

Bộ Giáo Dục Và Đào Tạo
Trường Đại Học Ngoại Ngữ - Tin Học Thành Phố Hồ Chí Minh
Khoa Công Nghệ Thông Tin



MÔN HỌC: LẬP TRÌNH MẠNG NÂNG CAO

**ĐỀ TÀI : XÂY DỰNG HỆ THỐNG CLIENT- SERVER BẢO MẬT VỚI
XÁC THỰC SỐ VÀ QUÉT SUBDOMAIN TỰ ĐỘNG**

Giảng Viên Hướng Dẫn: ThS. Phan Gia Lượng

Thành Viên:

1. Nguyễn Thị Kim Doanh – 22DH110511
2. Nguyễn Thúy Vy – 22DH114363

TP. Hồ Chí Minh, ngày 02 tháng 08 năm 2025

NHẬN XÉT CỦA GIẢNG VIÊN

[illegible]

LỜI CẢM ƠN

Đầu tiên chúng em xin gửi lời cảm ơn sâu sắc đến Trường Đại học Ngoại ngữ - Tin học TP. Hồ Chí Minh (HUFLIT) đã đưa bộ môn “Lập Trình Mạng Nâng Cao” vào chương trình giảng dạy. Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc đến giảng viên bộ môn – thầy Phan Gia Lượng về sự hướng dẫn và hỗ trợ quý báu của thầy trong môn học. Chính thầy là người đã tận tình dạy bảo và truyền đạt những kiến thức quý báu cho chúng em trong suốt học kì vừa qua. Trong thời gian tham dự lớp học của thầy, chúng em đã tiếp cận với rất nhiều kiến thức bổ ích và rất cần thiết cho quá trình học tập, làm việc sau này của chúng em.

Nhờ những kiến thức sâu rộng và sự nhiệt tình của thầy trong lớp thực hành chúng em đã giúp chúng em có cơ hội thực hiện bài tập nhóm và nhiều kiến thức mới một cách hiệu quả, cũng như là thực hiện bài báo cáo cuối kì đạt được kết quả tốt. Tuy nhiên những kiến thức và kĩ năng về môn học này của chúng em vẫn còn nhiều hạn chế. Do đó bài báo cáo cuối kỳ của chúng em khó tránh khỏi những sai sót. Kính mong cô xem xét và góp ý giúp bài báo cáo cuối kỳ của chúng em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn thầy! Chúng em xin chúc thầy luôn khỏe, vui vẻ và đạt được những thành công trong công tác giảng dạy.

MỤC LỤC

MỤC LỤC.....	3
DANH MỤC HÌNH ẢNH.....	7
CHƯƠNG I. GIỚI THIỆU ĐỀ TÀI.....	9
1. Giới thiệu đề tài.....	9
2. Mục tiêu nghiên cứu.....	10
3. Phạm vi nghiên cứu.....	10
CHƯƠNG II. CƠ SỞ LÝ THUYẾT.....	11
1. Mã hóa và xác thực (Encryption).....	11
1.1. Mã hóa đối xứng (Symmetric Encryption).....	11
1.2. Mã hóa bất đối xứng (Asymmetric Encryption).....	11
1.3. Chữ ký số (Digital Signature).....	11
1.4. Giao thức SSL/TLS	11
2. Lập trình Socket Client – Server.....	11
2.1. Socket TCP	11
2.2. Các bước triển khai Socket.....	12
2.3. Bảo mật Socket	12
3. Xử lý dữ liệu trong Netty (Codec, Pipeline).....	12
3.1. Codec trong Netty.....	12
3.2. Pipeline trong Netty	12
4. Pipeline – Chuỗi xử lý dữ liệu	12
4.1. Handler inbound / outbound	12
4.2. Ứng dụng trong bảo mật.....	12
5. Codec – Mã hóa và giải mã dữ liệu.....	13

5.1.	Tích hợp mã hóa và xác thực trong codec	13
5.2.	Ví dụ:	13
6.	Các handler phổ biến.....	13
6.1.	SSLHandler.....	13
6.2.	AuthenticationHandler.....	13
6.3.	BusinessLogicHandler	13
6.4.	LoggingHandler	13
7.	Đa luồng và Exacutor.....	13
7.1.	Vấn đề	13
7.2.	ExecutorService (Java)	13
7.3.	Netty EventLoop.....	13
8.	Quét Subdomain.....	14
8.1.	Tầm quan trọng	14
8.2.	Kỹ thuật.....	14
8.3.	Tích hợp tự động.....	14
8.4.	Công cụ đề xuất	14
CHƯƠNG III. PHÂN TÍCH YÊU CẦU.....		15
1.	Yêu cầu chức năng	15
2.	Yêu cầu phi chức năng	16
2.1.	Bảo mật	16
2.2.	Độ tin cậy.....	16
CHƯƠNG IV. PHƯƠNG PHÁP TRIỂN KHAI		17
1.	SHA-256-Thuật toán băm mã hóa (Hashing Algorithm).....	17
2.	RSA-Thuật toán mã hóa bất đối xứng (Asymmetric Encryption)	17

3.	AES (Advanced Encryption Standard)	17
4.	Base64 Encoding	17
5.	Sử dụng Socket cơ bản (Java SE)	18
6.	Sử dụng Netty (Framework lập trình mạng hiệu suất cao)	18
7.	Giải mã dữ liệu nhận (Inbound – từ client gửi đến)	18
8.	Mã hóa dữ liệu gửi (Outbound – từ server trả về)	18
9.	Kết hợp cả hai chiều (Inbound và Outbound)	19
10.	Làm việc với ByteBuf	19
11.	Đa luồng trong Java	19
12.	Executor và Thread Pool	19
13.	Kiến trúc hệ thống	20
14.	Các module và các luồng dữ liệu	22
14.1.	Các Module	22
14.2.	Database	22
14.3.	Các luồng dữ liệu	22
CHƯƠNG V. TRIỂN KHAI		24
1.	Môi trường phát triển	24
1.1.	IntelliJ IDEA 2025.1.3	24
1.2.	JDK – 23	24
1.3.	MySQL	24
2.	Cấu trúc mã nguồn	25
3.	Các đoạn mã nguồn quan trọng	27
3.1.	Class ClientApplication.java	27
3.2.	Class ClientHandler.java	28

3.3.	Class CryptoClient.java	29
3.4.	Class NettyClient.java	31
3.5.	Class CryptoUtils.java	32
3.6.	CryptoServer.java	36
3.7.	Class DatabaseManager.java	37
3.8.	Class NettyServer	38
3.9.	ServerApplication	39
3.10.	ServerHandler.java.....	40
3.11.	Class SubdomainScanner.java	43
CHƯƠNG IV. ĐÁNH GIÁ VÀ KẾT LUẬN		46
1.	Kết quả thực hiện	46
2.	Đánh giá hệ thống	51
3.	Kết luận	51
4.	Hạn chế và phương hướng phát triển	52
4.1.	Hạn chế	52
4.1.	Phương hướng phát triển	52
TÀI LIỆU THAM KHẢO.....		53

DANH MỤC HÌNH ẢNH

Hình 1. Kiến trúc hệ thống	20
Hình 2. Cấu trúc mã nguồn	25
Hình 3. Đoạn mã gửi message đến Server	27
Hình 4. Đoạn mã nhận về xác thất bại nếu chữ ký sai và nhận về các subdomain nếu xác thực thành công	28
Hình 5. Đoạn mã tạo chữ ký số SHA256withRSA	29
Hình 6. Đoạn mã mã hoá toàn bộ message hoặc public key bằng AES CBC.	29
Hình 7. Đoạn mã mã hoá toàn bộ nội dung cần gửi.....	30
Hình 8. Đoạn mã mã hoá AES key bằng RSA.....	30
Hình 9. Đoạn mã gửi kèm IV và PublicKey	30
Hình 10. Đoạn mã tạo và mã hoá message	31
Hình 11. Đoạn mã chuyển message sang JSON và gửi	31
Hình 12. Đoạn mã kết nối và cấu hình Netty client	32
Hình 13. Đoạn mã tạo khoá AES & IV	33
Hình 14. Đoạn mã mã hoá và giải mã AES	33
Hình 15. Đoạn mã mã hoá / Giải mã RSA	34
Hình 16. Đoạn mã ký và xác minh chữ ký số SHA256withRSA	34
Hình 17. Đoạn mã load khoá công khai từ file / string	35
Hình 18. Đoạn mã load khoá cá nhân từ file / string	35
Hình 19. Đoạn mã chuyển public key thành base64 để gửi trong message.....	35
Hình 20. Đoạn mã chuyển secret key (AES) thành bytes để mã hoá gửi đi	35
Hình 21. Đoạn mã load khóa riêng và khóa công khai của server.....	36
Hình 22. Đoạn mã giải mã AES key và tin nhắn	36
Hình 23. Đoạn mã giải mã encryptedMessage để lấy nội dung gốc client gửi.....	36
Hình 24. Đoạn mã xác thực chữ ký SHA256withRSA.....	37
Hình 25. Đoạn mã sau khi nhận được message, server sẽ giải mã và lưu clientPublicKey, AES key, IV để quản lý phiên.	38
Hình 26. Đoạn mã cho phép server nhận và xử lý các message dạng text.	38

Hình 27. Đoạn mã khởi tạo database	39
Hình 28. Đoạn mã khởi chạy Server	39
Hình 29. Đoạn mã đóng Server khi thoát.....	39
Hình 30. Đoạn mã xử lý Message từ Client	40
Hình 31. Đoạn mã lưu AES Key và IV vào database	40
Hình 32. Đoạn mã giải mã toàn bộ message bằng AES-CBC	41
Hình 33. Đoạn mã xác thực chữ ký SHA256withRSA.....	41
Hình 34. Đoạn mã scan subdomain khi xác thực thành công	42
Hình 35. Đoạn mã -xử lý lỗi	42
Hình 36. Đoạn mã - Đọc wordlist từ file resources.....	43
Hình 37. Đoạn mã tạo ExecutorService để quét song song các subdomain	44
Hình 38. Đoạn mã thu thập kết quả từ các Future.....	45
Hình 39. Hàm kiểm tra DNS hợp lệ.....	45
Hình 40. Giao diện console Server.....	46
Hình 41. Giao diện console Client	47
Hình 42. Server đã các mình thành công và bắt đầu scan domain.....	48
Hình 43. Server trả kết quả về cho Client	49
Hình 44. Client nhận được thông báo xác thực thành công.	50
Hình 45. Danh sách subdomain Server trả về	50

CHƯƠNG I. GIỚI THIỆU ĐỀ TÀI

1. Giới thiệu đề tài

Trong thời đại công nghệ thông tin phát triển mạnh mẽ, vấn đề bảo mật thông tin và xác thực nguồn gốc dữ liệu đang trở thành một trong những thách thức lớn đối với các hệ thống thông tin, đặc biệt là trong mô hình Client - Server. Việc trao đổi thông tin giữa client và server nếu không được bảo vệ đúng cách có thể tạo ra lỗ hổng cho các cuộc tấn công như giả mạo, nghe lén, hoặc sửa đổi dữ liệu. Do đó, việc áp dụng các phương pháp mã hóa và xác thực là vô cùng quan trọng để đảm bảo an toàn cho quá trình truyền tải dữ liệu.

Đề tài “Xây dựng hệ thống Client-Server xác thực chữ ký số và kiểm tra bảo mật subdomain sử dụng mã hóa” được xây dựng với mục đích tạo ra một mô hình hệ thống giao tiếp giữa client và server có khả năng bảo mật và xác thực đáng tin cậy. Trong hệ thống này, client thực hiện ký số thông điệp bằng thuật toán bất đối xứng (RSA) và gửi kèm theo khóa công khai đã được mã hóa bằng thuật toán đối xứng (AES-CBC). Server tiếp nhận, giải mã và xác minh chữ ký để xác định tính toàn vẹn và xác thực nguồn gốc của thông điệp. Nếu quá trình xác minh thành công, server sẽ thực hiện việc quét tất cả các subdomain của hệ thống (ví dụ: huflit.edu.vn) dựa trên danh sách wordlist được cung cấp. Kết quả sẽ được trả về cho client, giúp người dùng đánh giá được mức độ an toàn của hệ thống tên miền con.

Đề tài không chỉ giúp sinh viên hiểu rõ hơn về cách thức hoạt động của các thuật toán mã hóa hiện đại mà còn góp phần vào việc tăng cường kỹ năng lập trình mạng, xử lý dữ liệu và kiểm thử bảo mật trong môi trường thực tế.

2. Mục tiêu nghiên cứu

- Xây dựng hệ thống Client-Server có khả năng xác thực người dùng dựa trên kỹ thuật ký số RSA và thuật toán băm SHA-256.
- Mã hóa an toàn public key được gửi từ client bằng phương pháp mã hóa đối xứng AES ở chế độ CBC nhằm đảm bảo tính bảo mật trong quá trình truyền tải.
- Thực hiện xác minh chữ ký tại phía server để kiểm tra xem thông điệp có thực sự được gửi từ client hợp lệ hay không.
- Triển khai tính năng quét subdomain tự động, hỗ trợ kiểm thử bảo mật tên miền con của hệ thống huflit.edu.vn dựa trên từ điển wordlist phổ biến được lấy từ cộng đồng mã nguồn mở.
- Xây dựng cơ chế phản hồi thông minh, giúp client nhận được kết quả xác thực và quét subdomain một cách rõ ràng, chính xác và bảo mật.

3. Phạm vi nghiên cứu

Phạm vi thực hiện của đề tài được giới hạn trong các yếu tố sau:

- Ngôn ngữ lập trình: Sử dụng Java kết hợp với các thư viện bảo mật và thư viện xử lý mạng như Java Cryptography Architecture (JCA), Netty.
- Mô hình hệ thống: Mô hình Client-Server truyền thống với giao tiếp hai chiều.
- Phương pháp mã hóa:
 - o Mã hóa bất đối xứng: RSA (dùng để ký số thông điệp).
 - o Mã hóa đối xứng: AES (dùng để mã hóa public key ở phía client).
- Thuật toán xác thực: SHA256withRSA được sử dụng để ký và xác minh thông điệp.
- Phạm vi tên miền quét: Chỉ tập trung vào quét các subdomain của hệ thống huflit.edu.vn.
- Cơ sở dữ liệu: Sử dụng để lưu trữ key và IV được tạo ngẫu nhiên trong quá trình mã hóa, phục vụ cho việc giải mã public key tại phía server.

CHƯƠNG II. CƠ SỞ LÝ THUYẾT

1. Mã hóa và xác thực (Encryption)

Trong bối cảnh an ninh mạng ngày càng phức tạp, việc áp dụng các công nghệ mã hóa để đảm bảo tính bí mật, toàn vẹn và xác thực trong trao đổi dữ liệu trên hệ thống Client-Server là yêu cầu cấp thiết.

1.1. Mã hóa đối xứng (Symmetric Encryption)

- Cùng một khóa được dùng cho mã hóa và giải mã (ví dụ: AES, DES).
- Tốc độ nhanh, nhưng vẫn đề là phân phối khóa an toàn.

1.2. Mã hóa bất đối xứng (Asymmetric Encryption)

- Dùng cặp khóa công khai (public key) và riêng tư (private key).
- Ví dụ: RSA, ECC. Giúp xác thực danh tính giữa client và server.

1.3. Chữ ký số (Digital Signature)

- Ký số dữ liệu bằng private key của người gửi.
- Bên nhận dùng public key để xác minh.
- Đảm bảo tính toàn vẹn và xác minh nguồn gốc.

1.4. Giao thức SSL/TLS

- Được sử dụng phổ biến trong các kết nối Client-Server như HTTPS, FTPS.
- Kết hợp giữa mã hóa bất đối xứng (trao đổi khóa) và đối xứng (mã hóa dữ liệu).

2. Lập trình Socket Client – Server

Lập trình socket cho phép hai ứng dụng máy tính trao đổi dữ liệu trên giao thức TCP hoặc UDP.

2.1. Socket TCP

- Kết nối tin cậy, ba bước: SYN - SYN/ACK - ACK.
- Dùng trong các hệ thống yêu cầu toàn vẹn dữ liệu.

2.2. Các bước triển khai Socket

- Server: Mở cổng (bind port), chờ client (listen), chấp nhận kết nối (accept).
- Client: Khởi tạo socket, kết nối server (connect), gửi/nhận dữ liệu.

2.3. Bảo mật Socket

- Kết hợp OpenSSL, xác thực số.
- Giới hạn địa chỉ IP truy cập, bỏ qua kẻ tấn công.

3. Xử lý dữ liệu trong Netty (Codec, Pipeline)

Netty là framework mạnh mẽ giúp lập trình mạng theo mô hình non-blocking I/O với hiệu năng cao.

3.1. Codec trong Netty

- Encoder: chuyển đổi object → byte để gửi.
- Decoder: chuyển byte → object sau khi nhận.
- Giúp tách biệt logic dữ liệu và trình truyền.

3.2. Pipeline trong Netty

- Pipeline như một chuỗi các handler xử lý dữ liệu theo thứ tự.
- Mỗi handler xử lý 1 nhiệm vụ: decode, authenticate, business logic...

4. Pipeline – Chuỗi xử lý dữ liệu

Trong Netty, ChannelPipeline là trung tâm xử lý dữ liệu:

4.1. Handler inbound / outbound

- Inbound Handler: xử lý dữ liệu nhận (Decode, Auth, Business Logic).
- Outbound Handler: xử lý dữ liệu gửi (Encode, Logging, SSL).

4.2. Ứng dụng trong bảo mật

- Handler Decode + Xác thực = tách dữ liệu + kiểm tra danh tính.
- Handler Encode + Ký số = đóng gói dữ liệu bằng chữ ký số.

5. Codec – Mã hóa và giải mã dữ liệu

5.1. Tích hợp mã hóa và xác thực trong codec

- Trong encoder: mã hóa dữ liệu bằng AES, RSA.
- Trong decoder: giải mã + xác minh chữ ký số.

5.2. Ví dụ:

- Client gửi dữ liệu: encode(data + signature).
- Server nhận và decode: verify(data, signature).

6. Các handler phổ biến

6.1. SSLHandler

- Mã hóa kênh truyền TCP bằng SSL/TLS.

6.2. AuthenticationHandler

- Kiểm tra token, chữ ký số của client.

6.3. BusinessLogicHandler

- Xử lý nghiệp vụ chính (lịch sử dụng, truy vấn DB,...).

6.4. LoggingHandler

- Ghi log hoạt động truy cập, dữ liệu trao đổi.

7. Đa luồng và Executor

7.1. Vấn đề

- Server phục vụ nhiều client = cần đa luồng.

7.2. ExecutorService (Java)

- Tạo ThreadPool, chia việc xử lý.
- Tránh tạo mới quá nhiều luồng.

7.3. Netty EventLoop

- Mỗi kết nối gán cho 1 EventLoop (1 thread).

- Xử lý I/O phi đồng bộ (Non-blocking).

8. Quét Subdomain

8.1. Tầm quan trọng

- Giúp xác định bề mặt tấn công địa chỉ subdomain chưa được quản lý.
- Phát hiện các dịch vụ ngầm, API, staging server...

8.2. Kỹ thuật

- Brute-force: thử tất cả các tên trong từ điển.
- Certificate Transparency: tìm subdomain qua các chứng thư CA.
- Public APIs: VirusTotal, Shodan, SecurityTrails.

8.3. Tích hợp tự động

- Tạo module scan subdomain theo lịch.
- Ghi nhận kết quả và cảnh báo khi xuất hiện subdomain mới.

8.4. Công cụ đề xuất

- Sublist3r, Amass, Assetfinder, Subfinder.
- Python script dựa trên DNS request để scan nhanh.

CHƯƠNG III. PHÂN TÍCH YÊU CẦU

Để đảm bảo hệ thống được xây dựng đúng theo mục tiêu đề ra, việc phân tích yêu cầu là bước cần thiết đầu tiên trong quá trình thiết kế. Yêu cầu của hệ thống được chia thành hai nhóm chính: yêu cầu chức năng (mô tả những gì hệ thống cần thực hiện) và yêu cầu phi chức năng (mô tả chất lượng mà hệ thống cần đạt được như hiệu năng, bảo mật và độ tin cậy).

1. Yêu cầu chức năng

Chức năng chính của hệ thống

- Client:
 - Cho phép người dùng nhập một thông điệp bất kỳ.
 - Ký số thông điệp đó bằng private key RSA sử dụng thuật toán SHA256withRSA.
 - Mã hóa public key bằng thuật toán AES ở chế độ CBC với key và IV được sinh ngẫu nhiên.
 - Gửi gói tin bao gồm raw-message, chữ ký, và public key mã hóa đến server.
- Server:
 - Tiếp nhận dữ liệu từ client.
 - Giải mã public key bằng AES-CBC với key và IV được lưu trong cơ sở dữ liệu.
 - Xác thực chữ ký số: kiểm tra xem chữ ký có hợp lệ với nội dung thông điệp và public key hay không.
 - Nếu chữ ký hợp lệ:
 - Thực hiện quét các subdomain của tên miền huflit.edu.vn bằng cách nối các từ trong wordlist vào tên miền chính và kiểm tra phản hồi DNS.
 - Trả danh sách các subdomain có tồn tại về cho client.
 - Nếu chữ ký không hợp lệ:
 - Trả về thông điệp "VERIFICATION_FAILED" nhằm thông báo cho client biết rằng xác thực thất bại.

2. Yêu cầu phi chức năng

- Hệ thống phải xử lý các yêu cầu xác thực và quét subdomain trong thời gian hợp lý, tối ưu trải nghiệm người dùng.
- Việc xử lý song song (multi-threading) cần được cân nhắc khi có nhiều yêu cầu đến cùng lúc nhằm tránh tình trạng nghẽn hoặc quá tải server, hiệu năng.

2.1. Bảo mật

- Việc kết hợp mã hóa bất đối xứng (RSA) và mã hóa đối xứng (AES - CBC) giúp tận dụng điểm mạnh của cả hai phương pháp:
 - RSA đảm bảo tính xác thực và không thể chối bỏ của người gửi.
 - AES giúp giảm thời gian mã hóa/giải mã public key với tốc độ cao và tiết kiệm tài nguyên.
- Thông tin key và IV được lưu trữ an toàn trong cơ sở dữ liệu server, tránh nguy cơ bị lộ trong quá trình truyền tải.
- Dữ liệu truyền giữa client và server được bảo vệ khỏi các nguy cơ tấn công như giả mạo, sửa đổi nội dung, hoặc nghe lén.

2.2. Độ tin cậy

- Hệ thống cần có khả năng hoạt động ổn định trong mọi điều kiện, đảm bảo không trả về sai kết quả xác thực hoặc quét sai subdomain.
- Trong trường hợp xảy ra lỗi (ví dụ: dữ liệu gửi đến bị sai định dạng, chữ ký không hợp lệ, lỗi kết nối mạng), server phải phản hồi rõ ràng và không gây ảnh hưởng đến hệ thống chung.
- Việc xử lý lỗi và ghi log cần được thực hiện đầy đủ để phục vụ cho việc kiểm tra, bảo trì và mở rộng hệ thống sau này.

CHƯƠNG IV. PHƯƠNG PHÁP TRIỂN KHAI

1. SHA-256-Thuật toán băm mã hóa (Hashing Algorithm)

Mục đích: Sử dụng để bảo mật mật khẩu và xác thực tính toàn vẹn dữ liệu

- Triển khai:
 - Tạo hàm băm SHA-256 cho mật khẩu trước khi lưu trữ
 - Kiểm tra tính toàn vẹn của dữ liệu bằng cách so sánh giá trị băm
 - Sử dụng lớp MessageDigest trong Java để triển khai

2. RSA-Thuật toán mã hóa bất đối xứng (Asymmetric Encryption)

Mục đích: Trao đổi khóa an toàn và xác thực server

- Triển khai:
 - Tạo cặp khóa public/private sử dụng KeyPairGenerator
 - Mã hóa khóa phiên (session key) bằng RSA trước khi gửi
 - Giải mã bằng private key trên server
 - Độ dài khóa đề xuất: 2048 bit hoặc cao hơn

3. AES (Advanced Encryption Standard)

Mục đích: Mã hóa dữ liệu trao đổi với hiệu suất cao

- Triển khai:
 - Sử dụng AES-256 trong chế độ GCM (Galois/Counter Mode)
 - Tạo khóa ngẫu nhiên cho mỗi phiên làm việc
 - Triển khai bằng lớp Cipher của Java Cryptography Extension (JCE)

4. Base64 Encoding

Mục đích: Chuyển đổi dữ liệu nhị phân thành dạng text an toàn

- Triển khai:
 - Mã hóa dữ liệu đã mã hóa thành chuỗi Base64 trước khi truyền
 - Giải mã khi nhận dữ liệu từ mạng
 - Sử dụng lớp Base64 trong Java 8+

5. Sử dụng Socket cơ bản (Java SE)

- Triển khai:
 - Tạo server socket với ServerSocket trên cổng xác định
 - Client kết nối bằng lớp Socket
 - Thiết lập luồng vào/ra (InputStream/OutputStream)
 - Xử lý kết nối đồng thời bằng đa luồng

6. Sử dụng Netty (Framework lập trình mạng hiệu suất cao)

Mục đích: Xây dựng hệ thống mạng hiệu suất cao

- Triển khai:
 - Thiết lập EventLoopGroup cho server và client
 - Tạo Bootstrap/ServerBootstrap để cấu hình pipeline
 - Triển khai ChannelHandler để xử lý dữ liệu mã hóa
 - Tận dụng ByteBuf cho hiệu suất cao

7. Giải mã dữ liệu nhận (Inbound – từ client gửi đến)

- Quy trình:
 - Nhận dữ liệu từ client dưới dạng byte array hoặc ByteBuf
 - Giải mã Base64 nếu cần
 - Giải mã bằng khóa bí mật (AES/RSA tùy giai đoạn)
 - Kiểm tra tính toàn vẹn bằng SHA-256
 - Chuyển đổi thành đối tượng/dữ liệu gốc

8. Mã hóa dữ liệu gửi (Outbound – từ server trả về)

- Quy trình:
 - Kiểm tra tính toàn vẹn dữ liệu bằng SHA-256
 - Mã hóa dữ liệu bằng khóa phiên (AES)
 - Mã hóa Base64 nếu cần
 - Gửi dữ liệu đã mã hóa qua socket/Netty channel

9. Kết hợp cả hai chiều (Inbound và Outbound)

- Triển khai:
 - Thiết kế pipeline xử lý 2 chiều
 - Sử dụng cơ chế handshake để trao đổi khóa an toàn
 - Kết hợp giải mã inbound và mã hóa outbound trong cùng kênh giao tiếp
 - Đảm bảo đồng bộ hóa giữa client và server

10. Làm việc với ByteBuffer

- Tối ưu hóa:
 - Sử dụng ByteBuffer thay cho byte array trong Netty
 - Quản lý bộ nhớ hiệu quả với pooled buffers
 - Hỗ trợ xử lý dữ liệu nhị phân cho mã hóa/giải mã
 - Tận dụng tính năng zero-copy khi cần

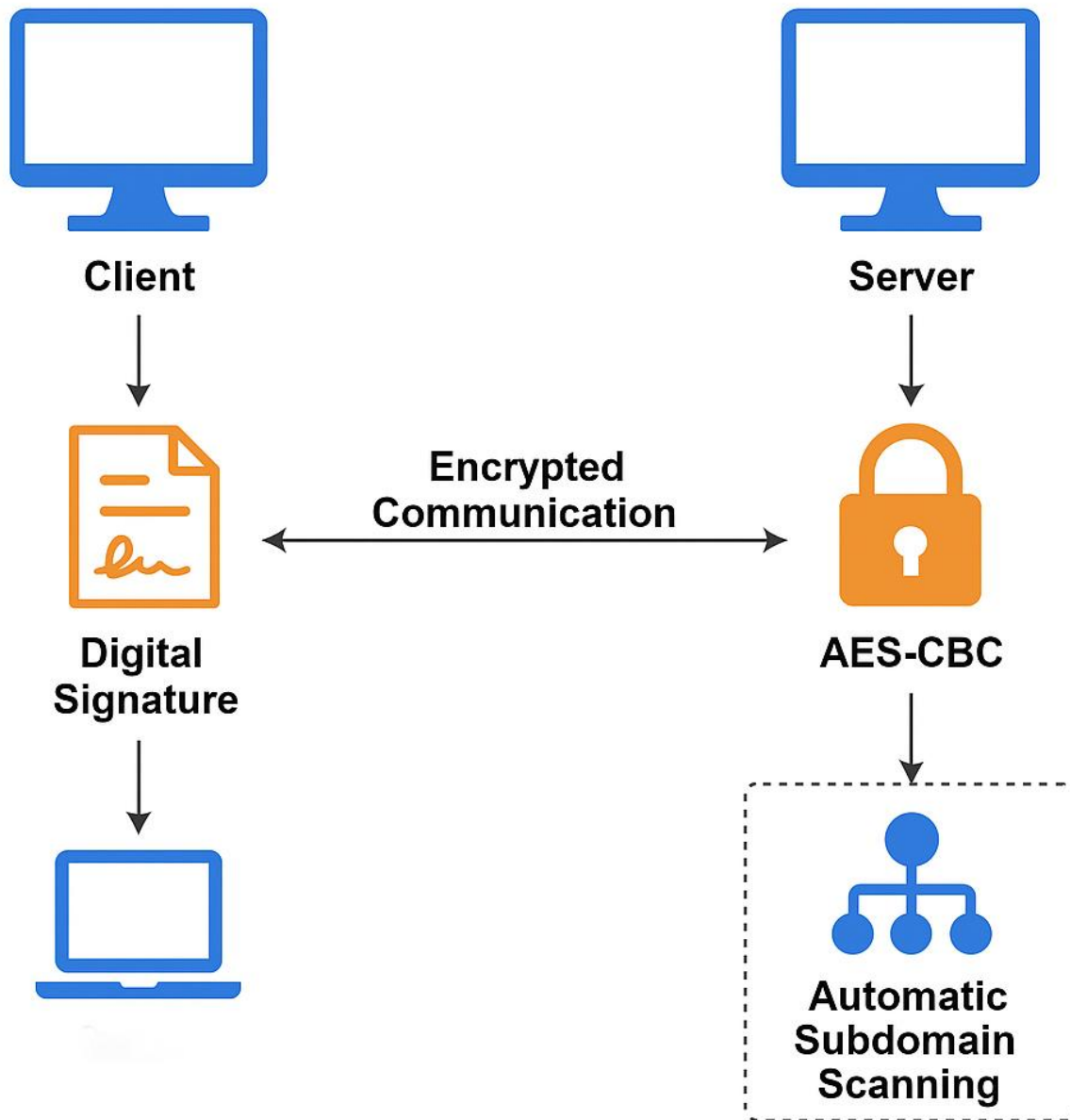
11. Đa luồng trong Java

- Triển khai:
 - Xử lý nhiều kết nối client đồng thời
 - Tách luồng xử lý I/O và nghiệp vụ
 - Đồng bộ hóa truy cập tài nguyên dùng chung
 - Sử dụng cơ chế lock/semaphore khi cần

12. Executor và Thread Pool

- Quản lý tài nguyên:
 - Sử dụng ThreadPoolExecutor để quản lý luồng
 - Cấu hình số luồng tối ưu cho hệ thống
 - Sử dụng ScheduledExecutorService cho tác vụ định kỳ
 - Kết hợp với Netty EventLoop cho hiệu suất cao

13. Kiến trúc hệ thống



Hình 1. Kiến trúc hệ thống

Client:

1. Ký số thông điệp (Digital Signature)

- Client tạo chữ ký số từ thông điệp gốc (ví dụ: tên miền cần quét) bằng private key và thuật toán SHA256withRSA.
- Mục tiêu: đảm bảo rằng chỉ client hợp lệ mới có thể gửi thông điệp được xác thực.

2. Mã hóa public key bằng AES-CBC

- Public key của client được mã hóa bằng AES chế độ CBC, sử dụng AES key và IV ngẫu nhiên sinh ra cho mỗi phiên làm việc.
- Giúp tránh bị lộ thông tin khóa khi truyền qua mạng.

3. Gửi gói tin

- Client gửi:
 - Thông điệp gốc (raw message),
 - Chữ ký số (digital signature),
 - Public key đã mã hóa (AES-CBC),
 - AES key và IV (được mã hóa bằng RSA public key của Server).

Encrypted Communication:

- Tất cả dữ liệu gửi qua socket được mã hóa an toàn, đảm bảo tính bảo mật khi truyền tải giữa Client và Server.

Server:

4. Giải mã và xác thực

- Server sử dụng private key của mình để giải mã AES key và IV.
- Dùng AES key + IV để giải mã public key của client.
- Xác minh chữ ký số bằng public key client vừa giải mã:
- Nếu hợp lệ → thực hiện tiếp bước sau.
- Nếu sai → trả về "VERIFICATION_FAILED".

5. AES-CBC Layer

- Giai đoạn giải mã toàn bộ quá trình xác thực, phục vụ việc đảm bảo rằng chỉ dữ liệu hợp lệ mới được xử lý tiếp theo.

Quét Subdomain tự động (Automatic Subdomain Scanning):

- Nếu xác thực thành công, server thực hiện quét subdomain (ví dụ: *.huflit.edu.vn) bằng cách:

- Đọc wordlist,
- Tạo URL con (subdomain),
- Kiểm tra phản hồi từ DNS/HTTP.
- Gửi danh sách subdomain hợp lệ về lại cho client dưới dạng JSON.

14. Các module và các luồng dữ liệu

14.1. Các Module

- ClientNetty.java: Gửi dữ liệu đã mã hóa và ký số đến server qua Netty socket.
- GenerateSignature.java: Tạo chữ ký số bằng thuật toán SHA256withRSA.
- AESCBC.java: Mã hóa và giải mã bằng thuật toán AES chế độ CBC.
- RSAUtil.java: Sinh khóa RSA, mã hóa AES key và IV bằng RSA public key của server
- ServerNetty.java: Thành phần xử lý chính của server, nhận và xử lý các kết nối client, giải mã dữ liệu, xác thực chữ ký số, thực hiện quét subdomain.
- VerifySignature.java: Dùng để xác thực chữ ký từ client bằng public key đã được giải mã.

14.2. Database

- Lưu trữ:
 - AES key và IV (mã hóa) theo từng phiên.
 - Log xác thực (thành công/thất bại).
 - Lịch sử client (nếu mở rộng).
- Sử dụng MySQL.

14.3. Các luồng dữ liệu

14.3.1. Từ Client đến Server

1. Người dùng nhập domain cần quét trên client.
2. Client ký số thông điệp (raw domain) bằng private key.
3. Client mã hóa public key bằng AES-CBC (AES key và IV sinh ngẫu nhiên).
4. AES key và IV được mã hóa bằng RSA public key của Server.
5. Client gửi đến Server:

- raw message (domain)
- chữ ký số
- public key mã hóa AES
- AES key mã hóa RSA
- IV mã hóa RSA

14.3.2.Xử lý tại Server

6. Server giải mã AES key và IV bằng private key.
7. Giải mã public key của client bằng AES.
8. Xác thực chữ ký số với raw message.
9. Nếu hợp lệ:
 - Quét các subdomain bằng wordlist.
 - Ghi log phiên làm việc và kết quả.
 - Trả kết quả JSON về cho client.
10. Nếu sai:
 - Trả "VERIFICATION_FAILED".

14.3.3.Trả dữ liệu Client

11. Client nhận phản hồi từ server:
 - Kết quả quét subdomain (nếu xác thực đúng).
 - Thông báo lỗi (nếu xác thực thất bại).

CHƯƠNG V. TRIỂN KHAI

1. Môi trường phát triển

1.1. IntelliJ IDEA 2025.1.3

IntelliJ IDEA là một môi trường phát triển tích hợp (IDE) mạnh mẽ dành cho Java, được phát triển bởi JetBrains. Phiên bản 2025.1.3 cung cấp các tính năng thông minh như tự động gợi ý mã, kiểm tra lỗi thời gian thực, hỗ trợ Maven/Gradle, cũng như tích hợp tốt với hệ thống điều khiển phiên bản như Git. Điều này giúp quá trình phát triển ứng dụng diễn ra nhanh chóng, thuận tiện và hiệu quả hơn.

1.2. JDK – 23

Java Development Kit (JDK) 23 là bộ công cụ phát triển mới nhất của ngôn ngữ lập trình Java, hỗ trợ nhiều tính năng mới như cải tiến hiệu suất, bảo mật và hỗ trợ lập trình hướng mô-đun tốt hơn. Việc sử dụng JDK 23 giúp đảm bảo ứng dụng tương thích với các tiêu chuẩn hiện đại và có khả năng tận dụng các tối ưu hóa mới nhất của nền tảng Java.

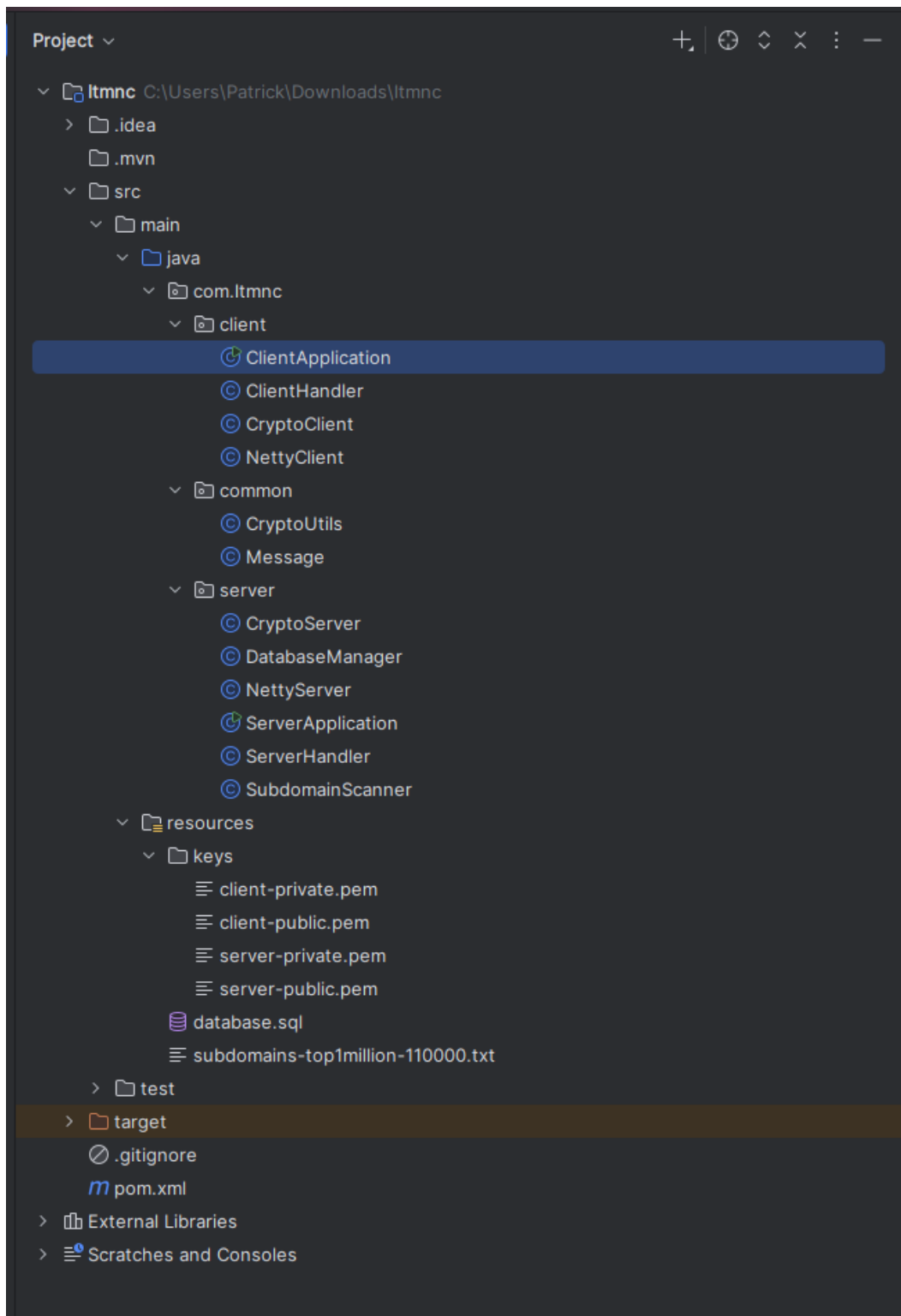
1.3. MySQL

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở phổ biến, được sử dụng để lưu trữ và truy xuất thông tin từ cơ sở dữ liệu một cách hiệu quả. Trong hệ thống này, MySQL được sử dụng để:

- Quản lý thông tin các khóa mã hóa (public key, secret key, IV),
- Lưu lịch sử các phiên giao tiếp giữa Client và Server,
- Ghi log kiểm chứng chữ ký số và kết quả xác thực.

Việc tích hợp MySQL giúp đảm bảo tính toàn vẹn và truy xuất dễ dàng đối với dữ liệu nhạy cảm, đồng thời giúp hỗ trợ việc mở rộng hệ thống về sau.

2. Cấu trúc mã nguồn



Hình 2. Cấu trúc mã nguồn

Tên project: LTMNC

- **.idea và .mvn:**

- **.idea:** Thư mục cấu hình của IntelliJ IDEA.
- **.mvn:** Dùng cho Maven wrapper, chứa file cấu hình Maven.

- **src/main/java** – Chứa mã nguồn Java chính

- **Client/**

- **ClientNetty.java**

Ứng dụng client chính sử dụng thư viện **Netty** để:

- Kết nối tới server.
- Gửi tin nhắn, dữ liệu, public key, chữ ký số, v.v.

- **Server/**

- **ServerNetty.java**

Server chính sử dụng Netty để:

- Nhận kết nối từ client.
- Giải mã dữ liệu, xác minh chữ ký số, xử lý yêu cầu từ client.

- **Utils/ – Các lớp tiện ích dùng chung**

- **AESCBC.java:** Mã hóa và giải mã dữ liệu bằng thuật toán AES-CBC.
- **GenerateSignature.java:** Tạo chữ ký số sử dụng thuật toán SHA256withRSA.
- **RSAUtil.java:** Tạo và quản lý RSA key pair, hỗ trợ mã hóa/giải mã RSA.
- **VerifySignature.java:** Xác minh tính hợp lệ của chữ ký số do client gửi lên.

- **src/main/resources/ – Tài nguyên dùng cho ứng dụng**

- **application.properties:** File cấu hình ứng dụng (port, config,...).
- **log4j.xml:** Cấu hình logging cho hệ thống.

- Các file khác phục vụ cho runtime như:
 - client-private.pem, client-public.pem
 - server-private.pem, server-public.pem
 - subdomains-top1million-110000.txt
 - database.sql
- **pom.xml**
 - File cấu hình chính của Maven:
 - Khai báo các dependencies (thư viện dùng chung).
 - Cấu hình Java version, các plugin build, test, package,...
 - Có thể bao gồm plugin cho Netty, GSON, Bouncy Castle, JDBC,...

3. Các đoạn mã nguồn quan trọng

3.1. Class ClientApplication.java

Đây là điểm bắt đầu gửi message chứa raw-data, chữ ký và public key mã hoá

```
// Gửi message đến server
System.out.println("Đang gửi message: " + input);
client.sendMessage(input);
```

Hình 3. Đoạn mã gửi message đến Server

3.2. Class ClientHandler.java

- Nhận về "VERIFICATION_FAILED" nếu chữ ký sai.
- Nhận về danh sách subdomain nếu xác thực thành công

```
// Parse JSON response
Message response = objectMapper.readValue(jsonResponse, Message.class);

// Hiển thị response
if (response.getResponse() != null) {
    System.out.println("Response: " + response.getResponse());
}

// Hiển thị scan results nếu có
if (response.getScanResults() != null && !response.getScanResults().isEmpty()) {
    System.out.println("\n=== Kết quả Scan Subdomain ===");
    String[] subdomains = response.getScanResults().split(regex: "\\n");
    System.out.println("Tìm thấy " + subdomains.length + " subdomain:");

    int count = 0;
    for (String subdomain : subdomains) {
        if (!subdomain.trim().isEmpty()) {
            System.out.println("  " + (++count) + ". " + subdomain.trim());
        }
    }

    System.out.println("Tổng cộng: " + count + " subdomain được tìm thấy");
}
```

Hình 4. Đoạn mã nhận về xác thất bại nếu chữ ký sai và nhận về các subdomain nếu xác thực thành công

3.3. Class CryptoClient.java

Client gửi một message với mã hoá BASE64(SHA256withRSA) kèm theo raw-message lên server và public key mã hoá bằng AES CBC

Hoặc mã hoá toàn bộ message bằng AES CBC rồi gửi lên server.

- Tạo chữ ký số SHA256withRSA
 - Client ký raw-message bằng SHA256withRSA, mã hoá BASE64
 - Đây là chữ ký dùng để server xác thực nội dung có bị giả mạo không.

```
// Bước 1: Tạo chữ ký số SHA256withRSA
System.out.println("1. Tạo chữ ký số với SHA256withRSA...");
String signature = CryptoUtils.signWithRSA(rawMessage, clientPrivateKey);
message.setSignature(signature);
System.out.println("    ✓ Chữ ký đã được tạo: " + signature.substring(0, 50) + "...");
```

Hình 5. Đoạn mã tạo chữ ký số SHA256withRSA

- Tạo AES key và IV
 - Mã hoá toàn bộ message hoặc public key bằng AES CBC.

```
// Bước 2: Generate AES key và IV
System.out.println("2. Tạo AES key và IV...");
SecretKey aesKey = CryptoUtils.generateAESKey();
byte[] iv = CryptoUtils.generateIV();
System.out.println("    ✓ AES key và IV đã được tạo");
```

Hình 6. Đoạn mã mã hoá toàn bộ message hoặc public key bằng AES CBC.

- hoá toàn bộ raw-message bằng AES CBC: Mã hoá toàn bộ nội dung cần gửi.

```
// Bước 3: Mã hóa message bằng AES-CBC
System.out.println("3. Mã hóa message bằng AES-CBC...");
String encryptedMessage = CryptoUtils.encryptAES(rawMessage, aesKey, iv);
message.setEncryptedMessage(encryptedMessage);
System.out.println("    ✓ Chữ ký đã được tạo: " +
    (signature.length() > 50 ? signature.substring(0, 50) + "...": signature));
```

Hình 7. Đoạn mã mã hoá toàn bộ nội dung cần gửi

- Mã hoá AES key bằng RSA (server's public key)
 - Gửi AES key đã được mã hoá bằng RSA cho Server → Server có thể giải mã AES key để giải nội dung.

```
// Bước 4: Mã hóa AES key bằng RSA public key của server
System.out.println("4. Mã hóa AES key bằng RSA...");
byte[] aesKeyBytes = CryptoUtils.secretKeyToBytes(aesKey);
String encryptedAESKey = CryptoUtils.encryptRSA(aesKeyBytes, serverPublicKey);
message.setEncryptedAESKey(encryptedAESKey);
System.out.println("    ✓ AES key đã được mã hóa: " + encryptedAESKey.substring(0, 50) + "...");
```

Hình 8. Đoạn mã mã hoá AES key bằng RSA

- Gửi kèm IV và PublicKey (ở dạng Base64 hoặc PEM)
 - Gửi IV (để AES CBC giải mã)
 - Gửi public key → Server dùng để verify chữ ký

```
String ivBase64 = Base64.getEncoder().encodeToString(iv);
message.setIv(ivBase64);

String publicKeyStr = CryptoUtils.publicKeyToString(clientPublicKey);
message.setPublicKey(publicKeyStr);
```

Hình 9. Đoạn mã gửi kèm IV và PublicKey

3.4. Class NettyClient.java

- Tạo và mã hoá message: Gọi đến CryptoClient.createEncryptedMessage()
 - Ký số raw-message bằng SHA256withRSA
 - Mã hoá raw-message bằng AES-CBC
 - Mã hoá AES key bằng RSA
 - Encode IV và client public key

```
// Tạo và mã hóa message
Message message = cryptoClient.createEncryptedMessage(rawMessage);
```

Hình 10. Đoạn mã tạo và mã hoá message

- Chuyển message sang JSON và gửi
 - Dữ liệu được serialize thành JSON (chuỗi ký tự) để server dễ dàng xử lý
 - \n để server phân tách các message (dùng decoder string theo dòng)

```
// Convert message thành JSON string
String jsonMessage = objectMapper.writeValueAsString(message);

// Gửi message qua channel
if (channel != null && channel.isActive()) {
    channel.writeAndFlush(o: jsonMessage + "\n");
    System.out.println("Đã gửi message được mã hóa đến server");
} else {
    System.err.println("Channel không active, không thể gửi message");
}
```

Hình 11. Đoạn mã chuyển message sang JSON và gửi

- Kết nối và cấu hình Netty client
 - Dùng StringDecoder + StringEncoder để gửi nhận JSON text
 - Dùng ClientHandler để nhận và xử lý phản hồi từ server (như "VERIFICATION_FAILED" hoặc danh sách subdomains)

```
Bootstrap bootstrap = new Bootstrap();
bootstrap.group(group)
    .channel(NioSocketChannel.class)
    .handler(new ChannelInitializer<SocketChannel>() {
        @Override no usages
        protected void initChannel(SocketChannel ch) throws Exception {
            ChannelPipeline pipeline = ch.pipeline();
            pipeline.addLast(new StringDecoder(CharsetUtil.UTF_8));
            pipeline.addLast(new StringEncoder(CharsetUtil.UTF_8));
            pipeline.addLast(new ClientHandler());
        }
    });
```

Hình 12. Đoạn mã kết nối và cấu hình Netty client

3.5. Class CryptoUtils.java

Đây là trung tâm xử lý mã hoá, ký số và giải mã, cũng như load key từ file

- Tạo khoá AES & IV (symmetric key encryption)
 - Sinh khoá AES ngẫu nhiên (256 bit) và khởi tạo IV (Initialization Vector – 16 byte).
 - Dùng cho mã hoá toàn bộ tin nhắn Client gửi đến Server bằng AES/CBC.

```
// Generate AES Key
public static SecretKey generateAESKey() throws Exception { 2 usages
    KeyGenerator keyGen = KeyGenerator.getInstance( algorithm: "AES");
    keyGen.init( keysize: 256);
    return keyGen.generateKey();
}

// Generate IV for AES
public static byte[] generateIV() { 2 usages
    SecureRandom random = new SecureRandom();
    byte[] iv = new byte[16];
    random.nextBytes(iv);
    return iv;
}
```

Hình 13. Đoạn mã tạo khoá AES & IV

- Mã hoá và giải mã AES (CBC mode)
 - Dùng để mã hoá dữ liệu raw hoặc cả message bằng AES với CBC mode.
 - Lý do quan trọng: Dùng phần này để:
 - Mã hoá toàn bộ Message
 - Mã hoá public key gửi đến server
 - Mã hoá chữ ký hoặc payload

```
// AES CBC Encryption
public static String encryptAES(String plainText, SecretKey key, byte[] iv) throws Exception { 2 usages
    Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/PKCS5Padding");
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
    byte[] encryptedBytes = cipher.doFinal(plainText.getBytes( charsetName: "UTF-8"));
    return Base64.getEncoder().encodeToString(encryptedBytes);
}

// AES CBC Decryption
public static String decryptAES(String encryptedText, SecretKey key, byte[] iv) throws Exception { 2 usages
    Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/PKCS5Padding");
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
    byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
    return new String(decryptedBytes, charsetName: "UTF-8");
}
```

Hình 14. Đoạn mã mã hoá và giải mã AES

- Mã hoá / Giải mã RSA (asymmetric encryption)
 - Mã hoá publicKey (hoặc bất kỳ byte nào) bằng RSA.
 - Có thể được dùng để gửi AES key qua mạng (nếu không dùng trực tiếp raw base64).

```
// RSA Encryption
public static String encryptRSA(byte[] data, PublicKey publicKey) throws Exception { 2 usages
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    byte[] encryptedBytes = cipher.doFinal(data);
    return Base64.getEncoder().encodeToString(encryptedBytes);
}
```

Hình 15. Đoạn mã mã hoá / Giải mã RSA

- Ký và xác minh chữ ký số SHA256withRSA
 - Tạo chữ ký số bằng thuật toán SHA-256 và RSA
 - Xác minh tính toàn vẹn + nguồn gốc dữ liệu.
 - Đây là trái tim của xác thực client-server.
 - Client ký message → Server kiểm tra → Nếu sai: VERIFICATION_FAILED.

```
// RSA Sign with SHA256
public static String signWithRSA(String data, PrivateKey privateKey) throws Exception { 2 usages
    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(privateKey);
    signature.update(data.getBytes("UTF-8"));
    byte[] signatureBytes = signature.sign();
    return Base64.getEncoder().encodeToString(signatureBytes);
}
```

Hình 16. Đoạn mã ký và xác minh chữ ký số SHA256withRSA

- Load khoá cá nhân và công khai từ file / string
 - Đọc file .pem từ classpath hoặc từ hệ thống để lấy private key hoặc public key.
 - Nếu Client không load đúng private key → không ký được → server từ chối.

```
// Load Private Key from PEM file - FIXED VERSION
public static PrivateKey loadPrivateKey(String filename) throws Exception { 2 usages
```

Hình 17. Đoạn mã load khoá công khai từ file / string

```
// Load Private Key from file system
public static PrivateKey loadPrivateKeyFromFile(String filepath) throws Exception { 1 usage
```

Hình 18. Đoạn mã load khoá cá nhân từ file / string

- Chuyển đổi định dạng khoá
 - Chuyển public key thành base64 để gửi trong message.

```
// Convert Public Key to String
public static String publicKeyToString(PublicKey publicKey) { 1 usage
    byte[] keyBytes = publicKey.getEncoded();
    String base64Key = Base64.getEncoder().encodeToString(keyBytes);
    return "-----BEGIN PUBLIC KEY-----\n" + base64Key + "\n-----END PUBLIC KEY-----";
}
```

Hình 19. Đoạn mã chuyển public key thành base64 để gửi trong message

- Chuyển secret key (AES) thành bytes để mã hoá gửi đi.

```
// Convert SecretKey to bytes
public static byte[] secretKeyToBytes(SecretKey secretKey) { 1 usage
    return secretKey.getEncoded();
}
```

Hình 20. Đoạn mã chuyển secret key (AES) thành bytes để mã hoá gửi đi

3.6. CryptoServer.java

- Load khóa riêng và khóa công khai của server
 - Dùng private key để giải mã AES key mà client gửi tới.
 - Vai trò: Server cần private key để giải mã encryptedAESKey.

```
// Load server keys
this.serverPrivateKey = CryptoUtils.loadPrivateKey( filename: "keys/server-private.pem");
this.serverPublicKey = CryptoUtils.loadPublicKey( filename: "keys/server-public.pem");
```

Hình 21. Đoạn mã load khóa riêng và khóa công khai của server

- Giải mã AES key và tin nhắn
 - Client gửi AES key đã mã hoá bằng RSA → server dùng private key để giải mã nó ra lại AES key thô.

```
// Bước 1: Giải mã AES key bằng server private key
byte[] aesKeyBytes = CryptoUtils.decryptRSA(request.getEncryptedAESKey(), serverPrivateKey);
SecretKey aesKey = CryptoUtils.bytesToSecretKey(aesKeyBytes);
System.out.println("    ✓ Đã giải mã AES key");
```

Hình 22. Đoạn mã giải mã AES key và tin nhắn

- Sau khi lấy được aesKey và iv, giải mã encryptedMessage (AES-CBC) để lấy nội dung gốc client gửi.

```
// Bước 2: Decode IV
byte[] iv = Base64.getDecoder().decode(request.getIv());
System.out.println("    ✓ Đã decode IV");
```

Hình 23. Đoạn mã giải mã encryptedMessage để lấy nội dung gốc client gửi.

- Xác thực chữ ký SHA256withRSA
 - Server nhận signature và rawMessage, cùng publicKey từ client.
 - Xác thực chữ ký bằng cách:

- Tạo SHA256 hash từ rawMessage
- Dùng RSA public key giải mã signature
- So sánh kết quả với hash gốc
- Kết luận: Nếu hợp lệ → tin nhắn đến từ đúng Client.

```
public boolean verifySignature(String originalMessage, String signature, String clientPublicKeyStr) throws Exception {
    System.out.println("    - Bắt đầu verify signature...");

    // Load client public key từ string
    PublicKey clientPublicKey = CryptoUtils.loadPublicKeyFromString(clientPublicKeyStr);
    System.out.println("        ✓ Đã load client public key");

    // Verify signature với SHA256withRSA
    boolean isValid = CryptoUtils.verifyWithRSA(originalMessage, signature, clientPublicKey);

    if (isValid) {
        System.out.println("        ✓ Signature verification thành công");
    } else {
        System.out.println("        × Signature verification thất bại");
    }

    return isValid;
}
```

Hình 24. Đoạn mã xác thực chữ ký SHA256withRSA

3.7. Class DatabaseManager.java

- Sau khi nhận được message, server sẽ giải mã và lưu clientPublicKey, AES key, IV để quản lý phiên.

```
public void storeKeys(String encryptedAESKey, String iv, String clientPublicKey) throws SQLException { 4 usages
    String insertSQL = "INSERT INTO session_keys (encrypted_aes_key, iv, client_public_key, session_id) " +
        "VALUES (?, ?, ?, ?)";

    // Tạo session ID đơn giản
    String sessionId = "SESSION_" + System.currentTimeMillis();

    try (PreparedStatement statement = connection.prepareStatement(insertSQL)) {
        statement.setString(parameterIndex: 1, encryptedAESKey);
        statement.setString(parameterIndex: 2, iv);
        statement.setString(parameterIndex: 3, clientPublicKey);
        statement.setString(parameterIndex: 4, sessionId);

        int rowsAffected = statement.executeUpdate();

        if (rowsAffected > 0) {
            System.out.println(" ✓ Đã lưu keys vào database với session ID: " + sessionId);
        } else {
            System.err.println(" × Không thể lưu keys vào database");
        }
    }
}
```

Hình 25. Đoạn mã sau khi nhận được message, server sẽ giải mã và lưu clientPublicKey, AES key, IV để quản lý phiên.

3.8. Class NettyServer

Là nơi khởi tạo server Netty

- Cho phép server nhận và xử lý các message dạng text (string + line-based).

```
// Thêm LineBasedFrameDecoder để xử lý messages kết thúc bởi \n
pipeline.addLast(new LineBasedFrameDecoder( maxLength: 8192));|
pipeline.addLast(new StringDecoder(CharsetUtil.UTF_8));
pipeline.addLast(new StringEncoder(CharsetUtil.UTF_8));
pipeline.addLast(new ServerHandler());
```

Hình 26. Đoạn mã cho phép server nhận và xử lý các message dạng text.

3.9. ServerApplication

- Tạo bảng session_keys hoặc các bảng cần thiết khác để lưu AES key, IV, public key từ phía Client gửi lên.

```
// Khởi tạo database
DatabaseManager dbManager = new DatabaseManager();
dbManager.initializeDatabase();
System.out.println("Database đã được khởi tạo thành công");
```

Hình 27. Đoạn mã khởi tạo database

- Mở server Netty để chờ Client kết nối.
- Port 8080 là cổng chính giao tiếp giữa Client và Server để truyền message mã hóa.

```
// Start server
System.out.println("Đang khởi động server trên port 8080...");
server.start();
```

Hình 28. Đoạn mã khởi chạy Server

- Đảm bảo server được dừng sạch sẽ và kết nối CSDL được đóng đúng cách khi kết thúc chương trình.

```
// Thêm shutdown hook để đóng server khi thoát
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    System.out.println("\nĐang thoát server...");
    server.stop();
    try {
        dbManager.close();
    } catch (Exception e) {
        System.err.println("Lỗi khi đóng database: " + e.getMessage());
    }
    System.out.println("Server đã thoát.");
}));
```

Hình 29. Đoạn mã đóng Server khi thoát

3.10. ServerHandler.java

Nhận message từ Client → Giải mã AES → Verify chữ ký SHA256withRSA → Nếu đúng thì scan subdomain huflit.edu.vn, nếu sai thì trả VERIFICATION_FAILED.

- Xử lý Message từ Client
 - Đọc một message JSON từ phía Client gửi sang (gồm: signature, rawMessage, publicKey, encryptedAESKey, iv, v.v...).
 - Chuyển sang dạng Message Java object để xử lý.

```
// Parse JSON message
Message request = objectMapper.readValue(jsonMessage, Message.class);
```

Hình 30. Đoạn mã xử lý Message từ Client

- Lưu AES Key và IV vào database
 - Giữ lại thông tin phiên làm việc: AES key (mã hóa), IV, và public key.
 - Giúp Server có thể theo dõi hoặc tái xử lý sau này (ví dụ cho audit hoặc session).

```
// Bước 1: Lưu AES key/IV vào database
databaseManager.storeKeys(
    request.getEncryptedAESKey(),
    request.getIv(),
    request.getPublicKey());
System.out.println("    ✓ Đã lưu keys vào database");
```

Hình 31. Đoạn mã lưu AES Key và IV vào database

- Giải mã toàn bộ message bằng AES-CBC
 - AES key và IV được dùng để giải mã message từ phía client.
 - Nếu không giải mã đúng, việc xác minh chữ ký tiếp theo sẽ luôn thất bại.
 - Phụ thuộc vào phương thức CryptoServer.decryptMessage(...).

```
// Bước 2: Giải mã message
String decryptedMessage = cryptoServer.decryptMessage(request);
System.out.println("    ✓ Đã giải mã message: " + decryptedMessage);
```

Hình 32. Đoạn mã giải mã toàn bộ message bằng AES-CBC

- Verify chữ ký SHA256withRSA
 - Dùng public key của client để xác minh signature có hợp lệ với decryptedMessage không.
 - Nếu hợp lệ: xác nhận message do client gửi.
 - Nếu không hợp lệ → VERIFICATION_FAILED.

```
// Bước 3: Verify signature
boolean isValid = cryptoServer.verifySignature(decryptedMessage,
    request.getSignature(),
    request.getPublicKey());
```

Hình 33. Đoạn mã xác thực chữ ký SHA256withRSA

- Scan subdomain khi xác thực thành công
 - Thực hiện nhiệm vụ chính:
 - Nếu verify thành công → Server gọi tới SubdomainScanner để quét tất cả các subdomain của huflit.edu.vn.
 - Danh sách subdomain được lấy từ file wordlist.

```
// Bước 5: Scan subdomain
System.out.println("3. Bắt đầu scan subdomain huflit.edu.vn...");
String scanResults = subdomainScanner.scanSubdomains("huflit.edu.vn");
response.setScanResults(scanResults);
System.out.println("    ✓ Hoàn thành scan subdomain");
```

Hình 34. Đoạn mã scan subdomain khi xác thực thành công

- Xử lý lỗi
 - Nếu lỗi xảy ra ở bất kỳ bước nào: Server sẽ không gửi kết quả scan, mà chỉ trả về "VERIFICATION_FAILED".

```
} catch (Exception e) {
    System.err.println("Lỗi khi xử lý request: " + e.getMessage());
    response.setResponse("VERIFICATION_FAILED");
}
```

Hình 35. Đoạn mã -xử lý lỗi

3.11. Class SubdomainScanner.java

- Đọc wordlist từ file resources
 - Load file wordlist từ thư mục resources/, xử lý các dòng hợp lệ và lưu vào danh sách.

```
private List<String> loadWordlist() throws Exception { 1 usage
    List<String> wordlist = new ArrayList<>();

    // Đọc từ resources folder
    System.out.println("    - Đọc wordlist từ resources: " + WORDLIST_FILE);

    // Sử dụng ClassLoader để đọc file từ resources
    try (BufferedReader reader = new BufferedReader(
        new InputStreamReader(
            getClass().getClassLoader().getResourceAsStream(WORDLIST_FILE))) {

        if (reader == null) {
            throw new Exception("File " + WORDLIST_FILE + " không tồn tại trong resources folder!");
        }

        String line;
        while ((line = reader.readLine()) != null) {
            line = line.trim().toLowerCase();
            if (!line.isEmpty() && !line.startsWith("#")) {
                wordlist.add(line);
            }
        }
    }
}
```

Hình 36. Đoạn mã - Đọc wordlist từ file resources

- Tạo ExecutorService để quét song song các subdomain
 - Tạo các task kiểm tra DNS cho mỗi subdomain, và submit vào thread pool.

```
// Sử dụng ThreadPoolExecutor để scan parallel
ExecutorService executor = Executors.newFixedThreadPool(MAX_THREADS);
List<Future<String>> futures = new ArrayList<>();

for (int i = 0; i < maxToTest; i++) {
    String subdomain = wordlist.get(i).trim() + "." + domain;

    Future<String> future = executor.submit(() -> {
        if (isSubdomainValid(subdomain)) {
            return subdomain;
        }
        return null;
    });

    futures.add(future);
}
```

Hình 37. Đoạn mã tạo ExecutorService để quét song song các subdomain

- Thu thập kết quả từ các Future
 - Lấy kết quả từ các luồng. Nếu resolve DNS thành công, subdomain sẽ được thêm vào danh sách kết quả.

```
for (Future<String> future : futures) {
    try {
        String result = future.get(TIMEOUT_MS, TimeUnit.MILLISECONDS);
        if (result != null) {
            foundSubdomains.add(result);
            System.out.println("    ✓ Tìm thấy: " + result);
        }

        completed++;
        if (completed % 100 == 0) {
            System.out.println("    - Đã hoàn thành: " + completed + "/" + maxToTest);
        }
    } catch (TimeoutException e) {
        future.cancel( mayInterruptIfRunning: true);
    } catch (Exception e) {
        // Ignore individual failures
    }
}
```

Hình 38. Đoạn mã thu thập kết quả từ các Future

- Hàm kiểm tra DNS hợp lệ
 - Kiểm tra xem subdomain có thực sự tồn tại không.
 - Resolve DNS của subdomain, nếu không lỗi thì subdomain tồn tại.

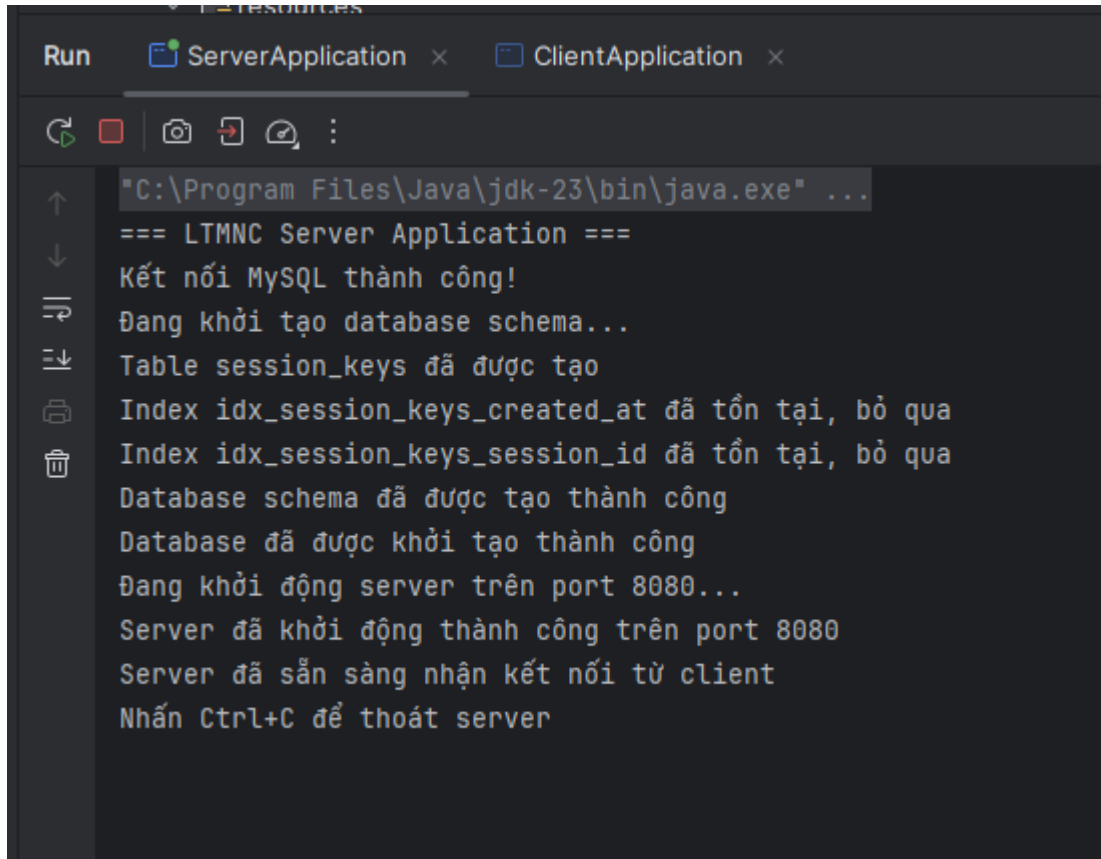
```
private boolean isSubdomainValid(String subdomain) { 1 usage
    try {
        // Thử resolve DNS
        InetAddress address = InetAddress.getByName(subdomain);
        return address != null;
    } catch (Exception e) {
        return false;
    }
}
```

Hình 39. Hàm kiểm tra DNS hợp lệ

CHƯƠNG IV. ĐÁNH GIÁ VÀ KẾT LUẬN

1. Kết quả thực hiện

Chạy file ServerApplication.java trước



```

Run  ServerApplication x ClientApplication x
" C:\Program Files\Java\jdk-23\bin\java.exe" ...
=== LTMNC Server Application ===
Kết nối MySQL thành công!
Đang khởi tạo database schema...
Table session_keys đã được tạo
Index idx_session_keys_created_at đã tồn tại, bỏ qua
Index idx_session_keys_session_id đã tồn tại, bỏ qua
Database schema đã được tạo thành công
Database đã được khởi tạo thành công
Đang khởi động server trên port 8080...
Server đã khởi động thành công trên port 8080
Server đã sẵn sàng nhận kết nối từ client
Nhấn Ctrl+C để thoát server
  
```

Hình 40. Giao diện console Server

Sau đó chạy đến file ClientApplication.java, nhập tin nhắn gửi đến Server và chờ xác thực

```

Run  ServerApplication x ClientApplication x
C:\Program Files\Java\jdk-23\bin\java.exe" ...
=== LTMNC Client Application ===
Đã load thành công các crypto keys
Kết nối đến server thành công!
Đã kết nối đến server: localhost:8080

Nhập message để gửi (hoặc 'quit' để thoát): hello
Đang gửi message: hello

=== Bắt đầu quá trình mã hóa ===
1. Tạo chữ ký số với SHA256withRSA...
   ✓ Chữ ký đã được tạo: safHHFoKKKy8VJsRV6TXCbK08wiinRCz2D4KicAibmTQziq6h4...
2. Tạo AES key và IV...
   ✓ AES key và IV đã được tạo
3. Mã hóa message bằng AES-CBC...
   ✓ Chữ ký đã được tạo: safHHFoKKKy8VJsRV6TXCbK08wiinRCz2D4KicAibmTQziq6h4...
4. Mã hóa AES key bằng RSA...
   ✓ AES key đã được mã hóa: Y+AvaC0kf+bMpqC0k1jYS66oUMFE6brD/6wB+X9rz6i7DnGJ0l...
5. Encode IV và Public Key...
   ✓ IV và Public Key đã được encode
=== Hoàn thành quá trình mã hóa ===

Đã gửi message được mã hóa đến server

```

Hình 41. Giao diện console Client

Phía server sẽ xác minh, nếu thành công sẽ bắt đầu scan subdomain

c

```

Run  ServerApplication x ClientApplication x
Nhấn Ctrl+C để thoát server
Server crypto keys đã được load thành công
Kết nối MySQL thành công!
Client đã kết nối: /127.0.0.1:59031

=== Nhận request từ Client ===
Client: /127.0.0.1:59031
1. Bắt đầu giải mã và verify...
  ✓ Đã lưu keys vào database với session ID: SESSION_1754061061170
  ✓ Đã lưu keys vào database
  - Bắt đầu giải mã message...
    ✓ Đã giải mã AES key
    ✓ Đã decode IV
    ✓ Đã giải mã message bằng AES-CBC
  ✓ Đã giải mã message: hello
  - Bắt đầu verify signature...
    ✓ Đã load client public key
    ✓ Signature verification thành công
    ✓ Signature verification THÀNH CÔNG
2. Đã trả về VERIFY
3. Bắt đầu scan subdomain huflit.edu.vn...
  - Đang tải wordlist từ file local...
  - Đọc wordlist từ resources: subdomains-top1million-110000.txt
  - Đã đọc 114438 subdomain từ resources
  - Đã tải 114438 từ từ wordlist
  - Sẽ test 1000 subdomain đầu tiên
  - Đang scan các subdomain...
    ✓ Tìm thấy: www.huflit.edu.vn
    ✓ Tìm thấy: mail.huflit.edu.vn
    ✓ Tìm thấy: autodiscover.huflit.edu.vn
    ✓ Tìm thấy: beta.huflit.edu.vn
    ✓ Tìm thấy: portal.huflit.edu.vn
    ✓ Tìm thấy: ads.huflit.edu.vn
    ✓ Tìm thấy: ipv4.huflit.edu.vn
    ✓ Tìm thấy: my.huflit.edu.vn
    ✓ Tìm thấy: forums.huflit.edu.vn
  - Đã hoàn thành: 100/1000
    ✓ Tìm thấy: en.huflit.edu.vn
    ✓ Tìm thấy: help.huflit.edu.vn
    ✓ Tìm thấy: ts.huflit.edu.vn
    ✓ Tìm thấy: lib.huflit.edu.vn
  - Đã hoàn thành: 200/1000
    ✓ Tìm thấy: lyncdiscover.huflit.edu.vn
    ✓ Tìm thấy: alumni.huflit.edu.vn
  
```

Hình 42. Server đã các mình thành công và bắt đầu scan domain

Sau khi scan hoàn tất sẽ thông báo về cho Client và in ra các subdomain hợp lệ

```

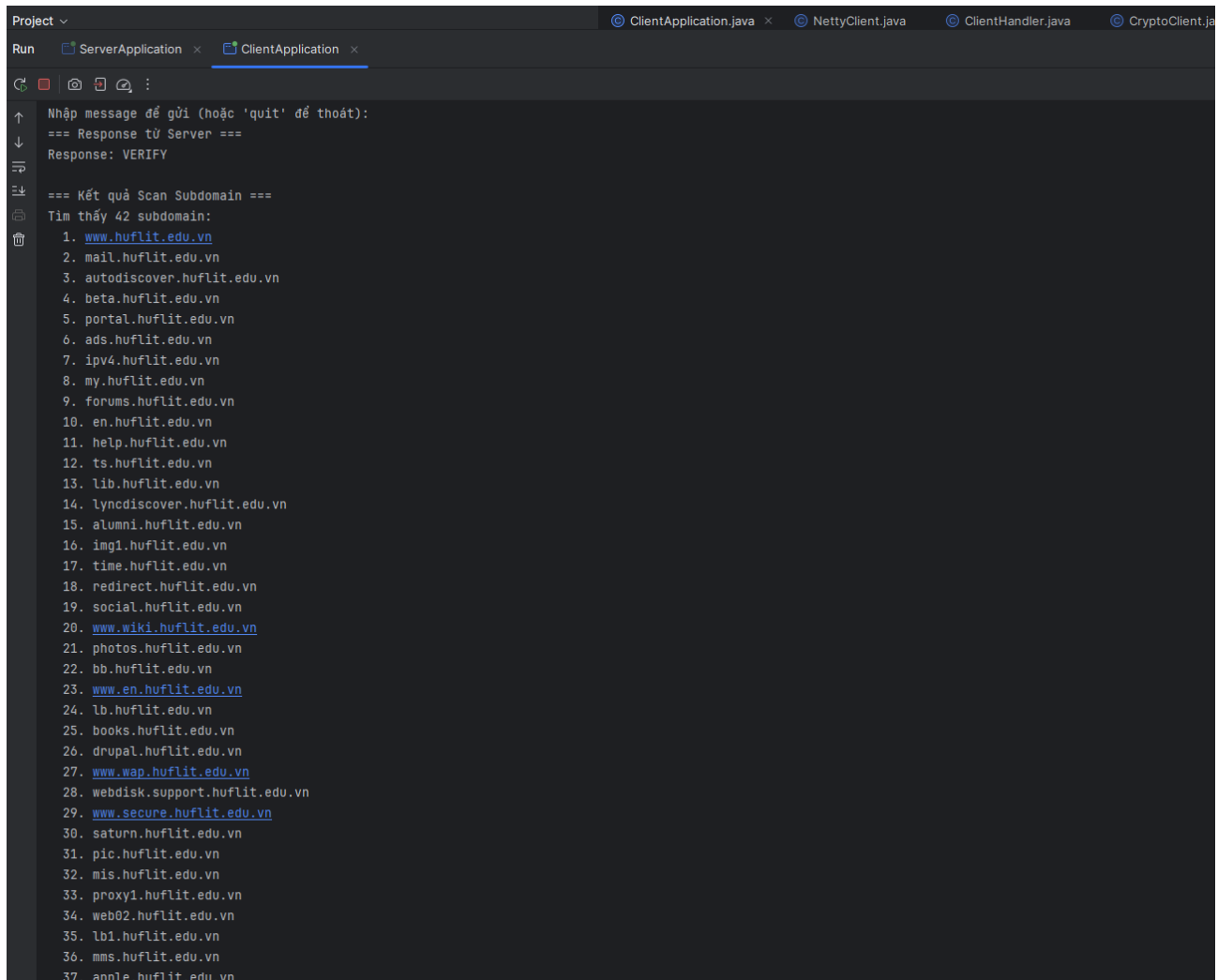
Project Alt+1
Run ServerApplication ClientApplication
ClientApplication.java NettyClient.java ClientHandler.java CryptoClient.java

- Tìm thấy: lib.huflit.edu.vn
- Đã hoàn thành: 200/1000
- Tìm thấy: lyncdiscover.huflit.edu.vn
- Tìm thấy: alumni.huflit.edu.vn
- Đã hoàn thành: 300/1000
- Tìm thấy: img1.huflit.edu.vn
- Tìm thấy: time.huflit.edu.vn
- Tìm thấy: redirect.huflit.edu.vn
- Tìm thấy: social.huflit.edu.vn
- Tìm thấy: www.wiki.huflit.edu.vn
- Tìm thấy: photos.huflit.edu.vn
- Tìm thấy: bb.huflit.edu.vn
- Đã hoàn thành: 400/1000
- Tìm thấy: www.en.huflit.edu.vn
- Đã hoàn thành: 500/1000
- Tìm thấy: lb.huflit.edu.vn
- Tìm thấy: books.huflit.edu.vn
- Tìm thấy: drupal.huflit.edu.vn
- Tìm thấy: www.wap.huflit.edu.vn
- Tìm thấy: webdisk.support.huflit.edu.vn
- Tìm thấy: www.secure.huflit.edu.vn
- Tìm thấy: saturn.huflit.edu.vn
- Đã hoàn thành: 600/1000
- Tìm thấy: pic.huflit.edu.vn
- Đã hoàn thành: 700/1000
- Tìm thấy: mis.huflit.edu.vn
- Tìm thấy: proxy1.huflit.edu.vn
- Tìm thấy: web02.huflit.edu.vn
- Tìm thấy: lb1.huflit.edu.vn
- Tìm thấy: mms.huflit.edu.vn
- Tìm thấy: apple.huflit.edu.vn
- Tìm thấy: sns.huflit.edu.vn
- Tìm thấy: classifieds.huflit.edu.vn
- Tìm thấy: repository.huflit.edu.vn
- Đã hoàn thành: 800/1000
- Tìm thấy: meeting.huflit.edu.vn
- Tìm thấy: twitter.huflit.edu.vn
- Đã hoàn thành: 900/1000
- Hoàn thành scan. Tìm thấy 42 subdomain hợp lệ
- Hoàn thành scan subdomain
Đã gửi response về client
=====

```

Hình 43. Server trả kết quả về cho Client

Client nhận được thông báo VERIFY và danh sách các subdomain hợp lệ



```

Project
Run
ServerApplication
ClientApplication
ClientApplication.java
NettyClient.java
ClientHandler.java
CryptoClient.java

Nhập message để gửi (hoặc 'quit' để thoát):
=== Response từ Server ===
Response: VERIFY

=== Kết quả Scan Subdomain ===
Tìm thấy 42 subdomain:
1. www.huflit.edu.vn
2. mail.huflit.edu.vn
3. autodiscover.huflit.edu.vn
4. beta.huflit.edu.vn
5. portal.huflit.edu.vn
6. ads.huflit.edu.vn
7. ipv4.huflit.edu.vn
8. my.huflit.edu.vn
9. forums.huflit.edu.vn
10. en.huflit.edu.vn
11. help.huflit.edu.vn
12. ts.huflit.edu.vn
13. lib.huflit.edu.vn
14. lyncdiscover.huflit.edu.vn
15. alumni.huflit.edu.vn
16. img1.huflit.edu.vn
17. time.huflit.edu.vn
18. redirect.huflit.edu.vn
19. social.huflit.edu.vn
20. www.wiki.huflit.edu.vn
21. photos.huflit.edu.vn
22. bb.huflit.edu.vn
23. www.en.huflit.edu.vn
24. lb.huflit.edu.vn
25. books.huflit.edu.vn
26. drupal.huflit.edu.vn
27. www.wap.huflit.edu.vn
28. webdisk.support.huflit.edu.vn
29. www.secure.huflit.edu.vn
30. saturn.huflit.edu.vn
31. pic.huflit.edu.vn
32. mis.huflit.edu.vn
33. proxy1.huflit.edu.vn
34. web02.huflit.edu.vn
35. lb1.huflit.edu.vn
36. mms.huflit.edu.vn
37. apple.huflit.edu.vn

```

Hình 44. Client nhận được thông báo xác thực thành công.



```

37. apple.huflit.edu.vn
38. sns.huflit.edu.vn
39. classifieds.huflit.edu.vn
40. repository.huflit.edu.vn
41. meeting.huflit.edu.vn
42. twitter.huflit.edu.vn
Tổng cộng: 42 subdomain được tìm thấy
=====

```

Hình 45. Danh sách subdomain Server trả về

2. Đánh giá hệ thống

Hệ thống đã được xây dựng đáp ứng đầy đủ các yêu cầu chức năng và phi chức năng được đề ra. Cơ chế xác thực chữ ký số giữa Client và Server được thực hiện chuẩn xác bằng thuật toán SHA256withRSA, kết hợp với phương pháp mã hóa AES-CBC giúp bảo đảm tính bí mật, toàn vẹn và xác thực nguồn gốc dữ liệu. Việc tích hợp thư viện Netty góp phần tối ưu hóa hiệu suất xử lý mạng, hỗ trợ truyền thông bất đồng bộ và đáp ứng hiệu quả các kết nối đồng thời.

Bên cạnh đó, chức năng quét subdomain hoạt động ổn định, chính xác dựa trên wordlist định sẵn. Các module xử lý mã hóa/giải mã, xác thực chữ ký, quản lý phiên làm việc và ghi log được tổ chức rõ ràng, tách biệt, thuận lợi cho việc mở rộng và bảo trì hệ thống trong tương lai.

Tuy vẫn còn tồn tại một số hạn chế, nhưng về tổng thể, hệ thống đã cho thấy khả năng ứng dụng thực tế cao, đặc biệt trong các tình huống yêu cầu xác thực dữ liệu và đánh giá bảo mật tên miền.

3. Kết luận

Đề tài đã hoàn thành mục tiêu xây dựng một hệ thống Client-Server bảo mật, ứng dụng các kỹ thuật xác thực bằng chữ ký số và mã hóa hiện đại để đảm bảo an toàn thông tin trong giao tiếp. Ngoài ra, việc tích hợp tính năng quét subdomain tự động góp phần hỗ trợ kiểm tra, đánh giá mức độ bảo mật hệ thống tên miền một cách chủ động và hiệu quả.

Thông qua quá trình thực hiện, nhóm đã:

- Nắm vững và áp dụng được các nguyên lý mã hóa và xác thực số trong môi trường mạng.
- Thành thạo việc triển khai giao tiếp bất đồng bộ hiệu năng cao bằng thư viện Netty.
- Rèn luyện kỹ năng tổ chức hệ thống đa tầng, xử lý đa luồng và quản lý socket nâng cao.
- Hiểu rõ hơn về quy trình quét subdomain và các kỹ thuật liên quan trong bảo mật hệ thống.

Đây là nền tảng quan trọng để tiếp tục nghiên cứu, phát triển các giải pháp bảo mật trong tương lai.

4. Hạn chế và phương hướng phát triển

4.1. Hạn chế

- Hệ thống hiện tại chỉ hỗ trợ quét subdomain cho một tên miền cố định (huflit.edu.vn) và wordlist tĩnh, chưa có cơ chế tùy chỉnh linh hoạt.
- Giao diện người dùng còn đơn giản, chỉ sử dụng console, gây khó khăn cho người dùng phổ thông.
- Thiếu cơ chế xử lý xác thực đa client đồng thời, hệ thống hiện mới kiểm thử với một client.
- Log hệ thống và cơ sở dữ liệu chưa đầy đủ, chưa có dashboard trực quan để giám sát, thống kê hoặc phân tích.
- Chưa tích hợp các cơ chế phòng chống tấn công như DoS hoặc giới hạn lượt xác thực sai nhằm bảo vệ server khỏi

4.1. Phương hướng phát triển

- Mở rộng hệ thống hỗ trợ xác thực và quét subdomain cho nhiều tên miền khác nhau, hỗ trợ tùy chọn wordlist do người dùng cung cấp.
- Phát triển giao diện người dùng đồ họa (GUI) hoặc ứng dụng web thân thiện, trực quan hơn cho người dùng không chuyên.
- Tích hợp hệ thống quản lý người dùng: phân quyền, theo dõi lịch sử phiên làm việc và xác thực.
- Tăng cường khả năng xử lý song song để hỗ trợ nhiều client truy cập đồng thời, đảm bảo hiệu suất cao hơn trong môi trường thực tế.
- Triển khai hệ thống bằng Docker hoặc các công nghệ container hóa để dễ dàng triển khai, mở rộng và bảo trì.
- Bổ sung các cơ chế bảo mật nâng cao như xác thực OAuth2, JWT token, hoặc mã hóa toàn bộ kênh truyền bằng SSL/TLS mặc định.

TÀI LIỆU THAM KHẢO

- [1] <https://netty.io/>
- [2] <https://gpcoder.com/3679-xay-dung-ung-dung-client-server-voi-socket-trong-java/>
- [3] <https://www.sentinelone.com/cybersecurity-101/cybersecurity/what-is-encryption/>
- [4] <https://topdev.vn/blog/xay-dung-ung-dung-client-server-voi-socket-trong-java/>