

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGOẠI NGỮ – TIN HỌC THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN MÔN HỌC: LẬP TRÌNH MẠNG NÂNG CAO
ĐỀ TÀI: THIẾT KẾ HỆ THỐNG TRUYỀN TIN AN TOÀN
KẾT HỢP KIỂM THỬ BẢO MẬT TÊN MIỀN PHỤ**

GIẢNG VIÊN HƯỚNG DẪN: ThS. Phan Gia Lượng

Thành viên nhóm:

Trần Thanh Tuấn Kiệt – 22DH114603

Tân Lai Hoàng – 22DH111164

Trần Xuân Thành – 22DH114735

Phạm Quốc Anh – 18DH110179

TP.HCM, tháng 8 năm 2025

LỜI CẢM ƠN

Trong quá trình thực hiện đồ án này, chúng em đã nhận được sự hỗ trợ và chỉ bảo tận tình của thầy Phan Gia Lượng. Chúng em muốn bày tỏ lòng biết ơn vì những kiến thức và kinh nghiệm thầy truyền đạt đã góp phần không ít giúp chúng em hoàn thành được báo cáo đúng hạn và hoàn chỉnh như những gì chúng em mong đợi. Chúng em cũng muốn gửi lời cảm ơn tới những người bạn cùng khóa đã phân nào hỗ trợ chúng em về mặt kiến thức trong quá trình thực hiện đồ án. Tuy nhiên, chúng em không thể khẳng định kết quả của chúng em là hoàn hảo và có thể phát sinh nhiều sai sót do những vấn đề không mong muốn xảy ra trong thời gian làm báo cáo. Chúng em xin sẵn sàng tiếp nhận những lời khuyên và sự chỉ bảo của thầy để rút kinh nghiệm cho những đồ án trong tương lai. Một lần nữa xin được gửi lời cảm ơn chân thành nhất đến thầy.

MỤC LỤC

LỜI CẢM ƠN	2
MỤC LỤC	3
DANH MỤC HÌNH ẢNH	4
DANH MỤC BẢNG BIỂU	6
CHƯƠNG I. GIỚI THIỆU	7
CHƯƠNG II. CƠ SỞ LÝ THUYẾT	8
1. Java là gì?	8
2. Thread là gì?	9
3. Socket là gì?	11
CHƯƠNG III. XÂY DỰNG VÀ TRIỂN KHAI	14
1. Xây dựng chương trình.	14
2. Triển khai demo.	28
CHƯƠNG IV. KẾT LUẬN	36
Tài liệu tham khảo	37
Phân chia công việc	38

DANH MỤC HÌNH ẢNH

Hình 1. Logo của ngôn ngữ lập trình Java	9
Hình 2. Một tiến trình với hai luồng thực thi	11
Hình 3. Chức năng của socket là kết nối giữa client và server thông qua TCP/IP và UDP để truyền và nhận giữ liệu qua Internet	13
Hình 4. Cây thư mục đồ án	14
Hình 5. Hàm main (1)	15
Hình 6. Hàm main (2)	15
Hình 7. Hàm handleRequest (1)	16
Hình 8. Hàm handleRequest (2)	16
Hình 9. Hàm handleRequest (3)	17
Hình 10. Hàm handleRequest (4)	17
Hình 11. Lớp CryptoUtils	18
Hình 12. Kết nối đến cơ sở dữ liệu MySQL	18
Hình 13. Hàm connect	18
Hình 14. Hàm initialize	19
Hình 15. Hàm saveKeyData	19
Hình 16. Lớp DomainExtractor	20
Hình 17. Hàm scan	20
Hình 18. Hàm isDomainAlive	21
Hình 19. Lớp ClientMain	21
Hình 20. Hàm loadPrivateKey	22
Hình 21. Hàm loadPublicKey	22
Hình 22. Hàm encodePublicKey	22
Hình 23. Hàm signMessage	23
Hình 24. Hàm generateAESKey	23
Hình 25. Hàm generateIV	23
Hình 26. Hàm encryptAES	24
Hình 27. Hàm encodeKey	24
Hình 28. Hàm encodeIV	24
Hình 29. Các thành phần giao diện	24
Hình 30. Hàm ShowUI (1)	25

Hình 31. Hàm ShowUI (2)	25
Hình 32. Hàm browseFile	26
Hình 33. Hàm handleSend (1).....	26
Hình 34. Hàm handleSend (2).....	27
Hình 35. Hàm handleSend (3).....	27
Hình 36. Tạo private key và public key	28
Hình 37. Tạo database để lưu trữ dữ liệu.....	28
Hình 38. keystore.jks.....	29
Hình 39. Khởi động Server	29
Hình 40. Client UI.....	30
Hình 41. Khởi động WireShark và quét các gói tin trong localhost	31
Hình 42. Người dùng nhập nội dung cần thiết vào các trường nhập dữ liệu.....	32
Hình 43. Server xử lý yêu cầu.....	33
Hình 44. Server quét domain	33
Hình 45. Thông tin đã được lưu vào cơ sở dữ liệu	34
Hình 46. Server trả kết quả về Client.....	34
Hình 47. Không có gói tin nào bị bắt và để lộ thông tin.....	34
Hình 48. WireShark bắt được toàn bộ thông tin mà Client gửi cho Server.....	35

DANH MỤC BẢNG BIỂU

Bảng 1. So sánh process và thread	10
Bảng 2. Các loại socket phổ biến	12

CHƯƠNG I. GIỚI THIỆU

Trong bối cảnh an ninh mạng ngày càng được quan tâm, việc đảm bảo tính toàn vẹn và xác thực của dữ liệu trong quá trình truyền thông giữa các hệ thống là vô cùng quan trọng. Các cuộc tấn công như giả mạo danh tính, nghe lén dữ liệu, hay khai thác các subdomain chưa được kiểm soát đang ngày càng phổ biến, đặc biệt đối với các hệ thống web.

Đề tài “**Thiết kế hệ thống truyền tin an toàn kết hợp kiểm thử bảo mật tên miền phụ**” nhằm xây dựng một hệ thống truyền thông an toàn giữa client và server thông qua việc kết hợp các kỹ thuật mã hoá và xác thực hiện đại như RSA, SHA-256 và AES theo mô hình mã hoá lai (hybrid encryption). Client sẽ ký số thông điệp bằng khóa riêng (private key), sau đó mã hoá bằng thuật toán AES-CBC, kèm theo khoá AES đã được mã hoá bằng RSA public key. Server sau khi nhận thông điệp sẽ giải mã, xác thực tính toàn vẹn và tính hợp lệ của thông điệp thông qua chữ ký số. Nếu xác thực thành công, server sẽ thực hiện quét tất cả các subdomain của hệ thống trang web (ví dụ: *huflit.edu.vn*) bằng cách sử dụng wordlist từ nguồn uy tín.

Thông qua đề tài này, chúng em có thể vận dụng kiến thức về mật mã học, giao thức xác thực, bảo mật hệ thống cũng như kỹ thuật trinh sát (reconnaissance) trong an ninh mạng để xây dựng một hệ thống thực tế, có thể áp dụng trong các quy trình kiểm thử và đánh giá bảo mật của tổ chức.

CHƯƠNG II. CƠ SỞ LÝ THUYẾT

1. Java là gì?

Java là một ngôn ngữ lập trình bậc cao, hướng đối tượng, được phát triển bởi James Gosling tại Sun Microsystems và ra mắt lần đầu vào năm 1995. Hiện nay, Java thuộc sở hữu của Oracle Corporation. Ngôn ngữ này nổi bật với khả năng "Write Once, Run Anywhere" (Viết một lần, chạy mọi nơi), cho phép mã nguồn Java chạy trên nhiều nền tảng khác nhau thông qua Java Virtual Machine (JVM).

a. Đặc điểm nổi bật của Java:

- **Đa nền tảng:** Nhờ JVM, mã Java có thể chạy trên nhiều hệ điều hành như Windows, macOS, Linux mà không cần chỉnh sửa.
- **Hướng đối tượng:** Java áp dụng mô hình lập trình hướng đối tượng, giúp tổ chức mã nguồn rõ ràng và dễ bảo trì.
- **Bảo mật cao:** Java được thiết kế với các tính năng bảo mật mạnh mẽ, giúp bảo vệ ứng dụng khỏi các mối đe dọa.
- **Quản lý bộ nhớ tự động:** Java sử dụng cơ chế thu gom rác (Garbage Collection) để tự động quản lý bộ nhớ, giảm thiểu rò rỉ bộ nhớ.

b. Ứng dụng của Java: Java được sử dụng rộng rãi trong nhiều lĩnh vực:

- **Phát triển ứng dụng web:** Java là nền tảng cho nhiều ứng dụng web quy mô lớn.
- **Ứng dụng di động:** Java là ngôn ngữ chính để phát triển ứng dụng Android.
- **Hệ thống doanh nghiệp:** Nhiều hệ thống quản lý doanh nghiệp và ngân hàng sử dụng Java cho backend.
- **Trí tuệ nhân tạo và học máy:** Java cung cấp nhiều thư viện hỗ trợ phát triển các ứng dụng AI và machine learning.
- **Internet of Things (IoT):** Java được sử dụng để lập trình các thiết bị IoT nhờ tính linh hoạt và khả năng chạy trên nhiều nền tảng.



Hình 1. Logo của ngôn ngữ lập trình Java

2. Thread là gì?

Trong lập trình và khoa học máy tính, **thread** (hay *luồng thực thi*) là đơn vị nhỏ nhất của chương trình mà hệ điều hành (hoặc JVM, nếu là Java) có thể quản lý và thực thi độc lập.

a. Khái niệm cơ bản:

- Một **thread** chứa **ngữ cảnh thực thi** bao gồm:
 - **Con trỏ chương trình** (program counter).
 - Tập **registers** (thanh ghi).
 - **Ngăn xếp (stack)** dành riêng cho thread.
- Khi chạy, hệ điều hành chuyển ngữ cảnh giữa các thread để mỗi thread có thể "chạy xen kẽ" hoặc cùng lúc (nếu nhiều lõi/Cores).

b. Luồng và tiến trình:

Yếu tố	Tiến trình (Process)	Luồng (Thread)
Không gian địa chỉ	Riêng biệt	Dùng chung với toàn bộ tiến trình
Tài nguyên	Mã, dữ liệu, file, tín hiệu...	Chia sẻ tài nguyên của tiến trình

Ngữ cảnh	PCB (Process Control Block)	TCB (Thread Control Block): ID, PC, ghi, stack...
Trọng lượng hệ thống	Nặng	Nhẹ (lightweight)

Bảng 1. So sánh process và thread

c. Tại sao dùng thread?

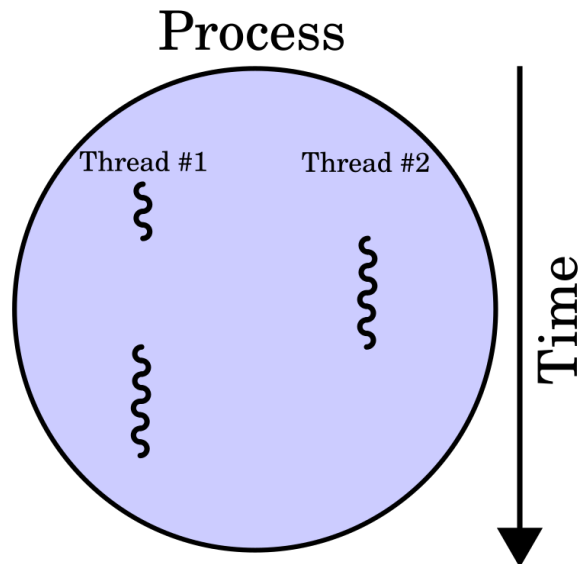
- **Tăng tốc và đa nhiệm:** các tác vụ có thể chạy song song hoặc song song giả (trên 1 lõi).
- **Chia sẻ dữ liệu dễ dàng:** threads dùng chung bộ nhớ, thuận tiện nhưng nguy cơ lỗi đồng bộ (race condition).
- **Minh họa:** Ví dụ ứng dụng soạn thảo văn bản – một thread xử lý nhập liệu, một thread xử lý định dạng hoặc tự động lưu, mà không gây đóng băng giao diện.

d. Các loại thread:

- **User-level Thread** (do ứng dụng quản lý): nhẹ, quản lý nhanh nhưng nếu block, toàn bộ process sẽ bị block.
- **Kernel-level Thread** (do hệ điều hành quản lý): được phân bổ trên nhiều lõi, nhưng chi phí context switch cao hơn.

e. Thread trong Java:

- JVM chạy như một tiến trình; thread được ánh xạ tới các thread hệ điều hành (thường).
- Mỗi thread Java là thể hiện của lớp *Thread*; có thể tạo thread bằng cách:
 - Kế thừa *Thread* và override *run()*, sau đó gọi *start()*.
 - Triển khai *Runnable* và truyền vào *Thread*.
- Threads chia sẻ bộ nhớ và file, nên việc tương tác nhanh nhưng dễ xảy ra lỗi nếu không đồng bộ đúng cách.



Hình 2. Một tiến trình với hai luồng thực thi

3. Socket là gì?

Trong lập trình và mạng máy tính, **socket** là một **điểm cuối (endpoint)** của kênh truyền thông hai chiều giữa các chương trình, có thể chạy trên cùng một máy hoặc trên các máy khác nhau.

a. Cấu trúc và định danh:

- Một socket được xác định bởi:
 - Địa chỉ IP.
 - Số cổng (port).
 - Giao thức truyền tải (TCP hoặc UDP).
- Trên hệ điều hành Unix, mỗi socket còn là một **file descriptor** – một số nguyên đại diện cho kết nối, giống như thao tác đọc/ghi file.

b. Các loại socket phổ biến:

Loại socket	Giao thức	Đặc trưng chính
Stream socket	TCP (kết nối)	Kênh dữ liệu liên tục, đảm bảo, có thứ tự
Datagram socket	UDP (không kết nối)	Gửi theo gói, không đảm bảo thứ tự, nhanh

Unix Domain Socket	Nội bộ máy tính	Giao tiếp giữa các tiến trình trên cùng hệ thống, API giống network socket
Raw socket	TCP/IP thấp cấp	Cung cấp truy cập mức thấp, dùng cho mạng theo dõi, phần mềm tùy chỉnh

Bảng 2. Các loại socket phổ biến

c. Các thức hoạt động (theo mô hình client-server):

- **Server:**

- Tạo socket (*socket()*).
- Gắn vào địa chỉ IP + port (*bind()*).
- Chờ kết nối (*listen()*).
- Chấp nhận kết nối (*accept()*).
- Nhận/gửi dữ liệu (*read()/write()* hoặc *recv()/send()*).
- Đóng socket (*close()*).

- **Client:**

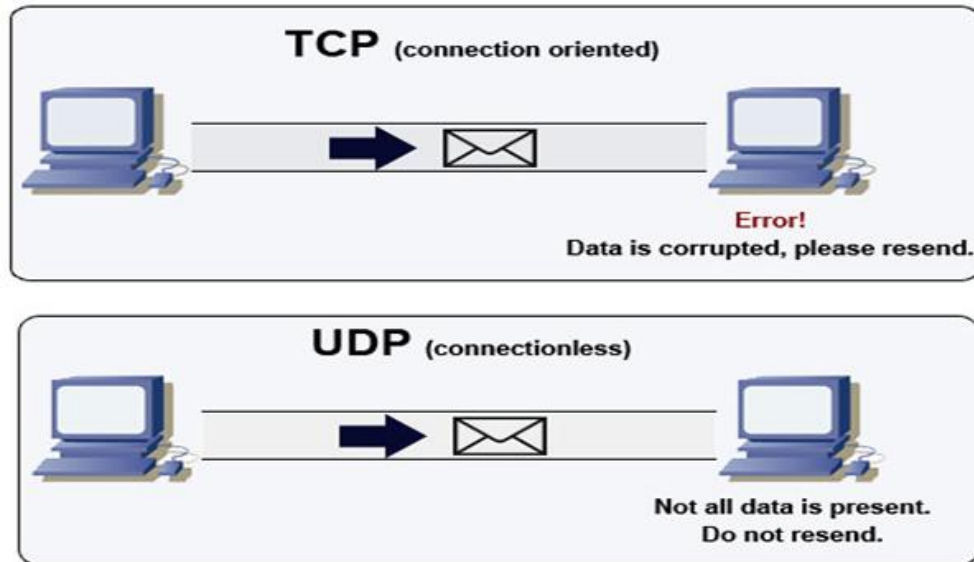
- Tạo socket.
- Kết nối tới server (*connect()*).
- Gửi/nhận dữ liệu.
- Đóng socket.

d. So sánh với các khái niệm khác:

- **Pipe** thường dùng trong cùng tiến trình hoặc tiến trình con, không qua mạng; socket thì vượt mạng và đa dạng hơn.
- **Socket vs Port:** Port là thành phần trong socket – socket = IP + port + giao thức.

e. Ứng dụng thực tế:

- Dùng trong các ứng dụng client-server: chat, web, game, ...
- Có thể dùng trong cùng máy để kết nối tiến trình với Unix Domain Socket.
- Sử dụng raw socket để viết công cụ giám sát mạng như ping, traceroute.

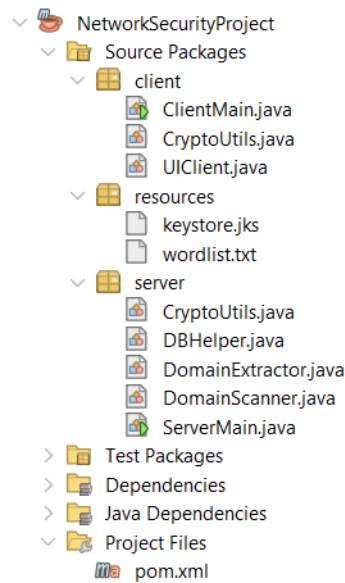


Hình 3. Chức năng của socket là kết nối giữa client và server thông qua TCP/IP và UDP để truyền và nhận giữ liệu qua Internet

CHƯƠNG III. XÂY DỰNG VÀ TRIỂN KHAI

1. Xây dựng chương trình.

Cây thư mục:



Hình 4. Cây thư mục đồ án

a. Server:

- **ServerMain.java:**

- **Hàm *main*:** Tạo và khởi chạy một server HTTPS an toàn, có khả năng xử lý nhiều yêu cầu đồng thời, và đảm bảo rằng các kết nối giữa client và server được mã hóa bằng SSL/TLS:

```

public static void main(String[] args) throws Exception {
    DBHelper.initialize(); // Khởi tạo kết nối database

    // Tạo HTTPS server chạy ở port 8443
    HttpsServer server = HttpsServer.create(new InetSocketAddress(8443), 0);
    System.out.println("Server đang chạy tại http://localhost:8443 ...");

    // Tạo context xử lý HTTP request
    server.createContext("/", ServerMain::handleRequest);
    server.setExecutor(Executors.newFixedThreadPool(10)); // Thread pool xử lý song song

    // Load keystore chứa chứng chỉ và khóa bí mật
    char[] password = "Tanlaihoang2922004.".toCharArray(); // mật khẩu của keystore
    KeyStore ks = KeyStore.getInstance("JKS");
    FileInputStream fis = new FileInputStream("src\\main\\java\\resources\\keystore.jks");
    ks.load(fis, password);

    // Tạo KeyManager từ keystore
    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
    kmf.init(ks, password);

    // Tạo TrustManager từ keystore
    TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
    tmf.init(ks);

    // Tạo SSLContext để mã hóa HTTPS
    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

```

Hình 5. Hàm main (1)

```

// Cấu hình SSL cho server
server.setHttpsConfigurator(new HttpsConfigurator(sslContext) {
    @Override
    public void configure(HttpsParameters params) {
        try {
            SSLContext context = getSSLContext();
            SSLEngine engine = context.createSSLEngine();
            params.setNeedClientAuth(false); // không yêu cầu client chứng thực
            params.setCipherSuites(engine.getEnabledCipherSuites());
            params.setProtocols(engine.getEnabledProtocols());
            params.setSSLParameters(context.getDefaultSSLParameters());
        } catch (Exception ex) {
            System.err.println("Lỗi cấu hình SSL: " + ex.getMessage());
        }
    }
});

server.start(); // Khởi động server
}

```

Hình 6. Hàm main (2)

- **Hàm *handleRequest***: Xác thực chữ ký số, giải mã public key, trích xuất domain, quét subdomain, và trả kết quả về client dưới dạng JSON:

```

private static void handleRequest(HttpExchange exchange) throws IOException {
    // Chỉ xử lý POST
    if (!exchange.getRequestMethod().equalsIgnoreCase("POST")) {
        exchange.sendResponseHeaders(405, -1); // 405: Method Not Allowed
        return;
    }

    // Đọc body từ request
    InputStream is = exchange.getRequestBody();
    String body = new String(is.readAllBytes(), StandardCharsets.UTF_8);
    System.out.println("Nhan request: " + body);

    // Phân tích JSON
    Gson gson = new Gson();
    Type mapType = new TypeToken<Map<String, String>>() {
    }.getType();
    Map<String, String> json = gson.fromJson(body, mapType);

    // Lấy các trường dữ liệu
    String rawMessage = json.get("raw_message");
    String signedMessage = json.get("signed_message");
    String encryptedPublicKey = json.get("encrypted_public_key");
    String aesKeyBase64 = json.get("key");
    String ivBase64 = json.get("iv");

    String response = null;
}

```

Hình 7. Hàm handleRequest (1)

```

try {
    // Giải mã AES key và IV
    byte[] keyBytes = Base64.getDecoder().decode(aesKeyBase64);
    SecretKey aesKey = new SecretKeySpec(keyBytes, "AES");

    byte[] ivBytes = Base64.getDecoder().decode(ivBase64);
    IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

    // Lấy IP client
    String clientIp = exchange.getRemoteAddress().getAddress().getHostAddress();

    // Nếu có private key mã hóa (hiện tại không dùng), giải mã thử
    String encryptedPrivateKey = json.containsKey("encrypted_private_key") ? json.get("encrypted_private_key") : null;
    if (encryptedPrivateKey != null) {
        CryptoUtils.decryptAES(encryptedPrivateKey, aesKey, ivSpec);
    }

    // Giải mã public key
    String publicKeyPEM = CryptoUtils.decryptAES(encryptedPublicKey, aesKey, ivSpec);

    // Xác thực chữ ký số
    boolean verified = CryptoUtils.verifySignature(rawMessage, signedMessage, publicKeyPEM);
}

```

Hình 8. Hàm handleRequest (2)


```

if (verified) {
    System.out.println("Xác thực thành công.");

    // Trích xuất domain từ raw_message
    List<String> extractedDomains = DomainExtractor.extractDomains(rawMessage);

    if (!extractedDomains.isEmpty()) {
        String firstDomain = extractedDomains.get(0);
        DBHelper.saveKeyData(clientIp, firstDomain, publicKeyPEM, aesKeyBase64, ivBase64);

        List<String> allSubdomains = new ArrayList<>();
        for (String domain : extractedDomains) {
            System.out.println("Scanning domain: " + domain);
            List<String> subdomains = DomainScanner.scan(domain, 20); // scan giới hạn 20 subdomain
            allSubdomains.addAll(subdomains);
        }

        response = gson.toJson(allSubdomains); // trả về danh sách subdomain
        System.out.println("Scan complete. Tổng cộng " + allSubdomains.size() + " subdomains.");
    } else {
        // Không có domain hợp lệ trong message
        System.out.println("Không tìm thấy domain hợp lệ trong chuỗi raw_message.");
        DBHelper.saveKeyData(clientIp, null, publicKeyPEM, aesKeyBase64, ivBase64);
        response = "INVALID_DOMAIN_FORMAT";
    }
} else {
    response = "VERIFICATION_FAILED"; // <-- THÊM DÒNG NÀY LÀ BẮT BUỘC
}

} catch (Exception ex) {
    response = "SERVER_ERROR: " + ex.getMessage();
}

```

Hình 9. Hàm *handleRequest* (3)

```

// Trả response về cho client
byte[] respBytes = response.getBytes(StandardCharsets.UTF_8);
exchange.getResponseHeaders().set("Content-Type", "application/json");
exchange.sendResponseHeaders(200, respBytes.length);
try (OutputStream os = exchange.getResponseBody()) {
    os.write(respBytes);
}
}

```

Hình 10. Hàm *handleRequest* (4)

- **CryptoUtils.java:** Giải mã dữ liệu AES-CBC từ chuỗi base64 (*decryptAES*); Xác thực chữ ký số với SHA256withRSA (*verifySignature*); Tải khóa công khai từ chuỗi PEM base64 (*loadPublicKeyFromPEM*);

```

public class CryptoUtils {

    // Giải mã chuỗi đã mã hóa bằng AES CBC, trả về chuỗi gốc
    public static String decryptAES(String encryptedBase64, SecretKey key, IvParameterSpec ivSpec) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding"); // Khởi tạo AES ở chế độ CBC
        cipher.init(Cipher.DECRYPT_MODE, key, ivSpec); // Gán key và IV
        byte[] decoded = Base64.getDecoder().decode(encryptedBase64); // Giải mã Base64 trước
        byte[] decrypted = cipher.doFinal(decoded); // Giải mã AES
        return new String(decrypted, StandardCharsets.UTF_8); // Trả về chuỗi kết quả
    }

    // Xác thực chữ ký số dùng thuật toán SHA256withRSA
    public static boolean verifySignature(String rawMessage, String base64Signature, String publicKeyPEM) throws Exception {
        PublicKey publicKey = loadPublicKeyFromPEM(publicKeyPEM); // Tạo đối tượng PublicKey từ PEM base64

        Signature signature = Signature.getInstance("SHA256withRSA"); // Sử dụng thuật toán ký RSA với SHA256
        signature.initVerify(publicKey); // Gán khóa để xác thực
        signature.update(rawMessage.getBytes(StandardCharsets.UTF_8)); // Cập nhật dữ liệu gốc để xác thực

        byte[] signedBytes = Base64.getDecoder().decode(base64Signature); // Giải mã chữ ký từ base64
        return signature.verify(signedBytes); // Trả về true nếu xác thực thành công
    }

    // Chuyển chuỗi PEM (base64) sang đối tượng PublicKey
    private static PublicKey loadPublicKeyFromPEM(String base64Key) throws Exception {
        byte[] decoded = Base64.getDecoder().decode(base64Key); // Giải mã base64
        X509EncodedKeySpec spec = new X509EncodedKeySpec(decoded); // Tạo định dạng khóa theo chuẩn X509
        KeyFactory kf = KeyFactory.getInstance("RSA"); // Lấy factory cho RSA
        return kf.generatePublic(spec); // Tạo đối tượng PublicKey từ spec
    }
}

```

Hình 11. Lớp CryptoUtils

- **DBHelper.java:**

- Các hằng số cấu hình cho việc kết nối đến cơ sở dữ liệu MySQL từ Java thông qua JDBC:

```

// Thông tin kết nối đến MySQL database tên là secure_network
private static final String URL = "jdbc:mysql://localhost:3306/secure_network?useSSL=false&serverTimezone=UTC";
private static final String USER = "root"; // Tên người dùng cơ sở dữ liệu
private static final String PASSWORD = "@Admin1234"; // Mật khẩu của người dùng cơ sở dữ liệu

```

Hình 12. Kết nối đến cơ sở dữ liệu MySQL

- **Hàm connect:** Dùng để khởi tạo và trả về kết nối đến MySQL:

```

public static Connection connect() throws SQLException {
    // Tạo kết nối đến cơ sở dữ liệu và trả về đối tượng Connection
    return DriverManager.getConnection(URL, USER, PASSWORD);
}

```

Hình 13. Hàm connect

- **Hàm *initialize***: Hiển thị thông báo ra màn hình console rằng kết nối MySQL đã thành công:

```
public static void initialize() {
    // Phương thức dùng để thông báo khi kết nối thành công
    System.out.println("Đã kết nối MySQL thành công.");
}
```

Hình 14. Hàm initialize

- **Hàm *saveKeyData***: Là một thành phần backend quan trọng giúp lưu lại thông tin các client đã gửi yêu cầu:

```
public static void saveKeyData(String clientIp, String result, String publicKey, String aesKey, String iv) {
    // Phương thức dùng để lưu thông tin khóa vào bảng key_storage

    String sql = "INSERT INTO key_storage (client_ip, result, public_key, aes_key, iv) VALUES (?, ?, ?, ?, ?)";
    // Câu lệnh SQL sử dụng PreparedStatement để tránh SQL injection

    try {
        Connection conn = connect(); // Tạo kết nối đến CSDL
        PreparedStatement stmt = conn.prepareStatement(sql) // Chuẩn bị câu lệnh SQL
    } {
        // Gán giá trị cho các dấu ? trong câu lệnh SQL
        stmt.setString(1, clientIp); // Gán địa chỉ IP của client
        stmt.setString(2, result); // Gán domain nếu hợp lệ, hoặc null nếu không
        stmt.setString(3, publicKey); // Gán khóa RSA public
        stmt.setString(4, aesKey); // Gán khóa AES đã mã hóa
        stmt.setString(5, iv); // Gán giá trị IV dùng cho AES

        stmt.executeUpdate(); // Thực thi câu lệnh INSERT để lưu vào DB
        System.out.println("Dữ liệu đã lưu vào MySQL."); // Thông báo lưu thành công
    } catch (SQLException e) {
        // Bắt lỗi nếu có vấn đề khi lưu vào DB
        System.err.println("Lỗi khi lưu vào DB: " + e.getMessage());
    }
}
```

Hình 15. Hàm saveKeyData

- **DomainExtractor.java**: Trích xuất các domain từ một đoạn văn bản đầu vào.

```

public class DomainExtractor {

    // Phương thức để trích xuất các domain từ đoạn văn bản đầu vào
    public static List<String> extractDomains(String text) {
        List<String> domains = new ArrayList<>(); // Tạo danh sách để chứa các domain tìm được

        // Regex để khớp với domain, ví dụ: youtube.com, sub.domain.co.uk
        String domainRegex = "\\b(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,}\\b";

        Pattern pattern = Pattern.compile(domainRegex); // Biên dịch biểu thức chính quy thành Pattern
        Matcher matcher = pattern.matcher(text); // Ảnh xạ Pattern với văn bản đầu vào để tìm các khớp

        // Duyệt từng kết quả tìm được
        while (matcher.find()) {
            domains.add(matcher.group(1)); // Lấy domain khớp và thêm vào danh sách
        }

        return domains; // Trả về danh sách các domain đã tìm thấy
    }
}

```

Hình 16. Lớp DomainExtractor

- **DomainScanner.java:**

- **Hàm scan:** quét và phát hiện các subdomain đang hoạt động của một tên miền gốc (*rootDomain*):

```

// Quét tối đa maxCount subdomains từ wordlist
public static List<String> scan(String rootDomain, int maxCount) {
    List<String> found = new ArrayList<>(); // Danh sách lưu subdomain tìm thấy

    try {
        // Tải wordlist từ GitHub chứa danh sách subdomain phổ biến
        URL wordlistURL = new URL("https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/DNS/subdomains-top1million-110000.txt");

        // Đọc nội dung wordlist qua stream
        BufferedReader reader = new BufferedReader(new InputStreamReader(wordlistURL.openStream()));

        String line;
        int count = 0;

        // Đọc từng dòng trong wordlist, tối đa maxCount dòng
        while ((line = reader.readLine()) != null && count < maxCount) {
            String subdomain = line.trim() + "." + rootDomain; // Gộp subdomain với domain chính

            // Kiểm tra xem subdomain có phản hồi HTTP hay không
            if (isDomainAlive("http://" + subdomain)) {
                System.out.println("Tìm thấy: " + subdomain); // Nếu tồn tại, in ra và thêm vào danh sách
                found.add(subdomain);
            } else {
                System.out.println("Không tồn tại: " + subdomain); // Nếu không tồn tại, in thông báo
            }
            count++; // Tăng bộ đếm để giới hạn số lượng kiểm tra
        }

    } catch (IOException e) {
        // Nếu có lỗi khi tải wordlist hoặc kết nối mạng, in lỗi
        System.err.println("Lỗi khi tải wordlist: " + e.getMessage());
    }

    return found; // Trả về danh sách các subdomain tồn tại
}

```

Hình 17. Hàm scan

- **Hàm *isDomainAlive*:** Kiểm tra xem một domain có phản hồi HTTP hợp lệ không:

```
// Hàm kiểm tra domain có "sống" hay không bằng cách gửi HTTP GET
public static boolean isDomainAlive(String urlStr) {
    try {
        URL url = new URL(urlStr); // Tạo URL từ chuỗi
        HttpURLConnection conn = (HttpURLConnection) url.openConnection(); // Mở kết nối HTTP

        conn.setConnectTimeout(2000); // Thiết lập timeout kết nối là 2 giây
        conn.setReadTimeout(2000); // Thiết lập timeout đọc dữ liệu là 2 giây
        conn.setRequestMethod("GET"); // Gửi yêu cầu GET

        int code = conn.getResponseCode(); // Lấy mã phản hồi HTTP

        return code == 200 || code == 301 || code == 302;

    } catch (IOException e) {
        // Nếu không kết nối được, trả về false
        return false;
    }
}
```

Hình 18. Hàm isDomainAlive

b. Client:

- **ClientMain.java:** Khởi chạy ứng dụng client với giao diện đồ họa Swing, tách biệt logic xử lý và giao diện theo mô hình MVC hoặc tương tự:

```
public class ClientMain {

    public static void main(String[] args) {
        // Gọi luồng giao diện đồ họa (GUI) của Java để tạo giao diện người dùng
        javax.swing.SwingUtilities.invokeLater(() -> {
            // Tạo đối tượng UIClient và hiển thị giao diện
            new UIClient().showUI();
        });
    }
}
```

Hình 19. Lớp ClientMain

- **CryptoUtils.java:**
 - **Hàm *loadPrivateKey*:** Giúp đọc và khôi phục khóa riêng RSA từ file .pem để sử dụng cho việc giải mã, ký số, hoặc xác thực trong ứng dụng bảo mật:

```
// Tải private key từ file PEM (định dạng PKCS#8)
public static PrivateKey loadPrivateKey(String filename) throws Exception {
    String key = new String(Files.readAllBytes(Paths.get(filename))) // Đọc toàn bộ file thành chuỗi
        .replace("-----BEGIN PRIVATE KEY-----", "") // Bỏ phần header PEM
        .replace("-----END PRIVATE KEY-----", "") // Bỏ phần footer PEM
        .replaceAll("\\s", ""); // Bỏ khoảng trắng, xuống dòng

    byte[] keyBytes = Base64.getDecoder().decode(key); // Giải mã base64
    PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes); // Tạo key spec PKCS#8
    KeyFactory kf = KeyFactory.getInstance("RSA"); // Sử dụng thuật toán RSA
    return kf.generatePrivate(spec); // Trả về đối tượng PrivateKey
}
```

Hình 20. Hàm loadPrivateKey

- **Hàm loadPublicKey:** Giúp đọc và tạo khóa công khai RSA từ file .pem, hỗ trợ cho các hoạt động mã hóa, kiểm tra chữ ký số, xác thực, ...:

```
// Tải public key từ file PEM (định dạng X.509)
public static PublicKey loadPublicKey(String filename) throws Exception {
    String key = new String(Files.readAllBytes(Paths.get(filename)))
        .replace("-----BEGIN PUBLIC KEY-----", "")
        .replace("-----END PUBLIC KEY-----", "")
        .replaceAll("\\s", "");

    byte[] keyBytes = Base64.getDecoder().decode(key);
    X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes); // X.509 dùng cho public key
    KeyFactory kf = KeyFactory.getInstance("RSA");
    return kf.generatePublic(spec);
}
```

Hình 21. Hàm loadPublicKey

- **Hàm encodePublicKey:** Chuyển PublicKey thành chuỗi, tiện để gửi qua mạng, ghi file, hoặc hiển thị:

```
// Chuyển public key sang chuỗi base64 (dùng để gửi đi)
public static String encodePublicKey(PublicKey publicKey) {
    byte[] encoded = publicKey.getEncoded(); // Lấy byte mảng
    return Base64.getEncoder().encodeToString(encoded); // Mã hóa base64
}
```

Hình 22. Hàm encodePublicKey

- **Hàm *signMessage***: dùng PrivateKey để ký một thông điệp, đảm bảo tính xác thực và toàn vẹn khi gửi sang Server hoặc Client khác:

```
// Ký tin nhắn bằng SHA256withRSA
public static String signMessage(String rawMessage, PrivateKey privateKey) throws Exception {
    Signature signature = Signature.getInstance("SHA256withRSA"); // Tạo đối tượng Signature
    signature.initSign(privateKey); // Khởi tạo với private key
    signature.update(rawMessage.getBytes(StandardCharsets.UTF_8)); // Cập nhật dữ liệu cần ký
    byte[] signedBytes = signature.sign(); // Ký và lấy kết quả
    return Base64.getEncoder().encodeToString(signedBytes); // Mã hóa base64 để gửi
}
```

Hình 23. Hàm *signMessage*

- **Hàm *generateAESKey***: Sinh ra khóa AES ngẫu nhiên, dùng cho mã hóa/giải mã dữ liệu bằng thuật toán AES:

```
// Sinh khóa AES 256-bit
public static SecretKey generateAESKey() throws Exception {
    KeyGenerator keyGen = KeyGenerator.getInstance("AES"); // Lấy generator AES
    keyGen.init(256); // Đặt độ dài khóa là 256-bit (cần JCE Unlimited nếu máy không hỗ trợ thì đổi thành 128)
    return keyGen.generateKey(); // Tạo khóa
}
```

Hình 24. Hàm *generateAESKey*

- **Hàm *generateIV***: Sinh ra IV ngẫu nhiên dùng cho các chế độ mã hóa đối xứng như AES-CBC để tăng tính bảo mật:

```
// Sinh IV ngẫu nhiên (16 byte cho AES CBC)
public static byte[] generateIV() {
    byte[] iv = new byte[16]; // 16 byte = 128-bit
    new SecureRandom().nextBytes(iv); // Sinh ngẫu nhiên
    return iv;
}
```

Hình 25. Hàm *generateIV*

- **Hàm *encryptAES***: Thực hiện mã hóa đối xứng dữ liệu bằng AES-CBC, và trả về dữ liệu đã mã hóa ở dạng Base64:

```
// Mã hóa dữ liệu bằng AES (CBC mode, có padding)
public static String encryptAES(String data, SecretKey key, byte[] ivBytes) throws Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding"); // AES CBC với padding
    IvParameterSpec iv = new IvParameterSpec(ivBytes);           // IV truyền vào
    cipher.init(Cipher.ENCRYPT_MODE, key, iv);                     // Chế độ mã hóa
    byte[] encrypted = cipher.doFinal(data.getBytes(StandardCharsets.UTF_8)); // Mã hóa dữ liệu
    return Base64.getEncoder().encodeToString(encrypted);         // Trả về dạng base64
}
```

Hình 26. Hàm encryptAES

- **Hàm encodeKey:** Dùng để biểu diễn khóa AES dưới dạng chuỗi Base64 – tiện cho việc gửi qua mạng hoặc lưu vào file JSON:

```
// Mã hóa (encode) khóa AES thành base64 để dễ gửi/nhận qua JSON
public static String encodeKey(SecretKey key) {
    return Base64.getEncoder().encodeToString(key.getEncoded());
}
```

Hình 27. Hàm encodeKey

- **Hàm encodeIV:** Giúp biểu diễn IV dưới dạng chuỗi Base64, tiện cho việc gửi kèm dữ liệu được mã hóa:

```
// Mã hóa IV thành base64
public static String encodeIV(byte[] iv) {
    return Base64.getEncoder().encodeToString(iv);
}
```

Hình 28. Hàm encodeIV

- **UIClient.java:**

- Các thành phần giao diện giúp người dùng thao tác với khóa và dữ liệu trong ứng dụng Client:

```
private JFrame frame; // Cửa sổ chính
private JTextField privateKeyField; // Ô nhập đường dẫn khóa riêng
private JTextField publicKeyField; // Ô nhập đường dẫn khóa công khai
private JTextArea messageArea; // Ô nhập nội dung tin nhắn
private JTextArea resultArea; // Ô hiển thị phản hồi từ server
```

Hình 29. Các thành phần giao diện

- **Hàm *ShowUI***: Hiển thị giao diện cho người dùng nhập thông tin cần thiết (private key, public key, nội dung tin nhắn) và gửi đến server:

```
public void showUI() {
    frame = new JFrame("Client UI - Gửi tin nhắn an toàn"); // Tạo cửa sổ
    frame.setSize(700, 600); // Kích thước cửa sổ
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Tắt chương trình khi đóng

    JPanel panel = new JPanel(new BorderLayout(10, 10)); // Panel chính, layout Border

    // Panel nhập liệu (trên cùng)
    JPanel inputPanel = new JPanel(new GridLayout(6, 1)); // Grid layout 6 hàng

    // Dòng nhập private key
    JPanel privateKeyPanel = new JPanel(new BorderLayout());
    privateKeyField = new JTextField(); // Trường nhập đường dẫn
    JButton browsePrivateBtn = new JButton("Browse..."); // Nút chọn file
    browsePrivateBtn.addActionListener(e -> browseFile(privateKeyField)); // Gắn sự kiện
    privateKeyPanel.add(privateKeyField, BorderLayout.CENTER);
    privateKeyPanel.add(browsePrivateBtn, BorderLayout.EAST);

    // Dòng nhập public key
    JPanel publicKeyPanel = new JPanel(new BorderLayout());
    publicKeyField = new JTextField();
    JButton browsePublicBtn = new JButton("Browse...");
    browsePublicBtn.addActionListener(e -> browseFile(publicKeyField));
    publicKeyPanel.add(publicKeyField, BorderLayout.CENTER);
    publicKeyPanel.add(browsePublicBtn, BorderLayout.EAST);

    messageArea = new JTextArea("Nhập message tại đây...", 3, 50); // Nhập tin nhắn
}
```

Hình 30. Hàm ShowUI (1)

```
// Thêm thành phần vào inputPanel
inputPanel.add(new JLabel("Đường dẫn private key (.pem):"));
inputPanel.add(privateKeyPanel);
inputPanel.add(new JLabel("Đường dẫn public key (.pem):"));
inputPanel.add(publicKeyPanel);
inputPanel.add(new JLabel("Message:"));
inputPanel.add(new JScrollPane(messageArea));

// Nút gửi
JButton sendBtn = new JButton("Gửi tới Server");
sendBtn.addActionListener(this::handleSend); // Gắn hàm gửi

// Vùng hiển thị kết quả (ở dưới)
resultArea = new JTextArea(10, 50);
resultArea.setEditable(false); // Không cho sửa
resultArea.setLineWrap(true); // Tự xuống dòng

// Gắn các panel vào frame
panel.add(inputPanel, BorderLayout.NORTH); // Trên
panel.add(sendBtn, BorderLayout.CENTER); // Giữa
panel.add(new JScrollPane(resultArea), BorderLayout.SOUTH); // Dưới

frame.add(panel); // Gắn panel vào frame
frame.setVisible(true); // Hiển thị giao diện
}
```

Hình 31. Hàm ShowUI (2)

- **Hàm *browseFile***: Xử lý mở hộp thoại chọn file và gán đường dẫn file vào *JTextField* tương ứng:

```
// Mở cửa sổ chọn file và gán đường dẫn vào ô nhập
private void browseFile(JTextField targetField) {
    JFileChooser fileChooser = new JFileChooser();
    int result = fileChooser.showOpenDialog(frame);
    if (result == JFileChooser.APPROVE_OPTION) {
        String path = fileChooser.getSelectedFile().getAbsolutePath();
        targetField.setText(path); // Gán vào text field
    }
}
```

Hình 32. Hàm *browseFile*

- **Hàm *handleSend***: Xử lý việc gửi thông tin từ client tới server qua kết nối HTTPS:

```
// Hàm xử lý khi người dùng bấm "Gửi tới Server"
private void handleSend(ActionEvent e) {
    try {
        String privateKeyPath = privateKeyField.getText().trim();
        String publicKeyPath = publicKeyField.getText().trim();
        String rawMessage = messageArea.getText().trim();

        // Kiểm tra nhập đủ đường dẫn
        if (privateKeyPath.isEmpty() || publicKeyPath.isEmpty()) {
            resultArea.setText("X INVALID: Bạn phải nhập đầy đủ đường dẫn private và public key.");
            return;
        }

        // Kiểm tra file tồn tại
        if (!Files.exists(Paths.get(privateKeyPath)) || !Files.exists(Paths.get(publicKeyPath))) {
            resultArea.setText("X INVALID: File khóa không tồn tại. Vui lòng kiểm tra lại đường dẫn.");
            return;
        }

        // Đọc khóa và ký message
        PrivateKey privateKey = CryptoUtils.loadPrivateKey(privateKeyPath);
        PublicKey publicKey = CryptoUtils.loadPublicKey(publicKeyPath);
        String signature = CryptoUtils.signMessage(rawMessage, privateKey); // Ký tin nhắn

        SecretKey aesKey = CryptoUtils.generateAESKey(); // Tạo khóa AES
        byte[] iv = CryptoUtils.generateIV(); // Sinh IV ngẫu nhiên

        String publicKeyPEM = CryptoUtils.encodePublicKey(publicKey); // Convert public key sang base64
        String encryptedPublicKey = CryptoUtils.encryptAES(publicKeyPEM, aesKey, iv); // Mã hóa bằng AES
    }
}
```

Hình 33. Hàm *handleSend* (1)

```

// Tạo JSON gửi lên server
Map<String, String> jsonMap = new HashMap<>();
jsonMap.put("raw_message", rawMessage);
jsonMap.put("signed_message", signature);
jsonMap.put("encrypted_public_key", encryptedPublicKey);
jsonMap.put("key", CryptoUtils.encodeKey(aesKey));
jsonMap.put("iv", CryptoUtils.encodeIV(iv));

Gson gson = new Gson();
String jsonPayload = gson.toJson(jsonMap); // Chuyển sang chuỗi JSON

// Bỏ qua xác thực chứng chỉ SSL (CHỈ DÙNG KHI TEST LOCALHOST)
TrustManager[] trustAllCerts = new TrustManager[]{
    new X509TrustManager() {
        @Override
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }

        @Override
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String authType) {
        }

        @Override
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String authType) {
        }
    }
};

```

Hình 34. Hàm handleSend (2)

```

// Thiết lập context SSL tin tất cả chứng chỉ
SSLContext sc = SSLContext.getInstance("TLS");
sc.init(null, trustAllCerts, new java.security.SecureRandom());
HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Bỏ qua kiểm tra hostname (dùng cho localhost)
HostnameVerifier allHostsValid = (hostname, session) -> true;
HttpsURLConnection.setDefaultHostnameVerifier(allHostsValid);

// Gửi POST tới server
URL url = new URL("https://localhost:8443");
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setDoOutput(true); // Cho phép ghi dữ liệu
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json");

// Gửi JSON
try (OutputStream os = conn.getOutputStream()) {
    os.write(jsonPayload.getBytes(StandardCharsets.UTF_8));
}

// Đọc phản hồi từ server
InputStream is = conn.getInputStream();
String response = new String(is.readAllBytes(), StandardCharsets.UTF_8);
resultArea.setText("📡 Server phản hồi:\n" + response);
} catch (Exception ex) {
    resultArea.setText("❌ Lỗi khi gửi: " + ex.getMessage());
}

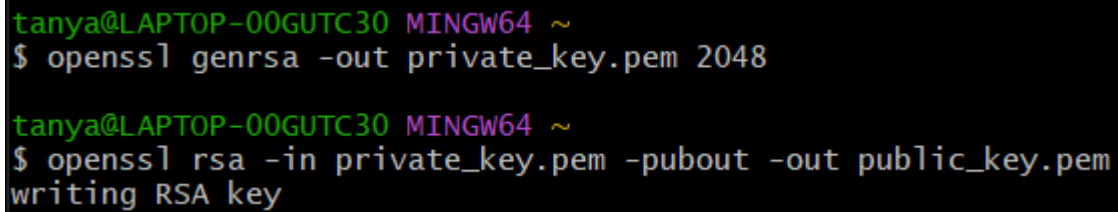
```

Hình 35. Hàm handleSend (3)

2. Triển khai demo.

Chuẩn bị:

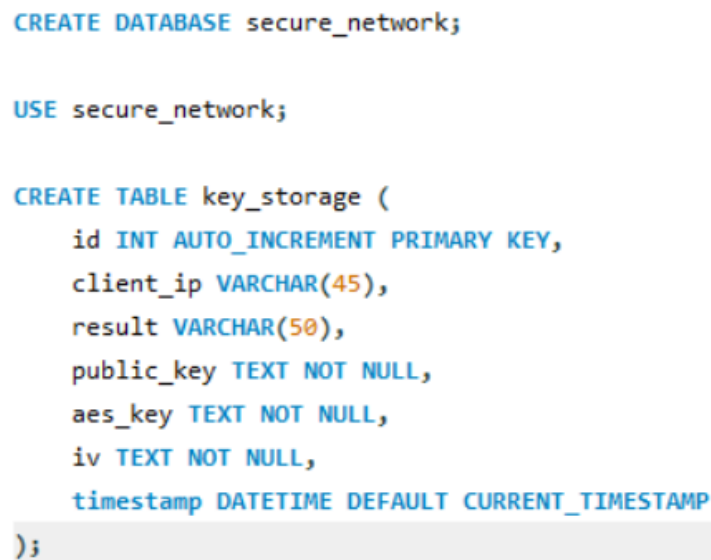
- File “private_key.pem” và “public_key.pem” (tạo bằng Git Bash):



```
tanya@LAPTOP-00GUTC30 MINGW64 ~  
$ openssl genrsa -out private_key.pem 2048  
  
tanya@LAPTOP-00GUTC30 MINGW64 ~  
$ openssl rsa -in private_key.pem -pubout -out public_key.pem  
writing RSA key
```

Hình 36. Tạo private key và public key

- Database secure_network có table key_storage (MySQL):



```
CREATE DATABASE secure_network;  
  
USE secure_network;  
  
CREATE TABLE key_storage (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    client_ip VARCHAR(45),  
    result VARCHAR(50),  
    public_key TEXT NOT NULL,  
    aes_key TEXT NOT NULL,  
    iv TEXT NOT NULL,  
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP  
);
```

Hình 37. Tạo database để lưu trữ dữ liệu

- File keystore.jks (tạo bằng KeyStore Explorer):



Hình 38. keystore.jks

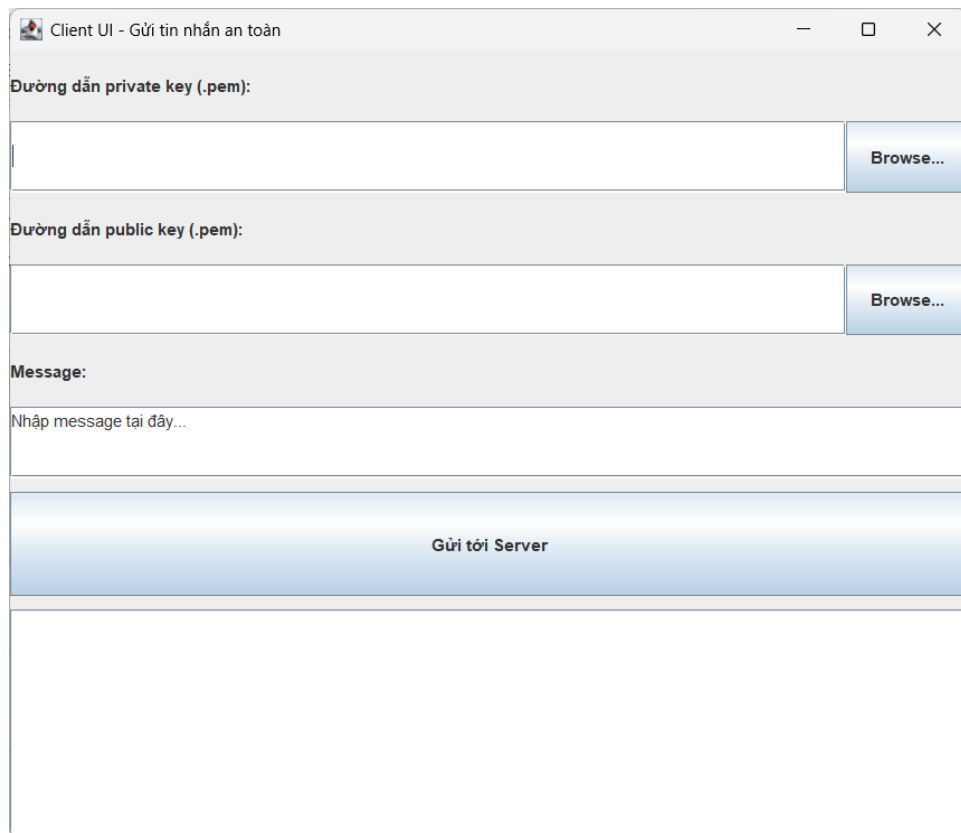
a. Khởi động:

- Khởi động Server. Server sẽ kết nối đến database và chạy trên cổng 8443:

```
Da ket noi MySQL thanh cong.  
Server dang chay tai http://localhost:8443 ...
```

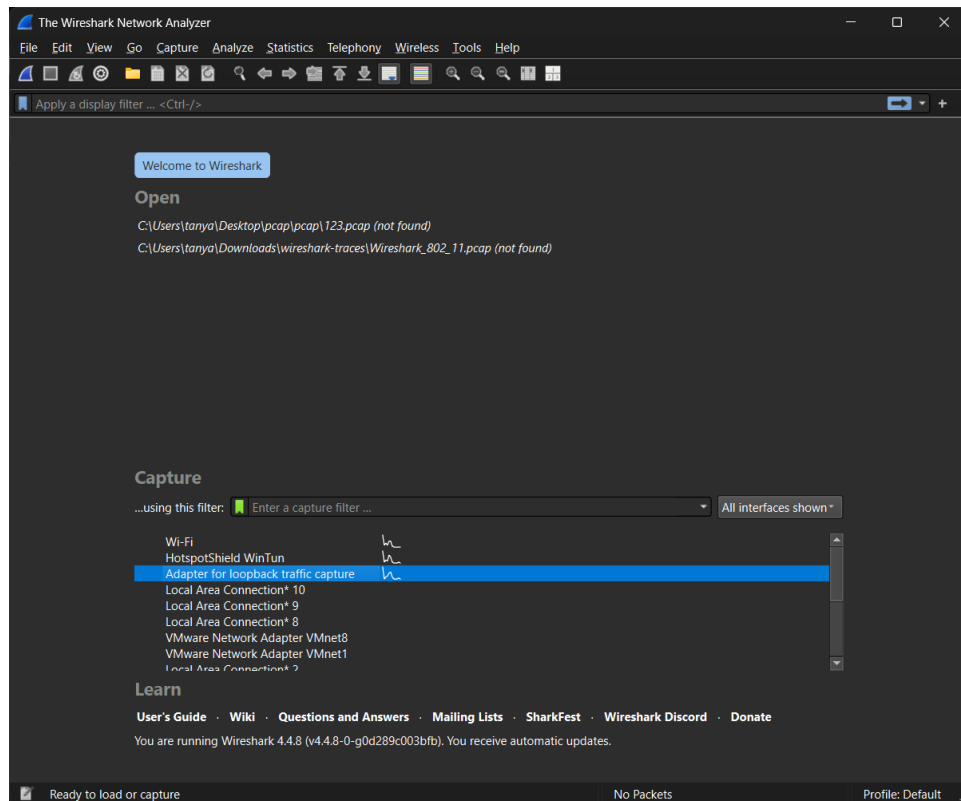
Hình 39. Khởi động Server

- Khởi động Client. Client sẽ kết nối đến Server thông qua cổng 8443 và hiển thị UI cho người dùng:



Hình 40. Client UI

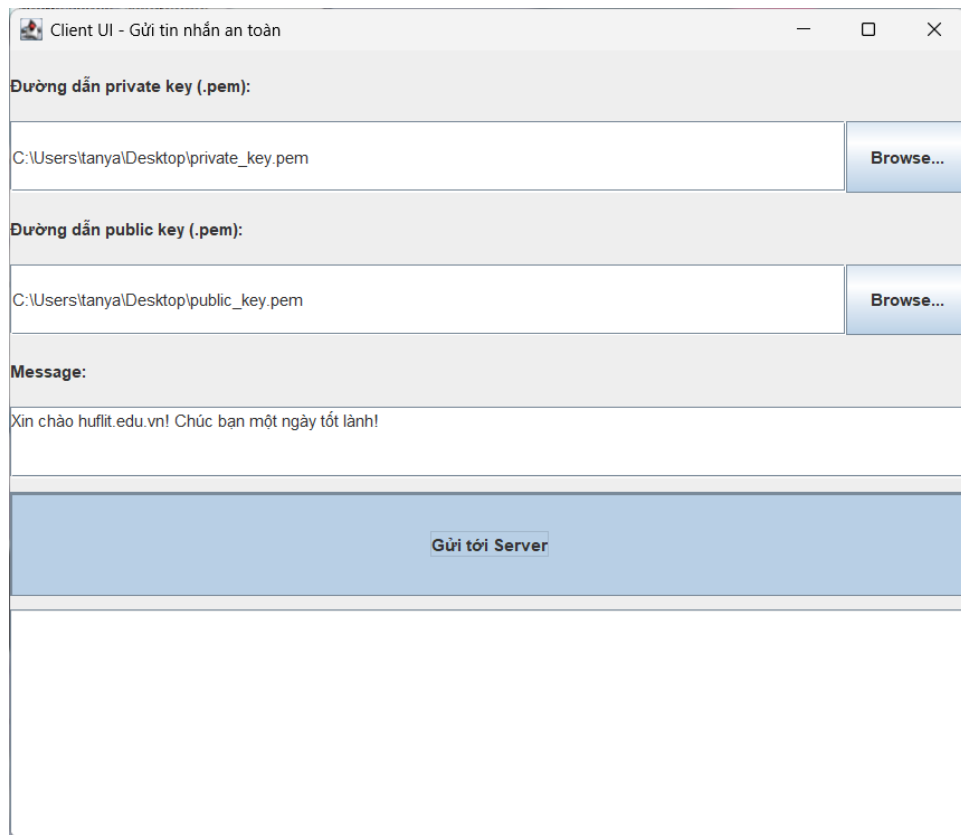
- Khởi động Wireshark và chọn “Adapter for loopback traffic capture” để quét các gói tin trong localhost.



Hình 41. Khởi động WireShark và quét các gói tin trong localhost

b. Chuẩn bị dữ liệu (Client-side):

- Người dùng nhập đường dẫn của 2 file “private_key.pem” và “public_key.pem” vào ô nhập liệu tương ứng rồi nhập nội dung tin nhắn có chứa domain bất kỳ vào ô Message rồi nhấn “Gửi tới Server”:



Hình 42. Người dùng nhập nội dung cần thiết vào các trường nhập dữ liệu

- Trong quá trình trước khi gửi, lớp `CryptoUtils` của Client sẽ tiến hành:
 - Ký tin nhắn bằng *SHA256withRSA*.
 - Encode chữ ký thành *BASE64*.
 - Sinh ngẫu nhiên *AES key* và *IV*.
 - Mã hóa *public_key.pem* bằng *AES-CBC*.
 - Chuẩn bị *JSON* gửi lên Server.
- c. **Server xử lý yêu cầu:** Server sau khi nhận request từ Client sẽ thực hiện:
 - Lưu *AES key* và *IV* vào database.
 - Giải mã *public key* đã mã hóa bằng *AES key* và *IV*.
 - Dùng *public key* đã giải mã để xác thực chữ ký:
 - Hash tin nhắn bằng *SHA-256*.
 - Dùng *RSA verify* với chữ ký đã được decode từ *BASE64*.
 - Nếu xác thực thành công thì tiếp tục.

- Nếu thất bại: trả về "VERIFICATION_FAILED".

```
Nhan request: {"raw_message":"Xin ch o huflit.edu.vn! Ch c m?t ng y t?t l nh!","encrypted_public_key":"z1COBSJpC/P
X c th c th nh c ng.
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Loi khi luu vao DB: Public Key Retrieval is not allowed
```

Hình 43. Server xử lý yêu cầu

d. Quét subdomain: Nếu xác thực thành công:

- Tải wordlist từ:
<https://github.com/danielmiessler/SecLists/blob/master/Discovery/DNS/subdomains-top1million-110000.txt>
- Quét domain: Dùng thư viện DNS resolver để kiểm tra các subdomain khả dụng của domain mà Client đã gửi:

```
Scanning domain: huflit.edu.vn
Tim thay: www.huflit.edu.vn
Tim thay: mail.huflit.edu.vn
Khong ton tai: ftp.huflit.edu.vn
Khong ton tai: localhost.huflit.edu.vn
Khong ton tai: webmail.huflit.edu.vn
Khong ton tai: smtp.huflit.edu.vn
Khong ton tai: webdisk.huflit.edu.vn
Khong ton tai: pop.huflit.edu.vn
Khong ton tai: cpanel.huflit.edu.vn
Khong ton tai: whm.huflit.edu.vn
Khong ton tai: ns1.huflit.edu.vn
Khong ton tai: ns2.huflit.edu.vn
Tim thay: autodiscover.huflit.edu.vn
Khong ton tai: autoconfig.huflit.edu.vn
Khong ton tai: ns.huflit.edu.vn
Khong ton tai: test.huflit.edu.vn
Khong ton tai: m.huflit.edu.vn
Khong ton tai: blog.huflit.edu.vn
Khong ton tai: dev.huflit.edu.vn
Khong ton tai: www2.huflit.edu.vn
Scan complete. T ng c ng 3 subdomains.
```

Hình 44. Server quét domain

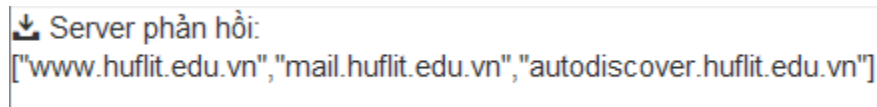
- Trả về danh sách subdomain hợp lệ về cho client.
- Lưu thông tin vào cơ sở dữ liệu:

#	id	client_ip	result	public_key	aes_key	iv	timestamp
1	26	127.0.0.1	hufflit.edu.vn	MIIBJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQE30Wf6ZUN/gj6MkgW7p6IMX72UD9Y ZiGz0+DKRYSimIMacDncddWB4G8cV+G3YGcnMfhes=	IWVvoWd06pPuc3fCmbQ75A==		2025-07-19 01:23:26.000
2	27	127.0.0.1	youtube.com	MIIBJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQE30Wf6ZUN/gj6MkgW7p6IMX72UD9Y KwBk+xxwvavfGgoWrCR4X7421dJPHRABG/bm2qfkk=	cEzw4anJ8H18KvoS/rU6uA==		2025-07-19 13:03:30.000
3	28	127.0.0.1	facebook.com	MIIBJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQE30Wf6ZUN/gj6MkgW7p6IMX72UD9Y 5Z3oK+o/NwOhKeQzH2vZ0z2FygWKNd/xJ5xJN2cB9w=	+sjHyPPhY/G6jHfu26uQ==		2025-07-19 13:10:23.000
4	29	127.0.0.1	facebook.com	MIIBJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQE30Wf6ZUN/gj6MkgW7p6IMX72UD9Y TWKfOPZ9RwyFuZTlaJfIF8tmNuy3NxMT9tL0G0JJeE=	jfpN38dKi5p9DG+GxSbvnw==		2025-07-19 13:13:00.000

Hình 45. Thông tin đã được lưu vào cơ sở dữ liệu

e. Response Client: Client nhận kết quả từ Server:

- Nếu xác minh thất bại: nhận "*VERIFICATION_FAILED*".
- Nếu thành công: nhận danh sách subdomain:



Hình 46. Server trả kết quả về Client

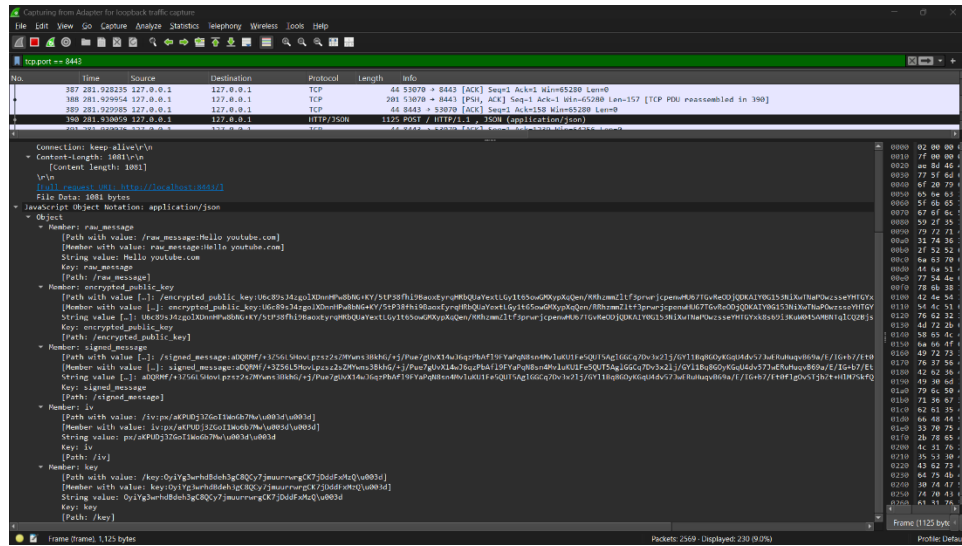
f. Kiểm tra WireShark:

- Sau khi sử dụng SSL/TLS, WireShark không thể quét bắt kỳ thông tin nào được gửi đi và gửi về giữa Client và Server:

1	0.000000	127.0.0.1	127.0.0.1	TCP	56 59214 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000062	127.0.0.1	127.0.0.1	TCP	56 8443 → 59214 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000075	127.0.0.1	127.0.0.1	TCP	44 59214 → 8443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4	0.007554	127.0.0.1	127.0.0.1	TLSv1.3	440 Client Hello
5	0.007612	127.0.0.1	127.0.0.1	TCP	44 8443 → 59214 [ACK] Seq=1 Ack=417 Win=65924 Len=0
6	0.195400	127.0.0.1	127.0.0.1	TLSv1.3	171 Server Hello
7	0.195481	127.0.0.1	127.0.0.1	TCP	44 59214 → 8443 [ACK] Seq=417 Ack=128 Win=65280 Len=0
8	0.195623	127.0.0.1	127.0.0.1	TLSv1.3	50 Change Cipher Spec
9	0.195644	127.0.0.1	127.0.0.1	TCP	44 59214 → 8443 [ACK] Seq=417 Ack=134 Win=65280 Len=0
10	0.201902	127.0.0.1	127.0.0.1	TLSv1.3	1139 Application Data
11	0.201945	127.0.0.1	127.0.0.1	TCP	44 59214 → 8443 [ACK] Seq=417 Ack=1229 Win=64256 Len=0
12	0.208506	127.0.0.1	127.0.0.1	TLSv1.3	50 Change Cipher Spec
13	0.208620	127.0.0.1	127.0.0.1	TCP	44 8443 → 59214 [ACK] Seq=1229 Ack=423 Win=65024 Len=0
14	0.221405	127.0.0.1	127.0.0.1	TLSv1.3	136 Application Data
15	0.221480	127.0.0.1	127.0.0.1	TCP	44 8443 → 59214 [ACK] Seq=1229 Ack=513 Win=64768 Len=0
16	0.226705	127.0.0.1	127.0.0.1	TLSv1.3	1049 Application Data
17	0.226820	127.0.0.1	127.0.0.1	TCP	44 59214 → 8443 [ACK] Seq=513 Ack=2234 Win=63232 Len=0
18	0.237734	127.0.0.1	127.0.0.1	TLSv1.3	239 Application Data
19	0.237769	127.0.0.1	127.0.0.1	TCP	44 8443 → 59214 [ACK] Seq=2234 Ack=708 Win=64768 Len=0
20	0.238817	127.0.0.1	127.0.0.1	TLSv1.3	1201 Application Data
21	0.238837	127.0.0.1	127.0.0.1	TCP	44 8443 → 59214 [ACK] Seq=2234 Ack=1865 Win=63488 Len=0
22	0.467131	127.0.0.1	127.0.0.1	TCP	56 59215 → 3306 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
23	0.467192	127.0.0.1	127.0.0.1	TCP	56 3306 → 59215 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
24	0.467734	127.0.0.1	127.0.0.1	TCP	44 59215 → 3306 [ACK] Seq=1 Ack=1 Win=65280 Len=0
25	0.511774	127.0.0.1	127.0.0.1	MySQL	122 Server Greeting proto=10 version=8.0.40
26	0.511864	127.0.0.1	127.0.0.1	TCP	44 59215 → 3306 [ACK] Seq=1 Ack=79 Win=65280 Len=0
27	0.548495	127.0.0.1	127.0.0.1	MySQL	289 Login Request user=root db=secure_network
28	0.548541	127.0.0.1	127.0.0.1	TCP	44 3306 → 59215 [ACK] Seq=79 Ack=240 Win=65280 Len=0
29	0.548565	127.0.0.1	127.0.0.1	MySQL	50 Caching_sha2_password perform_full_authentication
30	0.548595	127.0.0.1	127.0.0.1	TCP	44 59215 → 3306 [ACK] Seq=240 Ack=85 Win=65280 Len=0
31	0.558296	127.0.0.1	127.0.0.1	TCP	44 59215 → 3306 [ESTABLISHED] Seq=240 Ack=85 Win=65280 Len=0
32	0.558299	127.0.0.1	127.0.0.1	TCP	44 3306 → 59215 [ACK] Seq=85 Ack=247 Win=65280 Len=0
33	0.559592	127.0.0.1	127.0.0.1	MySQL	99 Response Error 1158
34	0.559595	127.0.0.1	127.0.0.1	TCP	44 59215 → 3306 [ACK] Seq=247 Ack=140 Win=0 Len=0
35	0.624810	127.0.0.1	127.0.0.1	TCP	45 58225 → 58226 [PSH, ACK] Seq=1 Ack=1 Win=255 Len=1
36	0.624537	127.0.0.1	127.0.0.1	TCP	44 58226 → 58225 [ACK] Seq=1 Ack=2 Win=199 Len=0

Hình 47. Không có gói tin nào bị bắt và để lộ thông tin

- So với trước khi dùng SSL/TLS:



Hình 48. WireShark bắt được toàn bộ thông tin mà Client gửi cho Server

CHƯƠNG IV. KẾT LUẬN

Qua quá trình thực hiện đồ án, nhóm đã xây dựng thành công mô hình hệ thống Client-Server có khả năng trao đổi dữ liệu một cách an toàn. Hệ thống ứng dụng các kỹ thuật mã hóa hiện đại như:

- **Mã hóa bất đối xứng RSA** để xác thực người gửi.
- **Mã hóa đối xứng AES (CBC Mode)** để đảm bảo tính bí mật cho nội dung truyền tải.
- **Chữ ký SHA256withRSA** để đảm bảo tính toàn vẹn.
- **Giao thức SSL/TLS** để mã hóa toàn bộ quá trình giao tiếp, giúp ngăn chặn việc rò rỉ thông tin trên mạng khi bị theo dõi bởi các công cụ như WireShark.

Bên cạnh đó, nhóm còn triển khai thread pool để xử lý đa luồng trên server, đảm bảo hiệu năng và khả năng xử lý đồng thời nhiều client.

Qua đồ án, nhóm đã nắm rõ hơn về cách thức triển khai và tích hợp các cơ chế bảo mật trong lập trình mạng, cũng như hiểu rõ vai trò của từng thành phần trong quy trình bảo mật: từ mã hóa dữ liệu, xác thực người dùng đến mã hóa luồng truyền thông. Điều này giúp tăng cường nhận thức về tầm quan trọng của bảo mật thông tin trong các hệ thống truyền thông hiện đại.

Tuy nhiên, hệ thống vẫn còn một số hạn chế như chưa xử lý hết các tình huống tấn công nâng cao (ví dụ: replay attack, man-in-the-middle phức tạp hơn), và việc triển khai SSL/TLS trong môi trường đa luồng vẫn còn khá thủ công. Nhóm mong muốn trong tương lai sẽ hoàn thiện hơn nữa hệ thống, tích hợp thêm các lớp bảo mật cao cấp như chứng thực CA, xác minh session và phòng chống tấn công nâng cao.

Tài liệu tham khảo

- <https://careerviet.vn/vi/talentcommunity/java-la-gi-tim-hieu-tu-a-z-ve-ngon-ngu-lap-trinh-java.35A520CF.html>
- <https://fptshop.com.vn/tin-tuc/danh-gia/java-la-gi-156801>
- <https://www.geeksforgeeks.org/java/java/>
- <https://aws.amazon.com/vi/what-is/java/>
- <https://topdev.vn/blog/tong-quan-ve-ngon-ngu-lap-trinh-java/>
- <https://itviec.com/blog/java-la-gi/>
- https://en.wikipedia.org/wiki/Thread_%28computing%29
- <https://www.techtarget.com/whatis/definition/thread>
- https://www.reddit.com/r/explainlikeimfive/comments/7y9deu/eli5_what_is_a_thread_in_programming/
- https://en.wikipedia.org/wiki/Thread_control_block
- <https://www.geeksforgeeks.org/operating-systems/thread-in-operating-system/>
- https://en.wikipedia.org/wiki/Java_concurrency
- <https://www.digitalocean.com/community/tutorials/what-is-a-socket>
- https://en.wikipedia.org/wiki/Network_socket
- <https://www.geeksforgeeks.org/computer-networks/socket-in-computer-network/>
- <https://www.scaler.com/topics/computer-network/socket-programming/>
- <https://medium.com/%40ahnafzamil/networking-101-what-are-sockets-7e2d274e153>
- https://www.youtube.com/watch?v=_FVvIJDQTxk
- <https://www.boardinfinity.com/blog/socket-in-computer-network/>

Phân chia công việc

Họ tên – MSSV	Công việc
Trần Thanh Tuấn Kiệt – 22DH114603	Lên kế hoạch, triển khai code
Tấn Lai Hoàng – 22DH111164	Tạo database, triển khai code
Trần Xuân Thành – 22DH114735	Sửa code, thêm chức năng
Phạm Quốc Anh – 18DH110179	Sửa code, viết báo cáo