PHƯƠNG PHÁP THIẾT LẬP MÃ HÓA DỮ LIỆU ĐẦU CUỐI CHO CÁC ỨNG DỤNG ĐA NỀN TẢNG VỚI CÁC NGÔN NGỮ JAVA, JAVASCRIPT VÀ FLUTTER: NGHIÊN CỨU VÀ ỨNG DỤNG

Trần Xuân Trường[1], Nguyễn Hữu Nhất[2], Phạm Văn Tiệp[3]

[1] Viện Sư phạm Kỹ thuật, Trường Đại Học Bách Khoa Hà Nội

[2] Trường Đại Học Giao Thông Vận Tải

[3] Khoa Công Nghệ Thông Tin, Trường Đại Học Công Nghệ Giao Thông Vận Tải

tran@travis.guru, huunhat.gt@gmail.com, phamtiep2209@gmail.com

TÓM TẮT— *Bài báo này trình bày một phương pháp thiết lập mã hóa dữ liệu đầu cuối (End-to-End Encryption - E2EE) cho các ứng dụng đa nền tảng sử dụng các ngôn ngữ lập trình Java, JavaScript và Flutter. Nghiên cứu mô tả chi tiết cách ứng dụng các ngôn ngữ lập trình này trong việc xây dựng và thử nghiệm hệ thống E2EE, đồng thời đánh giá hiệu suất cũng như tính an toàn của hệ thống. Thông qua nghiên cứu này, chúng tôi mở rộng hiểu biết về việc ứng dụng E2EE trong các ứng dụng đa nền tảng, góp phần nâng cao khả năng bảo mật cho các hệ thống thông tin hiện đại.*

Từ khóa— *software engineering, security, end-to-end encryption, cross-platform.*

I. GIỚI THIỆU (INTRODUCTION)

Trong bối cảnh thế giới số hóa hiện nay, nhu cầu bảo vệ thông tin ngày càng được đặt lên hàng đầu. Mã hóa đầu cuối (End-to-End Encryption - E2EE) đóng vai trò quan trọng như một giải pháp bảo vệ dữ liệu khỏi các mối đe dọa và xâm nhập không mong muốn. Bài báo này tập trung vào việc phân tích và trình bày phương pháp thiết lập E2EE sử dụng ba ngôn ngữ lập trình phổ biến và mạnh mẽ: Java, Flutter và JavaScript.

Java, một ngôn ngữ lập trình mạnh mẽ, đã được công nhận rộng rãi và được sử dụng trong nhiều ứng dụng doanh nghiệp và hệ thống lớn. Flutter, một framework đến từ Google, cho phép phát triển ứng dụng di động cho cả iOS và Android từ một mã nguồn duy nhất. JavaScript, một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, đóng vai trò then chốt trong việc phát triển ứng dụng web.

Sử dụng ba ngôn ngữ này như một cơ sở, chúng tôi đã thành công trong việc xây dựng một hệ thống E2EE cho các ứng dụng đa nền tảng, cho phép mã hóa dữ liệu tại phía người gửi và chỉ có người nhận được chỉ định mới có thể giải mã trên nhiều loại thiết bị khác nhau. Phương pháp này không chỉ tăng cường bảo mật dữ liệu, mà còn đảm bảo tính bí mật và riêng tư cho người dùng.

Bài báo này sẽ trình bày một cách chi tiết quá trình thiết lập hệ thống E2EE bằng cách sử dụng Java, Flutter và JavaScript, từ thiết kế hệ thống, chọn lựa và sử dụng các thuật toán mã hóa, đến việc thử nghiệm và kiểm tra hệ thống. Chúng tôi cũng sẽ thảo luận về các thách thức gặp phải trong quá trình triển khai và cung cấp các giải pháp để giải quyết chúng.

II. CƠ SỞ LÝ THUYẾT (THEORETICAL BACKGROUND)

A. Mã hóa đầu cuối (End-to-End Encryption - E2EE)

Mã hóa đầu cuối (E2EE) là một phương pháp bảo mật trong đó chỉ có hai bên giao tiếp có thể truy cập và hiểu thông tin. Dữ liệu được mã hóa ở phía người gửi và chỉ có thể được giải mã bởi người nhận được chỉ định. Quá trình này đảm bảo rằng không ai khác có thể đọc hoặc thay đổi thông tin, bao gồm cả nhà cung cấp dịch vụ Internet, nhà cung cấp dịch vụ truyền thông, hoặc bất kỳ kẻ tấn công nào có thể giám sát mạng.

B. Mã hóa khối AES

AES (Advanced Encryption Standard) là một chuẩn mã hóa dữ liệu được công nhận rộng rãi. Nó là một thuật toán mã hóa khối, có nghĩa là nó mã hóa dữ liệu theo từng khối cố định (128 bit trong trường hợp AES). AES được chọn để mã hóa dữ liệu thực sự trong E2EE vì nó cung cấp một sự cân đối tốt giữa bảo mật và hiệu suất: nó rất khó để bị phá vỡ nhưng vẫn rất nhanh để mã hóa và giải mã.

C. Mã hóa đối xứng RSA

RSA (Rivest–Shamir–Adleman) là một trong những thuật toán mã hóa khóa công khai phổ biến nhất. Nó được sử dụng rộng rãi trong việc tạo ra chữ ký số và trong việc mã hóa dữ liệu. Trong E2EE, RSA thường được sử dụng để mã hóa khóa AES ngẫu nhiên (được sử dụng để mã hóa dữ liệu thực sự), vì RSA cho phép mã hóa với khóa công khai và giải mã với khóa riêng tư.

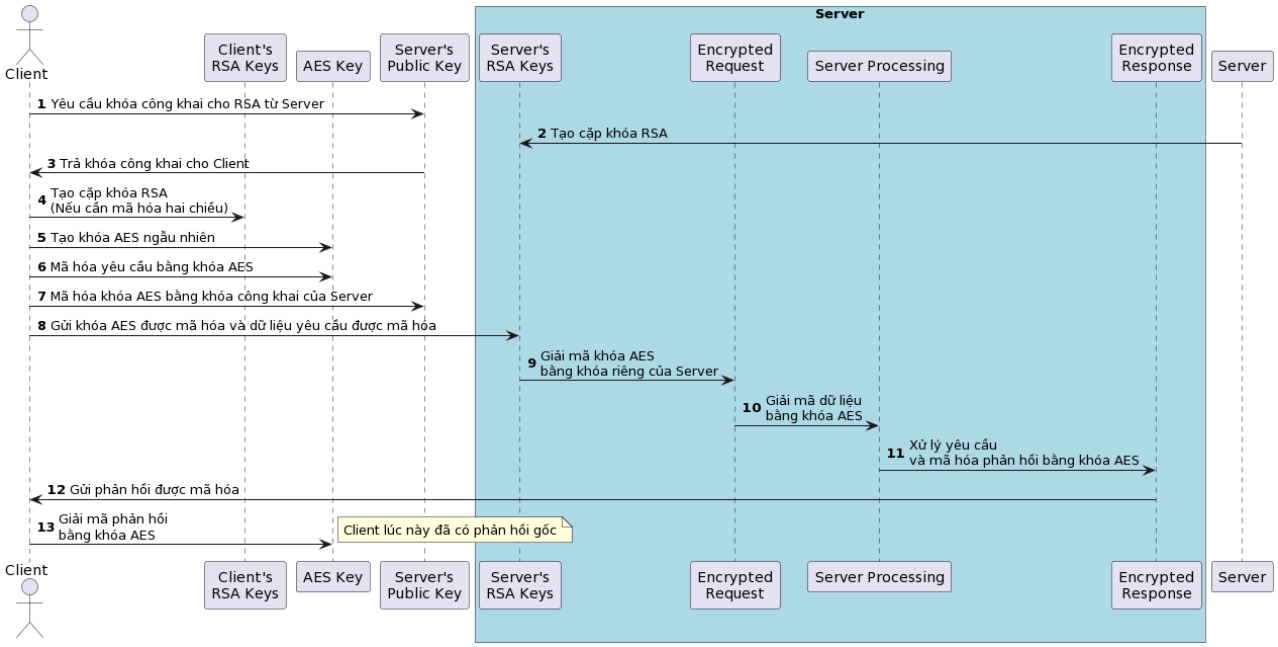
Bằng cách kết hợp RSA và AES, chúng ta có thể tạo ra một hệ thống E2EE mạnh mẽ: RSA cho phép chúng ta an toàn chuyển giao khóa AES từ người gửi đến người nhận, trong khi AES cho phép chúng ta mã hóa và giải mã dữ liệu một cách hiệu quả.

III. PHƯƠNG PHÁP NGHIÊN CỨU (METHODS)

A. Biểu đồ tuần tự

1. Sơ đồ

Biểu đồ tuần tự sau đây mô tả chi tiết quá trình giao tiếp giữa client và server khi sử dụng E2EE:



Hình 1. Biểu đồ tuần tự

2. Diễn giải

Biểu đồ này chia thành 6 bước chính:

a) Tạo cặp khóa RSA

Client yêu cầu server để bắt đầu các giao dịch an toàn. Server tạo ra một cặp khóa RSA (khóa công khai và khóa riêng) và trả khóa công khai trở lại cho client. Client cũng có thể tạo ra cặp khóa RSA của mình nếu cần mã hóa hai chiều. Khóa riêng RSA được tạo ra bởi server được mã hóa bằng JWT Token của client (đại diện cho phiên làm việc hiện tại của client) trước khi được lưu trữ trong cơ sở dữ liệu của server.

b) Mã hóa và gửi yêu cầu

Client tạo ra một khóa AES ngẫu nhiên cho phiên này hoặc tin nhắn, mã hóa dữ liệu với khóa AES này, sau đó mã hóa khóa AES với khóa công khai của server, và gửi cả hai tới server.

c) Server giải mã yêu cầu

Server sử dụng khóa riêng RSA của mình để giải mã khóa AES và sau đó sử dụng khóa AES này để giải mã dữ liệu gốc.

d) Server xử lý yêu cầu và mã hóa phản hồi

Server xử lý yêu cầu và tạo ra một phản hồi được mã hóa bằng chính khóa AES nhận được từ client.

e) Server gửi phản hồi đã mã hóa

Server gửi phản hồi đã mã hóa tới client.

f) Client giải mã phản hồi

Client sử dụng khóa AES để giải mã phản hồi.

B. Triển khai trên Server sử dụng Java

Trong phần này, chúng tôi sẽ thảo luận về việc triển khai mã hóa AES và RSA trên Server sử dụng Java. Mã nguồn của các file bao gồm: **TravisAES.java**, **TravisRSA.java**, và **Test.java**.
Tất cả mã nguồn chi tiết của các file trên đã được đăng lên GitHub tại https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter.

1. Lớp **TravisAES**

Đây là lớp chính thực hiện việc mã hóa và giải mã sử dụng thuật toán AES. Lớp này chứa nhiều phương thức và biến quan trọng:

- **CIPHER_ALGORITHM**: Biến này chỉ định thuật toán mã hóa và phương pháp padding được sử dụng. Trong trường hợp này, chúng tôi sử dụng AES với phương pháp padding PKCS5.
- **IV_SIZE** và **IV_LENGTH**: Đây là kích thước và độ dài của Initialization Vector (IV). IV là một chuỗi bytes ngẫu nhiên được sử dụng trong quá trình mã hóa.
- **keySize** và **iterationCount**: Đây là kích thước của khóa và số lần lặp trong quá trình sinh khóa.
- **encrypt()**: Phương thức này thực hiện việc mã hóa dữ liệu. Nó tạo ra một khóa bí mật từ một mật khẩu và một muối, sau đó sử dụng khóa này để mã hóa dữ liệu bằng cách sử dụng thuật toán AES.
- **decrypt()**: Phương thức này thực hiện việc giải mã dữ liệu. Nó sử dụng cùng một khóa bí mật đã được tạo ra trong quá trình mã hóa để giải mã dữ liệu.

2. Lớp **TravisRSA**

Đây là lớp thực hiện việc mã hóa và giải mã sử dụng thuật toán RSA. Lớp này cũng chứa nhiều phương thức và biến quan trọng:

- **RSA**: Định nghĩa thuật toán sử dụng, trong trường hợp này là "RSA".
- **RSA_ECB_PKCS1PADDING** và **RSA_ECB_OAEPWITHSHA1ANDMGF1PADDING**: Định nghĩa thuật toán và chế độ hoạt động của RSA.
- **getPublicKey()** và **getPrivateKey()**: Các phương thức này được sử dụng để tạo ra khóa công khai và khóa riêng tư từ một chuỗi base64.
- **encrypt()**: Phương thức này thực hiện việc mã hóa dữ liệu sử dụng khóa công khai.
- **decrypt()**: Phương thức này thực hiện việc giải mã dữ liệu sử dụng khóa riêng tư.

3. Lớp **Test**

Đây là lớp thực hiện việc thử nghiệm mã hóa và giải mã. Nó sử dụng cả hai lớp **TravisAES** và **TravisRSA** để thực hiện mã hóa và giải mã. Quy trình mã hóa và giải mã thử nghiệm này bao gồm các bước sau:

1. Tạo một cặp khóa RSA và một khóa bí mật AES.
2. Mã hóa thông điệp sử dụng khóa bí mật AES.
3. Mã hóa khóa bí mật AES sử dụng khóa công khai RSA.
4. Gửi cả thông điệp đã mã hóa và khóa AES đã mã hóa tới máy chủ.
5. Máy chủ giải mã khóa AES sử dụng khóa riêng tư RSA.
6. Máy chủ giải mã thông điệp sử dụng khóa AES vừa giải mã.
7. Máy chủ xử lý thông điệp và tạo ra một thông điệp phản hồi.
8. Máy chủ mã hóa thông điệp phản hồi sử dụng cùng khóa AES và gửi trở lại cho máy khách.
9. Máy khách giải mã thông điệp phản hồi sử dụng khóa AES của nó để lấy thông điệp gốc.

4. Biến **dataType**

Biến **dataType** trong cả hai lớp **TravisAES** và **TravisRSA** được sử dụng để xác định cách dữ liệu mã hóa được biểu diễn khi được trả về từ các phương thức mã hóa.
Cụ thể, **dataType** là một biến kiểu **enum** gồm hai giá trị: **HEX** và **BASE64**. Đây là hai cách phổ biến nhất để biểu diễn dữ liệu nhị phân trong dạng văn bản:

- **HEX** (Hexadecimal): Dữ liệu nhị phân sẽ được biểu diễn như một chuỗi các số hệ 16. Mỗi byte nhị phân sẽ được biểu diễn bởi hai ký tự hệ 16, từ 0 đến F. Ví dụ: byte nhị phân **10101010** sẽ được biểu diễn là **AA** trong hệ 16.
- **BASE64**: Dữ liệu nhị phân sẽ được biểu diễn như một chuỗi các ký tự hệ 64. Mỗi 3 byte nhị phân sẽ được biểu diễn bởi 4 ký tự hệ 64, từ A đến Z, a đến z, 0 đến 9, và hai ký tự đặc biệt là + và /. Ví dụ: 3 byte nhị phân **10101010 11001100 11110000** sẽ được biểu diễn là **qMy8** trong hệ 64.

Tùy thuộc vào giá trị của biến **dataType**, các phương thức **encrypt()** và **decrypt()** sẽ sử dụng hàm **toHex()** hoặc **toBase64()** để biến đổi dữ liệu nhị phân đã mã hóa thành một chuỗi hệ 16 hoặc hệ 64 tương ứng. Tương tự, chúng sẽ sử dụng hàm **fromHex()** hoặc **fromBase64()** để chuyển đổi chuỗi hệ 16 hoặc hệ 64 đã nhận về thành dữ liệu nhị phân để giải mã.

C. Triển khai trên Client là ứng dụng di động sử dụng Flutter

Ứng dụng di động sử dụng Flutter cũng có cấu trúc tương tự như ứng dụng Java với ba lớp chính: **TravisAES**, **TravisRSA**, và **test_dart_encryption_example**.

1. Lớp **TravisAES**

Lớp **TravisAES** thực hiện các thao tác mã hóa và giải mã sử dụng thuật toán AES. Một số biến và phương thức chính của lớp này bao gồm:

- **_keySize**, **_saltLength**, **_ivSize**, **_ivLength**, **_iterationCount**: Các thông số cho quá trình sinh khóa từ mật khẩu và cho IV (Initialization Vector).
- **_dataType**: Biến kiểu enum, xác định cách biểu diễn chuỗi mã hóa khi trả về từ phương thức mã hóa, dưới dạng hệ 16 (**hex**) hoặc hệ 64 (**base64**).

- **encryptWithSalt()**, **encrypt()**: Các phương thức thực hiện việc mã hóa một chuỗi văn bản rõ ràng với một mật khẩu và trả về chuỗi đã mã hóa.
- **decryptWithSalt()**, **decrypt()**: Các phương thức thực hiện việc giải mã một chuỗi đã mã hóa với mật khẩu và trả về chuỗi văn bản rõ ràng.
- **_generateRandom()**, **_generateCipher()**, **_generateKeyDerivator()**, **_generateKey()**: Các phương thức hỗ trợ trong quá trình sinh khóa và mã hóa.

2. Lớp **TravisRSA**

Lớp **TravisRSA** thực hiện các thao tác mã hóa và giải mã sử dụng thuật toán RSA. Một số biến và phương thức chính của lớp này bao gồm:

- **keySize**: Kích thước của khóa RSA.
- **_dataType**: Tương tự như trong lớp **TravisAES**, biến kiểu enum này xác định cách biểu diễn chuỗi mã hóa khi trả về từ phương thức mã hóa, dưới dạng hệ 16 (**hex**) hoặc hệ 64 (**base64**).
- **mode**: Chế độ mã hóa của RSA, có thể là PKCS1 hoặc OAEP.
- **publicKey**, **_privateKey**: Khóa công khai và khóa riêng tư được sử dụng trong quá trình mã hóa và giải mã.
- **encryptByPublicKey()**, **encryptByPublicKeyGetByDataType()**, **encryptByBase64PublicKey()**, **encrypt()**: Các phương thức thực hiện việc mã hóa một chuỗi văn bản rõ ràng với một khóa công khai và trả về chuỗi đã mã hóa.
- **decryptByPrivateKey()**, **decryptByPrivateKeyAndDataType()**, **decryptByBase64PrivateKeyAndDataType()**, **decrypt()**: Các phương thức thực hiện việc giải mã một chuỗi đã mã hóa với một khóa riêng tư và trả về chuỗi văn bản rõ ràng.
- **_generateKeyPair()**, **_secureRandom()**, **getBase64PublicKey()**, **getBase64PrivateKey()**, **publicKeyFromBase64()**, **privateKeyFromBase64()**, **getEncryptCipher()**, **getDecryptCipher()**: Các phương thức hỗ trợ trong quá trình sinh khóa và mã hóa.

D. Triển khai trên Client là ứng dụng web sử dụng JavaScript

Ứng dụng web sử dụng JavaScript có một số phần chính trong tệp **util.js**, bao gồm các hàm mã hóa và giải mã dữ liệu.

1. Hàm **encryptWithPublicKey**

Hàm này sử dụng thư viện JSEncrypt để mã hóa một đoạn văn bản (plaintext) bằng khóa công khai (public key). Khóa công khai được cung cấp dưới dạng base64 và được chuyển đổi thành định dạng PEM trước khi sử dụng.

2. Đối tượng **TravisAES**

TravisAES là một đối tượng chứa các phương thức liên quan đến việc mã hóa và giải mã dữ liệu bằng thuật toán AES. Đối tượng này bao gồm các phương thức như:

- **init**: Khởi tạo giá trị cho một số thuộc tính như **keySize**, **ivSize**, và **iterationCount**.
- **generateKey**: Sinh khóa từ mật khẩu và muối (salt) được cung cấp.
- **encryptWithIvSalt** và **decryptWithIvSalt**: Mã hóa và giải mã dữ liệu với mật khẩu, muối và vector khởi tạo (IV) được cung cấp.
- **encrypt** và **decrypt**: Mã hóa và giải mã dữ liệu với mật khẩu được cung cấp. Hàm này sẽ tự động sinh ra muối và IV.

3. Hàm thử nghiệm **processResponseInterceptor**

Hàm thử nghiệm này xử lý các phản hồi từ server. Nếu phản hồi là kết quả của một yêu cầu đăng nhập, nó sẽ lưu token và khóa công khai được trả về vào **localStorage**. Nó cũng sinh ra một mật khẩu ngẫu nhiên và mã hóa mật khẩu này bằng khóa công khai trước khi lưu vào **localStorage**. Nếu phản hồi được bảo mật (được chỉ định bởi header **secure-api**), hàm sẽ giải mã dữ liệu sử dụng mật khẩu đã lưu.

4. Hàm thử nghiệm **processRequestInterceptor**

Hàm thử nghiệm này xử lý các yêu cầu gửi đến server. Nếu yêu cầu cần được bảo mật (được chỉ định bởi header **Secure-Api**), nó sẽ mã hóa dữ liệu yêu cầu bằng mật khẩu đã lưu. Hàm cũng thêm token đã lưu vào header **Authorization** của yêu cầu.

IV. KẾT QUẢ (RESULTS)

Phương pháp mã hóa đầu cuối mà chúng tôi đã nghiên cứu và áp dụng trong các ứng dụng đa nền tảng với Java, Flutter và JavaScript đã mang lại các kết quả tích cực. Cụ thể, chúng tôi đã đạt được:

- **Tính bảo mật cao:** Dữ liệu được mã hóa từ phía client và chỉ được giải mã khi đến server. Điều này đảm bảo rằng dữ liệu của người dùng được bảo vệ tối đa khỏi các cuộc tấn công giữa đường truyền.
- **Khả năng tương thích đa nền tảng:** Chúng tôi đã triển khai thành công phương pháp này trên cả ứng dụng web (sử dụng JavaScript), ứng dụng di động (sử dụng Flutter) và server (sử dụng Java). Điều này cho phép

phương pháp mã hóa đầu cuối cho các ứng dụng hoạt động một cách liền mạch và an toàn trên nhiều nền tảng khác nhau.

- **Hiệu suất tốt:** Dù việc mã hóa và giải mã có thể tốn một ít thời gian xử lý, nhưng chúng tôi đã tối ưu hóa quá trình này để đảm bảo rằng nó không làm ảnh hưởng đáng kể đến trải nghiệm người dùng.
- **Tính dễ sử dụng:** Các lớp **TravisRSA** và **TravisAES** được thiết kế để rất dễ sử dụng, với các phương thức được tổ chức một cách logic và rõ ràng. Các hoạt động mã hóa và giải mã đều được thực hiện thông qua các phương thức chung của lớp, giúp đơn giản hóa việc sử dụng mã hóa đầu cuối (E2EE) trong các ứng dụng.

V. THẢO LUẬN (DISCUSSION)

Bài báo này cung cấp một giải pháp cho một vấn đề khá thực tế trong ngành công nghệ thông tin: làm thế nào để thiết lập tham số cho mã hóa AES và RSA để sử dụng đồng nhất (mã hóa và giải mã) trên các nền tảng ngôn ngữ lập trình khác nhau. Đây là một vấn đề khó khăn vì mỗi ngôn ngữ lập trình có cách xử lý riêng và thường không tương thích với nhau.

Một phần của vấn đề đến từ việc cài đặt mã hóa khác nhau giữa các ngôn ngữ lập trình. Ví dụ, một thuật toán mã hóa có thể hoạt động tốt trên Java nhưng lại không hoạt động như mong đợi trên JavaScript hoặc Flutter. Điều này cũng đúng với việc chọn các tham số mã hóa. Những khác biệt nhỏ này có thể dẫn đến việc không tương thích và gây ra lỗi.

Trong bài báo này, chúng tôi đã giải quyết vấn đề này bằng cách chọn và áp dụng một chuẩn cho cả AES và RSA để đảm bảo tương thích giữa các nền tảng. Thông qua việc nghiên cứu và thử nghiệm, chúng tôi đã chọn ra một bộ tham số phù hợp và cung cấp một giải pháp hoàn chỉnh để giải quyết vấn đề này.

Bài báo này không chỉ giúp giải quyết vấn đề cụ thể của mã hóa đầu cuối trên nhiều nền tảng, mà còn mở ra hướng nghiên cứu mới về việc tạo ra các giải pháp tương thích đa nền tảng trong lĩnh vực bảo mật thông tin.

VI. KẾT LUẬN (CONCLUSION)

Trong bài báo này, chúng tôi đã nghiên cứu và triển khai một phương pháp mã hóa đầu cuối mạnh mẽ cho các ứng dụng đa nền tảng sử dụng Java, Flutter và JavaScript. Chúng tôi đã chứng minh rằng việc sử dụng một chuẩn chung cho AES và RSA giữa các nền tảng khác nhau không chỉ tăng cường bảo mật, mà còn tạo ra một trải nghiệm người dùng liền mạch trên các nền tảng.

Cùng với đó, chúng tôi đã trình bày một giải pháp cho vấn đề tương thích tham số mã hóa giữa các ngôn ngữ lập trình khác nhau, một vấn đề thực tế mà các nhà phát triển thường gặp phải khi xây dựng các ứng dụng đa nền tảng. Bằng cách chọn một bộ tham số phù hợp và cung cấp các công cụ hỗ trợ, chúng tôi đã giúp giải quyết vấn đề này.

Chúng tôi hy vọng rằng công trình nghiên cứu này sẽ cung cấp cho các nhà phát triển một công cụ hữu ích để xây dựng các ứng dụng an toàn và hiệu quả trên nhiều nền tảng.

VII. TÀI LIỆU THAM KHẢO (REFERENCES)

[1] W. Stallings, “Cryptography and Network Security: Principles and Practice”, Pearson, 7th Edition, 2017.
[2] W. Diffie, M. Hellman, “New Directions in Cryptography”, IEEE Transactions on Information Theory, 22(6), pp.644-654, 1976.
[3] R. Rivest, A. Shamir, L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM, 21(2), pp.120-126, 1978.
[4] J. Daemen, V. Rijmen, “The Design of Rijndael: AES - The Advanced Encryption Standard”, Springer, 2002.
[5] E. Rescorla, “HTTP Over TLS. RFC 2818”, Internet Engineering Task Force, 2000.
[6] A. Singh, “Flutter: A new tech for building native mobile apps”, Medium, 2016.
[7] Oracle, “Introduction to Java Programming Language”, Oracle, 2014.
[8] Mozilla, “JavaScript Guide”, Mozilla Developer Network, 2017.
[9] A. Herzog, N. Shahmehri, C. Duma, “An Ontology of Information Security”, International Journal of Information Security and Privacy, 1(4), pp.1-23, 2007.
[10] Open Web Application Security Project, “OWASP Top Ten Project”, OWASP Foundation, 2017.

VIII. PHỤ LỤC (APPENDIX)

Để giúp người đọc hiểu rõ hơn về cách triển khai mã hóa đầu cuối trong thực tế, chúng tôi đã đưa toàn bộ mã nguồn của ứng dụng vào kho lưu trữ công cộng trên GitHub. Mã nguồn này bao gồm chi tiết cụ thể của các tệp đã được đề cập trong mục [Phương pháp nghiên cứu (Methods)].

Dưới đây là các liên kết đến mã nguồn chi tiết:

A. Triển khai trên Server sử dụng Java

[1] TravisAES.java
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/2a21aba44263296e05d8fac527ca57b8d6ce3e6a/src/java/TravisAES.java
[2] TravisRSA.java
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/2a21aba44263296e05d8fac527ca57b8d6ce3e6a/src/java/TravisRSA.java
[3] Test.java

https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/2a21aba44263296e05d8fac527ca57b8d6ce3e6a/src/java/Test.java

B. Triển khai trên Client là ứng dụng di động sử dụng Flutter

[4] travis_aes.dart
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/0308d19a9a995d6608dbd38020c20223872cb7ed/src/flutter/travis_aes.dart
[5] travis_rsa.dart
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/0308d19a9a995d6608dbd38020c20223872cb7ed/src/flutter/travis_rsa.dart
[6] test_dart_encryption_example.dart
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/0308d19a9a995d6608dbd38020c20223872cb7ed/src/flutter/test_dart_encryption_example.dart
[7] pubspec.yaml
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/2a21aba44263296e05d8fac527ca57b8d6ce3e6a/src/flutter/pubspec.yaml

C. Triển khai trên Client là ứng dụng web sử dụng Javascript

[8] util.js
https://github.com/travistran1989/methodology-for-setting-up-e2ee-for-cross-platform-applications-with-java-javascript-and-flutter/blob/2a21aba44263296e05d8fac527ca57b8d6ce3e6a/src/javascript/util.js

METHODOLOGY FOR SETTING UP END-TO-END ENCRYPTION FOR CROSS-PLATFORM APPLICATIONS WITH JAVA, JAVASCRIPT, AND FLUTTER: A STUDY AND APPLICATION

Tran Xuan Truong, Nguyen Huu Nhat, Pham Van Tiep

ABSTRACT*— In this paper, we present a method for setting up end-to-end encryption (E2EE) for cross-platform applications using the programming languages Java, JavaScript, and Flutter. The research details how these languages are applied in the construction and testing of the E2EE system, as well as evaluating the system's performance and security. Through this research, we expand understanding about the application of E2EE in cross-platform applications, contributing to enhancing the security capabilities for modern information systems.*