng trực tiếp.

- Ai sẽ đọc được dữ liệu: Chỉ người gửi và người nhận cuối cùng mới có thể đọc được dữ liệu. Các Server trung gian chỉ thấy dữ liệu đã mã hóa và không thể giải mã nội dung. Điều này có nghĩa là ngay cả Server cũng không thể đọc được tin nhắn riêng tư.

- Ứng dụng hiện tại của chúng em chưa triển khai mã hóa End-to-End. Server đóng vai trò trung tâm trong việc chuyển tiếp và quản lý tin nhắn, do đó Server cần có khả năng đọc và xử lý nội dung tin nhắn (ví dụ: để xác định người nhận tin nhắn riêng tư, lưu trữ lịch sử). Nhưng với mã hóa kết nối của chúng em, chỉ những Server và Client được “tin cậy” mới tạo được kết nối với nhau, và tin nhắn chuyển tiếp giữa client-server-client được bọc bằng JSON và đã được mã hóa nên MitM không thể đọc được.

# CHƯƠNG 3. THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG CHAT CLIENT-SERVER VỚI SSL/TLS

Chương này sẽ trình bày chi tiết về cấu trúc, các thành phần chính và quy trình hoạt động của hệ thống chat được phát triển, tập trung vào việc áp dụng giao thức SSL/TLS để đảm bảo an toàn và bảo mật cho việc kết nối và dữ liệu truyền tải. Đồng thời, chương này cũng mô tả giao diện người dùng được xây dựng dễ dàng bằng PyQt5.

Để có thể kết nối được các client ở MỌI NƠI mà chỉ cần có internet, và cũng bởi vì đây là một dự án có quy mô không quá lớn, nên chúng em đã sử dụng dụng Tailscale (địa chỉ: [Tailscale · Best VPN Service for Secure Networks](https://tailscale.com/)) với mục đích: tạo vpn để có thể có ip public của một thiết bị làm Server; tạo ra mạng TailNet để các client có thể nhắn tin với nhau giống như nhắn tin với nhau qua internet.

## 3.1 Cấu trúc và thành phần hệ thống

Hệ thống chat được thiết kế theo mô hình Client-Server, nơi một server trung tâm chịu trách nhiệm quản lý kết nối, xác thực người dùng và chuyển tiếp tin nhắn giữa các client. Các client tương tác với server thông qua kết nối được mã hóa bằng SSL/TLS và cung cấp giao diện người dùng để người dùng có thể gửi và nhận tin nhắn.

Hệ thống sẽ bao gồm những thành phần chính như sau:

* **server.py**

Đây là trái tim của hệ thống chat, đóng vai trò là máy chủ trung tâm. Nhiệm vụ chính là chấp nhận và quản lý các kết nối client, xử lý luồng giao tiếp, chuyển tiếp tin nhắn giữa các người dùng, và đặc biệt là thực hiện quá trình xác thực SSL/TLS với mỗi client. File này chứa logic cho việc khởi tạo socket, bọc nó trong SSL/TLS, lắng nghe kết nối, và tạo ra các luồng xử lý riêng biệt cho mỗi client. Nó cũng quản lý danh sách người dùng online và lịch sử tin nhắn.

- Xác thực TLS: Server được cấu hình để yêu cầu và xác minh chứng chỉ của client (Mutual TLS) thông qua ***context.verify\_mode = ssl.CERT\_REQUIRED*** và ***context.load\_verify\_locations(cafile="../Client/client\_cert.pem")***. Điều này đảm bảo rằng chỉ những client có chứng chỉ hợp lệ và được server tin cậy mới có thể kết nối.

- Quản lý Client: Mỗi client kết nối sẽ được xử lý bởi một luồng riêng biệt ***threading.Thread(target=handle\_client, args=(connstream, addr))***, cho phép server xử lý nhiều kết nối đồng thời mà không bị chặn.

* **client\_CORE.py**

Module này chứa các chức năng cốt lõi cho phía client, bao gồm việc thiết lập kết nối SSL/TLS với server, gửi và nhận tin nhắn được mã hóa, và quản lý trạng thái kết nối. Nó hoạt động như một lớp trừu tượng, tách biệt logic mạng khỏi giao diện người dùng.

- Kết nối SSL: Client sử dụng ***ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH)*** và ***context.load\_verify\_locations(cafile="server\_cert.pem")*** để xác minh chứng chỉ của server, đảm bảo rằng client đang kết nối đến server hợp lệ.

- Xác thực TLS (Client-side): Client cũng tải chứng chỉ và khóa riêng của mình (***context.load\_cert\_chain(certfile="client\_cert.pem", keyfile="client\_key.pem")***) để server có thể xác thực danh tính của client, hoàn thành Mutual TLS.

- Gửi/Nhận dữ liệu JSON: Phương thức ***send\_json*** và logic nhận dữ liệu từ server đảm bảo rằng mọi thông điệp được “đóng gói” và “giải mã” dưới dạng “JSON”, tạo điều kiện thuận lợi cho việc trao đổi dữ liệu có cấu trúc.

* **client\_GUI\_qt.py** :

File này chịu trách nhiệm xây dựng giao diện người dùng đồ họa (GUI) cho client sử dụng thư viện PyQt5. Giao diện này cung cấp các thành phần tương tác cho phép người dùng nhập tin nhắn, xem lịch sử chat và theo dõi danh sách người dùng online.

- Tương tác với client\_CORE: GUI tương tác với SSLClient (từ **client\_CORE.py**) để gửi tin nhắn, nhận cập nhật trạng thái kết nối và danh sách người dùng.

- Cập nhật GUI an toàn: Sử dụng ***pyqtSignal*** và ***SignalDispatcher*** để cho phép các luồng nền (nhận tin nhắn) cập nhật giao diện người dùng một cách an toàn và đồng bộ, tránh các vấn đề về đa luồng trong GUI.

- Giao diện thân thiện: Được thiết kế với phong cách giao diện “giống” Messenger, cung cấp trải nghiệm chat trực quan và dễ sử dụng.

* **cert.pem, key.pem**

Đây là cặp chứng chỉ số X.509 và khóa riêng RSA của server.

- cert.pem: Chứa chứng chỉ công khai của server, được sử dụng bởi client để xác minh danh tính của server trong quá trình bắt tay SSL/TLS.

- **key.pem**: Chứa khóa riêng của server, được sử dụng để giải mã dữ liệu được mã hóa bằng khóa công khai tương ứng và để ký các thông điệp trong quá trình bắt tay TLS.

* **client\_cert.pem, client\_key.pem**

Đây là cặp chứng chỉ số X.509 và khóa riêng RSA của client.

- client\_cert.pem: Chứa chứng chỉ công khai của client, được gửi đến server để server xác minh danh tính của client (trong Mutual TLS).

- client\_key.pem: Chứa khóa riêng của client, được sử dụng để giải mã dữ liệu được mã hóa bởi server (nếu server mã hóa bằng khóa công khai của client) và để ký các thông điệp.

## 3.2 Quy trình hoạt động

Quy trình hoạt động của hệ thống chat tập trung vào việc thiết lập và duy trì các kết nối SSL/TLS an toàn, đảm bảo rằng mọi giao tiếp giữa client và server đều được mã hóa và xác thực.

Trước khi tiến đến các bước thiết lập kết nối, server sẽ được khởi động và sẽ vào chế độ “lắng nghe” xem có client nào yêu cầu kết nối hay không

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, màu đen

Nội dung do AI tạo ra có thể không chính xác.

0.0.0.0:2021 : điều này thể hiện Server “lắng nghe” trên tất cả các interface, trong trường hợp ứng dụng của chúng em là port 2021 (là port chúng em tự tạo, và nó có thể ra được mạng nhờ Tailscale)

* **Bước 1: Client khởi tạo kết nối SSL với server.**

Khi ứng dụng client được khởi động, nó sẽ cố gắng thiết lập một kết nối TCP đến địa chỉ IP và cổng của server. Sau đó, client sẽ bọc kết nối TCP này bằng một lớp SSL/TLS.

- Client tạo một ***ssl.SSLContext*** với mục đích xác thực server (***ssl.Purpose.SERVER\_AUTH***).

- Client tải chứng chỉ của server (***server\_cert.pem***) vào context để có thể xác minh server.

- Client tải chứng chỉ và khóa riêng của chính nó (***client\_cert.pem, client\_key.pem***) vào context để chuẩn bị cho quá trình xác thực lẫn nhau (Mutual TLS) với server.

- Client sử dụng ***context.wrap\_socket() để tạo ssl\_socket*** từ socket TCP thông thường và cố gắng kết nối đến server.

* **Bước 2: Server xác thực chứng chỉ client.**

Khi server nhận được yêu cầu kết nối từ client, quá trình bắt tay SSL/TLS (SSL/TLS Handshake) sẽ diễn ra. Trong đó, server sẽ yêu cầu và xác minh chứng chỉ của client.

- Server tạo một ***ssl.SSLContext*** với mục đích xác thực client (***ssl.Purpose.CLIENT\_AUTH***).

- Server tải chứng chỉ và khóa riêng của chính nó (***cert.pem, key.pem***).

- Quan trọng nhất, server tải chứng chỉ của client (***client\_cert.pem***) vào ***context.load\_verify\_locations*** và đặt ***context.verify\_mode = ssl.CERT\_REQUIRED***. Điều này buộc server phải yêu cầu chứng chỉ từ client và chỉ chấp nhận kết nối nếu chứng chỉ của client hợp lệ và được server tin cậy.

- Nếu quá trình xác thực thành công, kết nối SSL/TLS an toàn giữa client và server sẽ được thiết lập. Nếu thất bại, server sẽ từ chối kết nối hoặc phát sinh lỗi SSL.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

* **Bước 3: Sau handshake, client gửi/nhận tin nhắn được mã hóa qua SSL.**

Khi kênh liên lạc bảo mật đã được thiết lập, mọi dữ liệu được gửi giữa client và server sẽ tự động được mã hóa bởi lớp SSL/TLS.

- Client sử dụng phương thức ***send\_json(data)*** để gửi các thông điệp dưới dạng JSON. Dữ liệu JSON được chuyển đổi thành bytes và gửi qua ***self.ssl\_socket.sendall()***.

- Server và client liên tục lắng nghe dữ liệu đến thông qua ***ssl\_socket.recv()***. Dữ liệu nhận được sẽ được giải mã tự động bởi lớp SSL/TLS và được xử lý dưới dạng JSON.

- Các tin nhắn được gửi đi bao gồm thông tin về loại tin nhắn (type), người gửi (sender), nội dung (content) và dấu thời gian (timestamp).

* **Bước 4: Server chuyển tiếp tin nhắn tới các client khác.**

Khi server nhận được một tin nhắn từ một client, nó sẽ xử lý tin nhắn đó và chuyển tiếp đến các client liên quan hoặc toàn bộ các client đang online tùy thuộc vào loại tin nhắn (chat công khai hay riêng tư).

- Server giải mã tin nhắn JSON nhận được.

- Đối với tin nhắn công khai (***"type": "chat\_message"***), server sẽ duyệt qua danh sách ***online\_users*** và gửi tin nhắn đến tất cả các client đang kết nối (Bằng hình thức broadcast).

- Đối với tin nhắn riêng tư (nếu được triển khai), server sẽ tìm kiếm client đích trong ***online\_users*** và chỉ gửi tin nhắn đến client đó.

- Server cũng thêm tin nhắn vào ***message\_history*** để duy trì lịch sử chat và gửi lịch sử này đến client khi client yêu cầu hoặc khi client mới kết nối.

Ví dụ hai client “Hoàn”và “Hiệp” đã đăng nhập và nhắn tin cho nhau, đây là những gì sẽ xuất hiện trên Server

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

## 3.3 Giao diện người dùng

Giao diện người dùng của hệ thống chat được xây dựng bằng thư viện PyQt5, cung cấp một trải nghiệm tương tác trực quan và thân thiện, mô phỏng phong cách của các ứng dụng nhắn tin hiện đại giống như Messenger.

Các thành phần chính của giao diện gồm:

* Popup nhập tên người dùng: Khi ứng dụng khởi động lần đầu hoặc khi cần, một hộp thoại ***QInputDialog*** sẽ xuất hiện yêu cầu người dùng nhập tên của mình.
* Thông báo hệ thống: Các tin nhắn từ hệ thống (ví dụ: thông báo kết nối, ngắt kết nối của người dùng khác) cũng được hiển thị trong khu vực chat, thường với định dạng khác để phân biệt.
* Khu vực hiển thị tin nhắn: Đây là khu vực chính hiển thị tất cả các tin nhắn đã gửi và nhận. Các tin nhắn được định dạng với tên người gửi, nội dung tin nhắn và dấu thời gian thực.
* Hộp nhập tin nhắn : Một ô văn bản cho phép người dùng nhập nội dung tin nhắn. Người dùng có thể nhấn Enter để gửi tin nhắn hoặc nhấn nút "Gửi".
* Nút "Gửi": Kích hoạt chức năng gửi tin nhắn từ hộp nhập tin nhắn đến server.
* Danh sách người dùng online: Một danh sách hiển thị tên của tất cả các người dùng đang kết nối và hoạt động trên server. Danh sách này được cập nhật động khi có người dùng kết nối hoặc ngắt kết nối.
* Trạng thái kết nối: Một nhãn hoặc khu vực hiển thị trạng thái hiện tại của kết nối với server (ví dụ: "Đã kết nối", "Đang ngắt kết nối", "Lỗi kết nối").

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.

Giao diện được thiết kế để tự động cuộn xuống cuối cuộc trò chuyện khi có tin nhắn mới, đảm bảo người dùng luôn nhìn thấy những thông tin mới nhất. Các tín hiệu (signals) và khe (slots) của PyQt5 được sử dụng rộng rãi để đảm bảo cập nhật giao diện người dùng mượt mà và an toàn từ các luồng khác nhau.

# CHƯƠNG 4: ĐÁNH GIÁ VÀ KẾT QUẢ

Chương này trình bày các quy trình kiểm thử và đánh giá hiệu quả bảo mật của hệ thống chat đã phát triển. Chúng em tập trung vào việc xác minh tính toàn vẹn, bảo mật và khả năng hoạt động của hệ thống trong các tình huống thực tế.

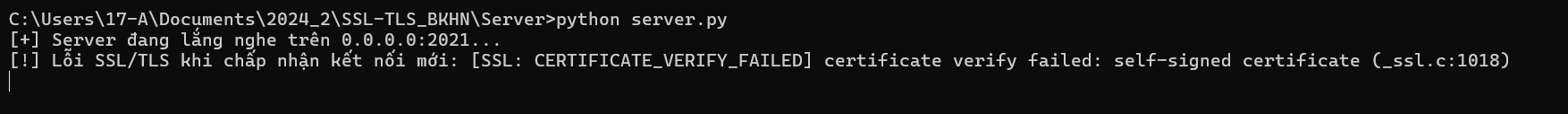
## 4.1. Kiểm thử bảo mật

Để đánh giá hiệu quả bảo mật của hệ thống, chúng em đã tiến hành các tình huống kiểm thử cụ thể nhằm mô phỏng các cuộc tấn công phổ biến và kiểm tra khả năng phòng chống của hệ thống.

* + 1. Tình huống 1: sai chứng chỉ client

Client cố gắng kết nối với server bằng một chứng chỉ không hợp lệ hoặc không được server tin cậy (***fake\_client\_cert.pem, fake\_client\_key.pem***).

Kết quả: Kết nối bị từ chối. Server từ chối SSL/TLS handshake do xác minh chứng chỉ client thất bại.



Đánh giá: Hệ thống ngăn chặn thành công các kết nối trái phép dựa trên chứng chỉ.

* + 1. Tình huống 2: Tấn công Man-in-the-Middle (MITM)

Chúng em giả định khi không dùng SSL/TLS để mã hóa, attacker có thể nghe lén và sửa đổi dữ liệu. Chúng em đã tạo một phiên bản ***server\_no\_ssl.py*** và ***client\_no\_ssl.py*** dùng socket thô không có SSL, và đã tạo ***mitm\_proxy.p***y để mô phỏng attacker đứng giữa. Sau đó, chúng em cho chạy 3 file .py mới này thì kết quả thu được như sau (chúng em mô phỏng trong local host):

Server:

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình

Nội dung do AI tạo ra có thể không chính xác.

Client:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

Attacker đứng giữa:

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình, màu đen

Nội dung do AI tạo ra có thể không chính xác.

Kết quả: MitM có thể đọc được thông tin gửi đi 🡺 không an toàn.

Đánh giá: Mutual TLS (mTLS) đảm bảo tính xác thực lẫn nhau, ngăn chặn MITM.

* + 1. Tình huống 3: Bắt gói bằng Wireshark

Chúng em sử dụng công cụ phân tích gói tin mạng (Wireshark) để bắt và phân tích dữ liệu truyền qua lại giữa client và server.

Khi hai client nhắn tin với nhau như sau:

Ảnh có chứa văn bản, phần mềm, Biểu tượng máy tính, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

Sử dụng Wireshark, chúng em đã bắt được gói va hiển thị như sau:

Ảnh có chứa ảnh chụp màn hình, văn bản, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

* Cho thấy rằng nội dung “Đại học Bách Khoa Hà Nội” đã được mã hóa rất khó đọc !

## 4.2. Đánh giá

Dựa trên các kết quả kiểm thử, hệ thống chat được chúng em đánh giá như sau:

* Hoạt động ổn định**:** Hệ thống duy trì kết nối liên tục và xử lý các tin nhắn một cách đáng tin cậy. Các luồng xử lý riêng biệt cho mỗi client (như trong ***server.py***) giúp đảm bảo tính ổn định và khả năng mở rộng.
* Kết nối được bảo mật**:** Việc triển khai SSL/TLS với Mutual TLS (mTLS) đảm bảo rằng cả client và server đều xác thực lẫn nhau trước khi thiết lập kênh liên lạc an toàn. Điều này ngăn chặn các cuộc tấn công giả mạo (spoofing) và đảm bảo rằng chỉ các bên được ủy quyền mới có thể tham gia vào cuộc trò chuyện.
* Chứng chỉ số và SSL**:** Việc sử dụng ***cert.pem***, ***key.pem*** cho server và ***client\_cert.pem*** để xác minh client (như đã cấu hình trong ***get\_ssl\_context*** của ***server.py***) cùng với việc client tin tưởng ***server\_cert.pem*** (như trong ***get\_context*** của ***client\_CORE.py***) là nền tảng vững chắc cho việc xác thực. Điều này giúp ngăn chặn các cuộc tấn công MITM, đảm bảo tính bảo mật và toàn vẹn của dữ liệu truyền đi.
* Bảo vệ dữ liệu hiệu quả**:** Qua kiểm thử với Wireshark, chúng tôi xác nhận rằng nội dung tin nhắn được mã hóa hoàn toàn trên đường truyền, bảo vệ thông tin khỏi các hành vi nghe lén trái phép. Thuật toán mã hóa mạnh mẽ như TLS\_AES\_256\_GCM\_SHA384 càng củng cố khả năng bảo mật của hệ thống.

Kết luận tổng thể: Hệ thống chat đã phát triển đáp ứng tốt các yêu cầu về bảo mật, cung cấp một kênh giao tiếp an toàn và đáng tin cậy cho người dùng!

# KẾT LUẬN

Dự án của chúng em đã có thể đảm bảo được mức độ an toàn về các mặt: Thiết lập kết nối giữa Client-Server; Đảm bảo thông tin khi trung chuyển được “bọc” lại một cách an toàn, không thể can thiệp và đọc được. Mặc dù còn một vài thiếu sót trong quá trình phát triển như phần tối ưu giao diện, các chức năng trong việc nhắn tin, độ phức tạp nhỏ khi sử dụng ứng dụng (thiếu server thật, không thể public ra internet, không phải là một app riêng)… Chúng em mong nhận được sự góp ý, chỉ bảo của thầy để đề tài của chúng em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

# PHỤ LỤC

**Link github toàn bộ dự án:**