ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN
MẬT MÃ HỌC

ĐỀ TÀI: Mã hóa lai CP-ABE + AES kết hợp chữ ký số hậu lượng tử CRYSTALS-Dilithium cho hệ thống EHR
Hybrid CP-ABE + AES Encryption with Post-Quantum Digital Signatures (CRYSTALS-Dilithium) for EHR Systems

Giảng viên hướng dẫn: TS. Nguyễn Ngọc Tự

Thực hiện bởi Nhóm 5, gồm:

Họ và tên	MSSV	Vai trò
Nguyễn Thế Anh	23520066	Trưởng nhóm
Nguyễn Đức Hùng	23520565	Thành viên
Phan Đức Anh	23520071	Thành viên

○○ Tp. Hồ Chí Minh, 06/2025 ○○

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

....., ngày…….tháng……năm 2024

Người nhận xét

(Ký tên và ghi rõ họ tên)

MỤC LỤC

1. Tổng quan.....8
1.1. Bối cảnh và kịch bản ứng dụng cụ thể.....9
1.2. - Mục tiêu và Phạm vi Đồ án.....10
1.2.1. Các mục tiêu bảo mật cần đạt được.....10
1.2.2. Các mục tiêu triển khai của đồ án.....11
1.3 Giải pháp.....11
2. Tài sản, các bên liên quan và vai trò.....11
2.1. Các bên liên quan chính (Actors & Stakeholders).....11
2.2. Các tài sản cần bảo vệ (Assets).....12
2.3 Luồng tương tác giữa các bên liên quan.....14
3. Phân tích rủi ro bảo mật.....17
3.1. Spoofing.....17
3.2. Tampering.....18
3.3. Repudiation.....19
3.4. Information Disclosure.....19
3.5. Denial of Service.....20
3.6. Elevation of Privilege.....20
3.7. Spoofing / Tampering.....21
4. Kiến trúc Giải pháp.....21
4.1. Công cụ mật mã CP-ABE (abe_core.py):.....22
4.2. Cơ quan tin cậy (Trusted Authority - TA):.....23
4.3. Cổng ủy quyền (Authorization Gateway):.....25
4.4. Cơ sở dữ liệu siêu dữ liệu PostgreSQL:.....26
4.5. Cổng API RESTful:.....26
4.6. Lưu trữ đối tượng đám mây (ví dụ: AWS S3):.....28
4.7. Cơ chế thu hồi quyền truy cập: Phương pháp thu hồi dựa trên thời hạn.....28
4.8. Nguyên tắc thiết kế bảo mật:.....31
4.9 Lựa chọn chữ ký số - đảm bảo toàn vẹn hậu lượng tử với CRYSTALS-Dilithium. 31
4.2.1. Mô hình đe dọa và an ninh dài hạn.....32
4.9.3. Lựa chọn thư viện triển khai.....33

5. Quá trình phát triển & kiểm thử. 34
6. Triển khai và tự động hóa DevOps (Tùy chọn nâng cao trong tương lai). 36
7. Kiến trúc hệ thống. 38
7.1. Tổng quan kiến trúc. 41
7.2. Mô hình dữ liệu. 43
7.3. Quy trình mã hóa dữ liệu EHR sử dụng Hybrid CP-ABE + AES. 44
1. Khởi tạo hệ thống (setup). 44
2. Sinh khóa người dùng (keygen). 49
3. Luồng tải dữ liệu lên (Hàm upload). 53
4. Mã hóa dữ liệu EHR và bảo vệ khóa đối xứng AES. 58
5. Tạo Metadata, ký PQC, và hoàn thiện gói dữ liệu mã hóa. 61
6. Lưu trữ AES-GCM Key (store_key_package). 63
7. Truy xuất ciphertext (get_ctdk). 66
8. Luồng tải dữ liệu xuống và giải mã. 69
8. Triển Khai & Vận Hành 73
9. Result & Comments. 76

DANH MỤC BẢNG BIỂU

Bảng 1 - User 43

Bảng 2 - EHR_File 43

Bảng 3 - Hàm setup() 46

Bảng 4 - Hàm keygen() 51

Bảng 5 - Hàm upload 55

Bảng 6 - Hàm encrypt() 60

Bảng 7 – Gọi encryp trong Wrapper 62

Bảng 8 - Hàm store_key_package 64

Bảng 9 - Hàm get_key_package 67

DANH MỤC HÌNH ẢNH

Hình 1 - Luồng khởi tạo tài khoản và cấp phát khóa 14

Hình 2 - Luồng tải lên & đặt chính sách 15

Hình 3 – Luồng mã hóa & upload dữ liệu lên 15

Hình 4 - Luồng lưu trữ 16

Hình 5 - Luồng tải dữ liệu xuống 16

Hình 6 - Luồng giải mã 17

Hình 7 – Khởi tạo hệ thống (TA Setup) 38

Hình 8 - Đăng ký & cấp phát khóa bí mật.....39

Hình 9 - Tải lên & Mã hóa EHR 39

Hình 10 - Tải xuống & Chuẩn bị giải mã EHR.....40

Hình 11 - Giải mã 41

Hình 12 - Khởi tạo hệ thống 44

Hình 13 - Quá trình sinh khóa 49

Hình 14 - Luồng tải dữ liệu lên 53

Hình 15 - Mã hóa và upload dữ liệu 58

Hình 16 - Sơ đồ Luồng Lưu trữ Gói Khóa (Key Package) tại TA 63

Hình 17 - Truy xuất CTdk 66

Hình 18 - Luồng tải dữ liệu 69

Hình 19 - Giải mã EHR 71

Hình 20 - Trung bình thời gian mã hóa và giải mã 76

Hình 21 - setup() 77

Hình 22 – Deploy và get publickey 77

Hình 23 - get secret key 77

Hình 24 - Lưu master key 77

Hình 25 - Người dùng A có role là Doctor và department là Department là Cardiology upload file 78

Hình 26 - Người dùng A upload file thành công 78

Hình 27 - Người dùng B 79

Hình 28 - Người dùng B có cùng role và department với người dùng A tải file được . 79

Hình 29 - Người dùng C khác thuộc tính với người dùng A nên không tải được file về 80

NỘI DUNG BÀI LÀM

Link website: https://quantumabe.me/login.html

Link Github: https://github.com/ursuswh-metamorphic/CP_ABE-AES-_Hybrid_Encryption_For_EHR_Management_System-

Tài khoản và mật khẩu thử nghiệm:

Acccount 1:

Username: **alice@example.com**

Password: **InitPassalice**

Account 2:

Username: **bob@example.com**

Password: **InitPassbob**

Account 3:

Username: **carol@example.com**

Password: **InitPasscarol**

1. Tổng quan.

Sự phát triển của các hệ thống Hồ sơ Sức khỏe Cá nhân (Personal Health Record - PHR) đang trao cho bệnh nhân quyền kiểm soát chưa từng có đối với dữ liệu sức khỏe của chính mình. Thay vì bị phân mảnh tại nhiều bệnh viện, phòng khám, bệnh nhân giờ đây có thể tập hợp, lưu trữ và quản lý thông tin y tế của mình trên các nền tảng đám mây. Tuy nhiên, quyền kiểm soát này đi kèm với một thách thức bảo mật lớn: làm thế nào để bệnh nhân có thể chia sẻ các phần khác nhau trong hồ sơ của mình cho các đối tượng khác nhau (bác sĩ chuyên khoa, bác sĩ cấp cứu, người thân) một cách an toàn, linh hoạt và chi tiết?

Đồ án này giải quyết bài toán đó bằng cách thiết kế và triển khai một prototype cho nền tảng PHR an toàn. Hệ thống cho phép bệnh nhân, với vai trò là chủ sở hữu dữ liệu (Data Owner), có thể tải lên các tệp tin y tế của mình và tự định nghĩa các chính sách truy cập phức tạp trên từng tệp, sử dụng mã hóa lai dựa trên thuộc tính (CP-ABE) và AES.

1.1. Bối cảnh và kịch bản ứng dụng cụ thể.

Nhóm em xây dựng kịch bản xoay quanh một nền tảng PHR giả định có tên là "MyHealth Vault" và một người dùng là ông Nguyễn Văn An.

- Bối cảnh: Ông An là một người chủ động về sức khỏe, ông lưu trữ toàn bộ hồ sơ y tế của mình (kết quả khám từ nhiều nơi, lịch sử tiêm chủng, phim X-quang...) trên MyHealth Vault. Giờ đây, ông cần chia sẻ các thông tin này cho nhiều người khác nhau để phục vụ các mục đích khác nhau.

Kịch bản chi tiết: Ông Nguyễn Văn An quản lý và chia sẻ hồ sơ sức khỏe

- Tải lên và chia sẻ cho bác sĩ chuyên khoa:
  - Ông An cần đi khám tim mạch với BS. Trần Thị Bích. Ông tải lên tệp An_KhamTongQuat_2024.pdf chứa các kết quả khám sức khỏe gần đây.
  - Người tải lên: Nguyễn Văn An (chủ sở hữu).
  - Chính sách truy cập ông An đặt ra: Chỉ có bác sĩ chuyên khoa tim mạch mới được xem. Chính sách là: “doctor AND cardiology".
  - Kết quả kiểm soát: BS. Bích, người có đủ thuộc tính, có thể truy cập và xem tệp. Các bác sĩ khác không thể.
- Chuẩn bị cho tình huống khẩn cấp:
  - Để chuẩn bị cho các tình huống bất trắc, ông An tải lên tệp An_ThongTinKhanCap.pdf chứa thông tin nhóm máu, dị ứng thuốc và bệnh mãn tính.
  - Người tải lên: Nguyễn Văn An.
  - Chính sách truy cập ông An đặt ra: Tệp này phải có thể được truy cập bởi bác sĩ chuyên khoa của ông, HOẶC bởi bất kỳ bác sĩ nào trong phòng cấp cứu. Chính sách là: (doctor AND cardiology) OR emergency_staff.
  - Kết quả kiểm soát: BS. Bích có thể xem. Một bác sĩ cấp cứu tại một bệnh viện bất kỳ, người được hệ thống cấp thuộc tính tạm thời emergency_staff, cũng có thể xem được ngay lập tức.
- Chia sẻ cho người thân:
  - Ông An muốn con gái mình, chị Mai, giúp theo dõi lịch uống thuốc hàng ngày. Ông tải lên tệp An_LichUongThuoc.docx.
  - Người tải lên: Nguyễn Văn An.

- Chính sách truy cập ông An đặt ra: Chỉ có người thân được chỉ định mới được xem. Chính sách là: "FamilyMai".
- Kết quả kiểm soát: Chị Mai, người được ông An mời và được yêu cầu cho admin để cấp thuộc tính FamilyMai, có thể truy cập tệp này. Toàn bộ y bác sĩ, kể cả BS. Bích, đều không thể truy cập.

1.2. - Mục tiêu và Phạm vi Đồ án.

Để giải quyết các thách thức trong kịch bản đã nêu, đồ án hướng tới việc đáp ứng các mục tiêu bảo mật cốt lõi thông qua việc triển khai một hệ thống prototype với các mục tiêu cụ thể.

1.2.1. Các mục tiêu bảo mật cần đạt được.

Hệ thống phải đảm bảo các đặc tính bảo mật sau, được áp dụng trực tiếp vào kịch bản Bệnh viện VINHEALTH:

- Bảo mật (Confidentiality): Đảm bảo chỉ những người dùng có thuộc tính phù hợp với chính sách mới có thể giải mã và đọc được nội dung của một tệp tin EHR.
  - Ví dụ: Nội dung tệp An_LichUongThuoc.docx phải tuyệt đối bí mật với tất cả mọi người trừ chị Mai.
- Kiểm soát truy cập chi tiết (Fine-grained Access Control): Hệ thống phải có khả năng thực thi các chính sách truy cập phức tạp dựa trên sự kết hợp của nhiều thuộc tính (vai trò, chuyên khoa, mã định danh...).
  - Ví dụ: Hệ thống phải thực thi chính xác chính sách do ông An đặt ra: (doctor AND cardiology) OR emergency_staff.
- Toàn vẹn (Integrity): Đảm bảo dữ liệu EHR và các chính sách liên quan không bị thay đổi trái phép trong quá trình lưu trữ và truyền tải.
- Không từ chối (Non-repudiation): Ghi lại nhật ký các hành động quan trọng (ví dụ: ai đã yêu cầu khóa để giải mã tệp nào) để ngăn chặn việc người dùng chối bỏ hành động của mình.
- Tính xác thực (Authenticity): Đảm bảo các thực thể tham gia vào hệ thống (người dùng, Trusted Authority) đều là chính danh.
- Sẵn có (Availability): Đảm bảo người dùng được ủy quyền có quyền truy cập đáng tin cậy và kịp thời vào EHRs.

1.2.2. Các mục tiêu triển khai của đồ án.

Để hiện thực hóa các mục tiêu bảo mật trên, phạm vi công việc của đồ án này bao gồm:

Thiết kế kiến trúc: Xây dựng một kiến trúc hệ thống khả thi, tách biệt rõ ràng các thành phần: Trusted Authority (TA) để quản lý khóa, API Gateway để xử lý nghiệp vụ, và nơi lưu trữ tệp tin (Cloud Storage).

Triển khai Prototype: Phát triển một hệ thống prototype hoạt động được, bao gồm các chức năng cốt lõi:

Khởi tạo hệ thống (setup).

Tạo khóa cho người dùng dựa trên thuộc tính (keygen).

Mã hóa một tệp tin với một chính sách truy cập (encrypt).

Giải mã tệp tin bằng khóa cá nhân (decrypt).

Chứng minh tính khả thi: Sử dụng prototype đã xây dựng để thực thi lại toàn bộ các luồng nghiệp vụ trong kịch bản của bệnh nhân Nguyễn Văn An. Ghi lại kết quả (dưới dạng hình chụp màn hình và log) để chứng minh hệ thống hoạt động đúng như thiết kế.

Đánh giá sơ bộ: Thực hiện các phép đo đơn giản để đánh giá hiệu năng (thời gian xử lý) của các hoạt động mã hóa và giải mã chính trong prototype.

1.3 Giải pháp.

Module cốt lõi (abe_core.py) cung cấp các hàm cần thiết để tạo khóa, mã hóa và giải mã dữ liệu sử dụng CP-ABE. Giải pháp sử dụng **mã hóa lai (hybrid encryption)** kết hợp CP-ABE với mã hóa đối xứng (AES) để cải thiện hiệu suất.

2. Tài sản, các bên liên quan và vai trò.

2.1. Các bên liên quan chính (Actors & Stakeholders).

- Bệnh nhân (Data Owner & Data Creator):
  - Trong kịch bản: Ông Nguyễn Văn An.
  - Vai trò: Đây là vai trò trung tâm của hệ thống. Ông An là người sở hữu, người tải lên và là người trực tiếp đặt ra các chính sách truy cập cho từng tệp tin y tế của mình.
- Người dùng dữ liệu (Data Users):
  - Trong kịch bản:

        - Bác sĩ chuyên khoa: BS. Trần Thị Bích (yêu cầu thuộc tính: doctor, cardiology).
        - Nhân viên y tế cấp cứu (yêu cầu thuộc tính: emergency_staff).
        - Người thân: Chị Mai (yêu cầu thuộc tính: FamilyMai).
        - Vai trò: Là những người được ông An cấp quyền truy cập dữ liệu. Họ chỉ có thể giải mã và xem các tệp tin nếu các thuộc tính của họ khớp với chính sách do ông An đã đặt ra. Họ không thể tự tạo hay sửa đổi dữ liệu của ông An.
- Quản trị viên Nền tảng (Platform Administrator):
    - Vai trò: Là người quản lý và bảo trì nền tảng "MyHealth Vault". Đây là thực thể duy nhất có quyền khởi tạo tài khoản người dùng trong hệ thống. Quản trị viên sẽ tạo trước tài khoản cho các bác sĩ, người thân... và gán các thuộc tính đã được xác minh ngoài đời thực (offline verification) cho các tài khoản đó. Hệ thống không có chức năng đăng ký công khai.
- Cơ quan Tin cậy (Trusted Authority - TA):
    - Vai trò: Là một dịch vụ hệ thống cốt lõi của nền tảng "MyHealth Vault". Nó chịu trách nhiệm về toàn bộ hoạt động mật mã:
        - Bảo vệ Khóa chính (MK) của toàn bộ nền tảng.
        - Cấp phát Khóa bí mật (SK) cho tất cả người dùng (ông An, BS. Bích, chị Mai...) dựa trên các thuộc tính mà Quản trị viên Nền tảng đã gán cho họ.
- Nhà cung cấp Dịch vụ Đám mây (Cloud Service Provider - CSP):
    - Trong kịch bản: Amazon Web Services (AWS).
    - Vai trò: Cung cấp hạ tầng lưu trữ (AWS S3) cho các tệp tin đã mã hóa. CSP là thực thể không đáng tin cậy về quyền riêng tư.

2.2. Các tài sản cần bảo vệ (Assets).

Để xây dựng mô hình an ninh, hệ thống xác định các tài sản thông tin và mật mã quan trọng cần được bảo vệ ở mức độ cao nhất.

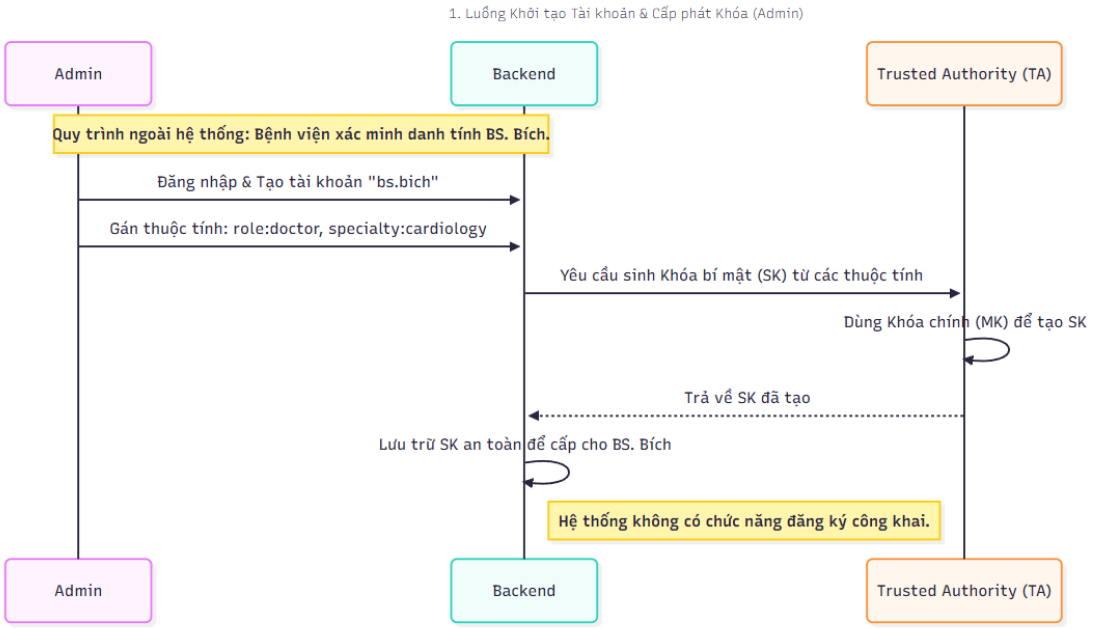
- Bản rõ (Plaintext):

- Mô tả: Nội dung gốc, chưa được mã hóa của các tệp tin y tế. Đây là tài sản nhạy cảm và có giá trị nhất cần được bảo vệ tính bí mật.
- Ví dụ: Nội dung chi tiết trong tệp An_LichUongThuoc.docx.

– Khóa Phiên Đối xứng (Symmetric Session Key):
  - Mô tả: Một khóa AES-256 được tạo ngẫu nhiên, chỉ sử dụng một lần duy nhất để mã hóa nội dung của một tệp EHR. Khóa này là "chìa khóa" tạm thời để mở dữ liệu, do đó việc bảo vệ nó là cực kỳ quan trọng.
  - Ví dụ: Khóa AES được tạo ra để mã hóa tệp An_ThongTinKhanCap.pdf.
– Bản mã của Khóa Phiên (ABE-Encrypted Session Key):
  - Mô tả: Là Khóa Phiên Đối xứng sau khi đã được "bọc" hoặc mã hóa bằng thuật toán CP-ABE với một chính sách truy cập do người dùng đặt ra. Đây là tài sản cốt lõi của cơ chế kiểm soát truy cập.
  - Ví dụ: Khóa AES ở trên, sau khi đã được mã hóa bằng chính sách (doctor and cardiology) or emergency_staff.
– Dữ liệu EHR đã mã hóa (Encrypted EHR Data):
  - Mô tả: Nội dung tệp EHR sau khi đã được mã hóa bằng Khóa Phiên Đối xứng. Đây là tài sản được lưu trữ trên các hệ thống không đáng tin cậy như AWS S3.
  - Ví dụ: Tệp An_KhamTongQuat_2024.pdf được lưu trên S3 dưới dạng một chuỗi bytes không thể đọc được.
– Khóa Bí mật Người dùng (User's Secret Key - SK):
  - Mô tả: Khóa riêng tư, không thể chia sẻ, được TA cấp cho mỗi người dùng. Khóa này chứa các thuộc tính của người dùng và là công cụ duy nhất để giải mã các Khóa Phiên đã được bảo vệ bằng ABE.
  - Ví dụ: Khóa SK của chị Mai chứa thuộc tính "FamilyMai".
– Khóa Chủ (Master Key - MK):
  - Mô tả: Khóa tối cao của toàn bộ hệ thống ABE, được TA sử dụng để tạo ra tất cả các Khóa bí mật (SK). Đây là tài sản tối mật, được bảo vệ nghiêm ngặt nhất, ví dụ như trong AWS KMS.
  - Ví dụ: Khóa tối cao duy nhất của nền tảng "MyHealth Vault".
– Gói Siêu dữ liệu và Chữ ký số (Signed Metadata Package):

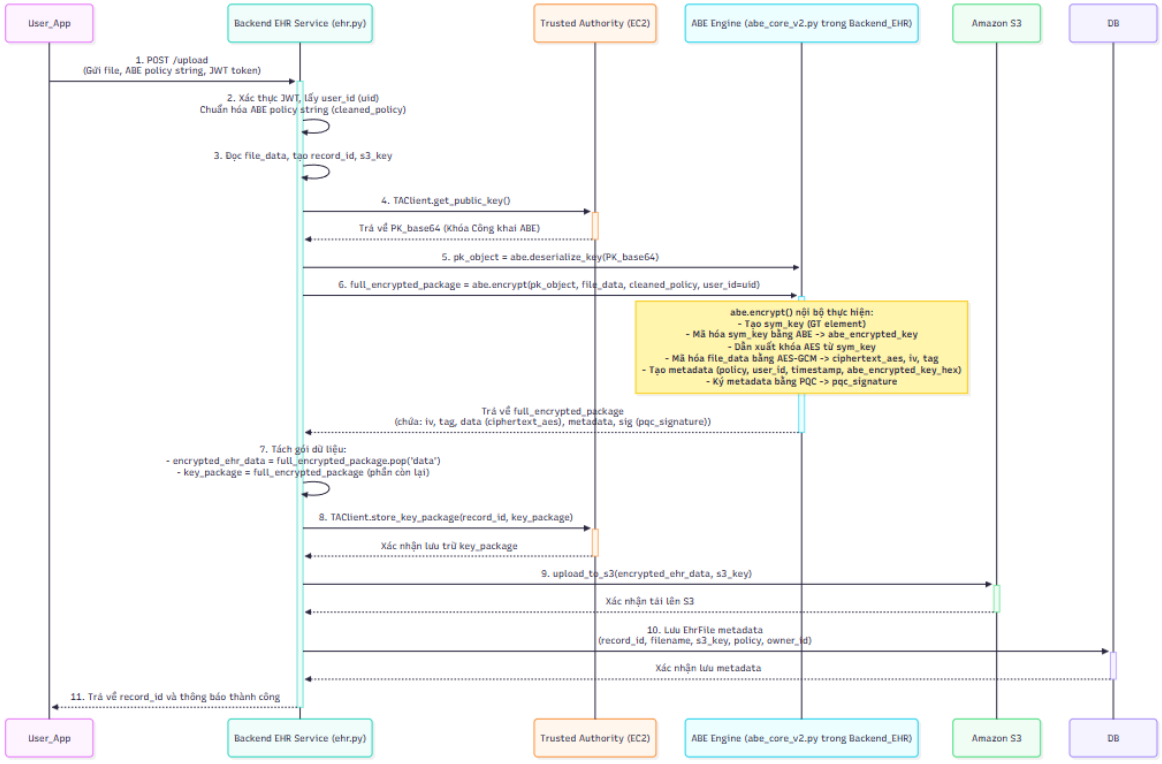
- Mô tả: Đây là một tài sản mật mã tổng hợp, có vai trò đảm bảo tính toàn vẹn và xác thực ngữ cảnh. Nó bao gồm:
  - Metadata: Một đối tượng chứa thông tin ngữ cảnh quan trọng như policy, user_id, timestamp, và Bản mã của Khóa Phiên.
  - Chữ ký số PQC: Toàn bộ đối tượng metadata này được ký điện tử bằng thuật toán hậu lượng tử Dilithium2.
- Ví dụ: Trước khi lưu trữ, hệ thống sẽ tạo một gói metadata cho tệp của ông An, sau đó dịch vụ Backend sẽ dùng khóa ký của mình để tạo ra chữ ký cho gói metadata đó. Chữ ký này đảm bảo rằng không ai có thể thay đổi chính sách hay ngữ cảnh của tệp mà không bị phát hiện.

2.3 Luồng tương tác giữa các bên liên quan.



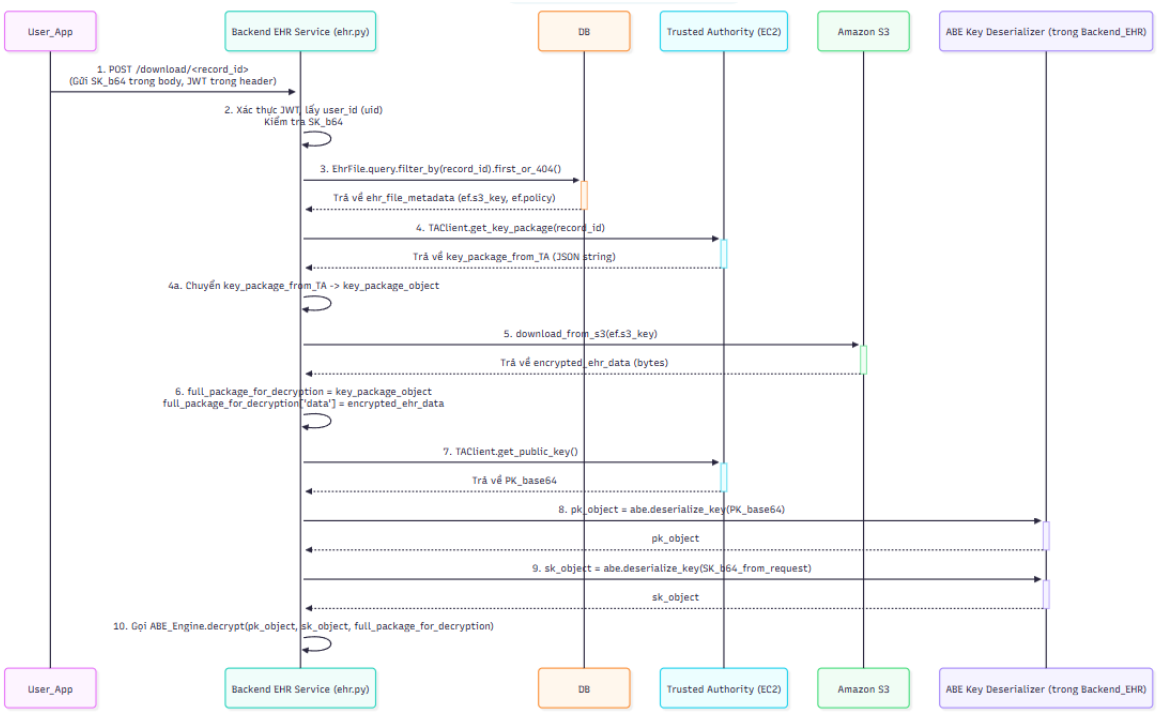
Hình 1 - Luồng khởi tạo tài khoản và cấp phát khóa

Hình 2 - Luồng tải lên & đặt chính sách

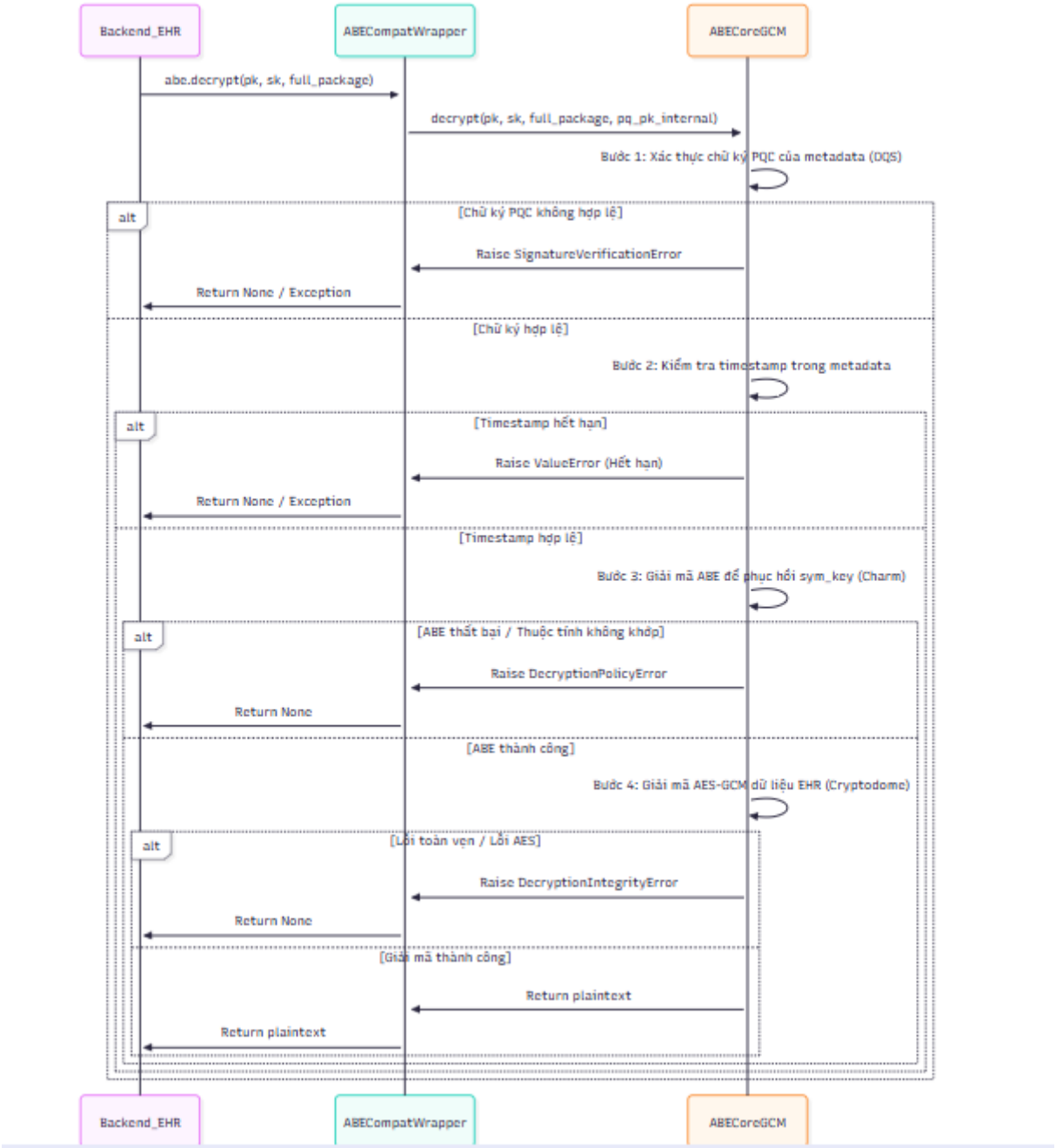


Hình 3 – Luồng mã hóa & upload dữ liệu lên

Hình 4 - Luồng lưu trữ



Hình 5 - Luồng tải dữ liệu xuống



Hình 6 - Luồng giải mã

3. Phân tích rủi ro bảo mật.

Để đảm bảo tính bảo mật cho hệ thống "MyHealth Vault", nhóm em thực hiện phân tích rủi ro dựa trên mô hình STRIDE, áp dụng trực tiếp vào các luồng nghiệp vụ và tài sản đã xác định trong kịch bản của bệnh nhân Nguyễn Văn An.

3.1. Spoofing.

- Kịch bản tấn công cụ thể: Một y tá trong bệnh viện sử dụng máy tính mà BS. Bích quên đăng xuất để yêu cầu truy cập hồ sơ của ông An.

- Tác nhân (Actor): Y tá hoặc người dùng nội bộ khác.
- Mục tiêu (Target): Tài khoản của BS. Bích.
- Tác động (Impact): Truy cập trái phép vào dữ liệu nhạy cảm mà BS. Bích có quyền, vi phạm quyền riêng tư của ông An.
- Biện pháp giảm thiểu (Mitigation):
    - Xác thực mạnh: Yêu cầu mật khẩu hoặc yếu tố thứ hai khi thực hiện các hành động nhạy cảm (như giải mã).
    - Chính sách Time-out: Tự động đăng xuất phiên làm việc sau một khoảng thời gian không hoạt động.

3.2. Tampering.

- Kịch bản tấn công cụ thể: Một quản trị viên CSDL bất mãn sửa đổi trực tiếp chính sách của tệp An_LichUongThuoc.docx trong bảng ehr_files từ FamilyMai thành doctor.
- Tác nhân (Actor): Quản trị viên CSDL.
- Mục tiêu (Target): Cột policy trong cơ sở dữ liệu.
- Tác động (Impact): BS. Bích có thể giải mã và xem được lịch uống thuốc, một thông tin mà ông An không hề muốn chia sẻ cho bác sĩ.
- Biện pháp giảm thiểu (Mitigation):
    - Kiểm soát truy cập CSDL: Áp dụng nguyên tắc đặc quyền tối thiểu, chỉ tài khoản dịch vụ của Backend mới có quyền ghi/sửa bảng này.
    - Ghi nhật ký (Audit Log): Bật ghi nhật ký ở tầng CSDL để theo dõi mọi thay đổi vào bảng ehr_files, đặc biệt là cột policy.
    - Toàn vẹn Chính sách bằng Chữ ký số (Cryptographic Policy Integrity)**:** Đây là biện pháp giảm thiểu cốt lõi. Trong kiến trúc hệ thống, chuỗi chính sách (policy) được đưa vào một gói metadata và toàn bộ gói này được ký điện tử bằng chữ ký hậu lượng tử Dilithium2. Khi giải mã, hệ thống sẽ xác thực chữ ký này trước tiên. Bất kỳ sự thay đổi nào đối với cột policy trong cơ sở dữ liệu sẽ dẫn đến sự không khớp với chính sách đã được ký trong metadata, khiến cho quá trình xác thực chữ ký thất bại và yêu cầu giải mã sẽ bị từ chối ngay lập tức. Điều này cung cấp một sự đảm bảo bằng mật mã rằng chính sách không thể bị giả mạo.

3.3. Repudiation.

- Kịch bản tấn công cụ thể: Ông An tải lên một tệp chứa thông tin sai lệch, sau đó khi có sự cố, ông An chối bỏ rằng mình đã từng tải lên tệp đó.
- Tác nhân (Actor): Ông An (Người dùng hợp pháp).
- Mục tiêu (Target): Tính toàn vẹn của nhật ký hệ thống.
- Tác động (Impact): Gây tranh cãi về nguồn gốc dữ liệu, làm mất uy tín của hệ thống. Khó truy cứu trách nhiệm.
- Biện pháp giảm thiểu (Mitigation):
    - Ghi nhật ký chi tiết: Hệ thống phải ghi lại log không thể sửa đổi (immutable log) cho mọi hành động upload, bao gồm: user_id, filename, timestamp, source_ip.
    - Thông báo: Gửi email hoặc thông báo trong ứng dụng cho ông An ngay sau khi upload thành công để xác nhận hành động.
    - **Bằng chứng Mật mã về Nguồn gốc (Cryptographic Proof of Origin):** Ngoài việc ghi log, hệ thống còn tạo ra một bằng chứng mật mã không thể chối cãi. Gói metadata được ký điện tử có chứa trường user_id của người thực hiện upload. Chữ ký này ràng buộc một cách chặt chẽ định danh của người dùng với hành động tạo ra bản mã tại một thời điểm cụ thể, khiến cho việc chối bỏ trở nên bất khả thi về mặt kỹ thuật.

3.4. Information Disclosure.

- Kịch bản tấn công cụ thể: Kẻ tấn công truy cập được vào máy chủ TA và đánh cắp file key_package_store.json.
- Tác động (Impact): Kẻ tấn công có được toàn bộ các "gói khóa" (bao gồm IV, tag, metadata và chữ ký). Tuy nhiên, đây là một thiết kế an toàn theo nguyên tắc "phòng thủ theo chiều sâu":
    1. Bản thân khóa phiên vẫn được mã hóa bằng ABE bên trong metadata, kẻ tấn công không thể giải mã nếu không có khóa bí mật (SK) của người dùng hợp lệ.
    2. Dữ liệu EHR thực tế được lưu trữ riêng biệt trên S3, kẻ tấn công vẫn chưa có được phần này.
- Biện pháp giảm thiểu (Mitigation):

- Mã hóa tại nơi lưu trữ (Encryption-at-Rest): Toàn bộ ổ đĩa của máy chủ TA phải được mã hóa.
- Cách ly mạng: TA phải được đặt trong một mạng riêng (VPC), được bảo vệ bởi tường lửa chặt chẽ, chỉ cho phép kết nối từ địa chỉ IP của dịch vụ Backend, và mã hóa kết nối giữa TA và Backend là TLS1.3.

3.5. Denial of Service.

– Kịch bản tấn công cụ thể: Một người dùng (ví dụ: Chị Mai) viết một script gọi liên tục đến API yêu cầu giải mã tệp, làm quá tải Backend hoặc TA.
– Tác nhân (Actor): Người dùng hợp pháp nhưng có ý đồ xấu (hoặc máy bị nhiễm malware).
– Mục tiêu (Target): API Gateway, TA Service.
– Tác động (Impact): Hệ thống trở nên chậm hoặc không phản hồi với những người dùng khác. Ông An không thể chia sẻ hồ sơ cho bác sĩ trong tình huống khẩn cấp.
– Biện pháp giảm thiểu (Mitigation):
    - Giới hạn tần suất (Rate Limiting): Áp dụng rate limiting trên API Gateway cho mỗi người dùng/địa chỉ IP. Ví dụ: không quá 10 yêu cầu giải mã mỗi phút.
    - Giám sát tài nguyên: Cảnh báo khi CPU/memory của TA hoặc Backend tăng đột biến.

3.6. Elevation of Privilege.

– Kịch bản tấn công cụ thể: Một lỗi trong logic phân tích chính sách của Backend cho phép chính sách "family_relation:daughter" cũng khớp với thuộc tính doctor (ví dụ: lỗi parsing chuỗi).
– Tác nhân (Actor): Lỗi lập trình (bug) trong hệ thống.
– Mục tiêu (Target): Logic xử lý chính sách truy cập.
– Tác động (Impact): Một người dùng (BS. Bích) có thể truy cập vào các tài nguyên vốn chỉ dành cho người dùng khác (Chị Mai), vi phạm mô hình kiểm soát truy cập.
– Biện pháp giảm thiểu (Mitigation):

- Kiểm thử đơn vị (Unit Test): Viết các bộ test case chi tiết cho hàm xử lý chính sách với nhiều trường hợp biên (chính sách phức tạp, rỗng, sai cú pháp).
- Review mã nguồn (Code Review): Yêu cầu một thành viên khác trong nhóm kiểm tra lại logic của phần xử lý chính sách.

3.7. Spoofing / Tampering.

– Kịch bản tấn công cụ thể: Một kẻ tấn công có quyền truy cập vào cơ sở dữ liệu của TA (nơi lưu các "gói khóa") hoặc chặn được gói tin trên đường truyền. Kẻ tấn công cố gắng sửa đổi các thành phần trong gói khóa (ví dụ: thay đổi policy trong metadata để cấp quyền cho chính mình, hoặc thay đổi abe_key_hex để gây lỗi từ chối dịch vụ).

- Tác nhân: Kẻ tấn công nội bộ/bên ngoài có quyền truy cập vào kênh truyền thông hoặc nơi lưu trữ của TA.
- Mục tiêu: Gói khóa (Key Package) được lưu trữ trên TA.
- Tác động: Nếu thành công, kẻ tấn công có thể nâng cao đặc quyền, truy cập trái phép dữ liệu, hoặc gây ra lỗi từ chối dịch vụ.

Biện pháp giảm thiểu (Mitigation):

- Niêm phong Toàn vẹn bằng Chữ ký PQC (Integrity Seal via PQC Signature): Đây là biện pháp phòng thủ cốt lõi. Toàn bộ gói metadata (bao gồm policy, user_id, timestamp, và abe_key_hex) đã được ký điện tử bằng thuật toán Dilithium2. Tại thời điểm giải mã, bước đầu tiên luôn là xác thực chữ ký này. Bất kỳ sự thay đổi nào, dù là nhỏ nhất, trên metadata cũng sẽ làm cho chữ ký trở nên không hợp lệ. Quá trình giải mã sẽ ngay lập tức bị hủy bỏ, ngăn chặn hoàn toàn cuộc tấn công này.

4. Kiến trúc Giải pháp.

– Để đáp ứng các yêu cầu bảo mật nghiêm ngặt trong khi cung cấp kiểm soát truy cập linh hoạt và chi tiết cho EHRs nhạy cảm trong đám mây, nhóm em đề xuất một hệ thống dựa trên Mã hóa dựa trên chính sách mã hóa (CP-ABE) làm cơ chế mật mã cốt lõi. Hệ thống bao gồm các thành phần sau:

4.1. Công cụ mật mã CP-ABE (abe_core.py):

Đây là module trung tâm, cung cấp một lớp trừu tượng hóa mạnh mẽ cho tất cả các hoạt động mật mã phức tạp. Nó được thiết kế theo kiến trúc "Vỏ bọc Tương thích" (Adapter Pattern), bao gồm:

- **Lớp lõi ABECoreGCM:** Chịu trách nhiệm thực thi các nguyên thủy mật mã ở tầng thấp, bao gồm CP-ABE, AES-GCM, và chữ ký hậu lượng tử Dilithium2.
- **Lớp vỏ bọc ABECompatWrapper:** Cung cấp một giao diện (API) đơn giản và thống nhất cho các dịch vụ khác (như Backend) gọi đến, ẩn đi sự phức tạp của việc xử lý mật mã bên trong.

Các chức năng chính được cung cấp bao gồm:

- **Khởi tạo hệ thống (setup()):** Được gọi bởi TA để tạo ra Khóa Công khai (PK) và Khóa Chủ (MK) cho toàn hệ thống.
- **Tạo khóa bí mật (keygen()):** Được gọi bởi TA để tạo Khóa Bí mật (SK) cho người dùng dựa trên một tập thuộc tính.
- **Mã hóa Toàn diện (encrypt()):** Đây là một hàm lai bậc cao. Khi được Backend gọi, nó sẽ tự động thực hiện một chuỗi các thao tác:
    - Tạo một khóa phiên đối xứng tạm thời.
    - Mã hóa khóa phiên này bằng CP-ABE theo chính sách được cung cấp.
    - Sử dụng khóa phiên để mã hóa dữ liệu EHR bằng AES-GCM, tạo ra bản mã và thẻ xác thực (tag).
    - Tạo một gói metadata chứa thông tin ngữ cảnh (policy, user_id, timestamp, và khóa phiên đã mã hóa ABE).
    - Ký điện tử lên toàn bộ gói metadata bằng chữ ký hậu lượng tử Dilithium2.
    - Trả về một "gói dữ liệu mã hóa hoàn chỉnh" chứa tất cả các thành phần trên, sẵn sàng để được phân tách và lưu trữ.
- **Giải mã Toàn diện (decrypt()):** Khi nhận được một "gói dữ liệu mã hóa hoàn chỉnh", hàm này sẽ tự động thực hiện các bước kiểm tra an ninh theo đúng thứ tự:
    - Xác thực chữ ký Dilithium2 của metadata.
    - Kiểm tra timestamp trong metadata để chống tấn công phát lại.

- Giải mã ABE để phục hồi khóa phiên.
- Giải mã AES-GCM và xác thực tag để khôi phục dữ liệu gốc.

- **Thư viện và Đặc điểm kỹ thuật:** Hệ thống được xây dựng trên nền tảng Python, sử dụng các thư viện đã được kiểm chứng: Charm-Crypto cho CP-ABE (lược đồ BSW07, đường cong SS512), Cryptodome cho AES-GCM, và oqs-python cho chữ ký hậu lượng tử Dilithium2. Việc serialize khóa được thực hiện an toàn bằng objectToBytes và base64, loại bỏ hoàn toàn rủi ro từ pickle.

4.2. Cơ quan tin cậy (Trusted Authority - TA):

– Mô tả: Trusted Authority (TA) là một thành phần cốt lõi, độc lập và chuyên biệt trong hệ thống, chịu trách nhiệm quản lý vòng đời của các khóa mật mã dựa trên thuộc tính (ABE). Vai trò chính của TA bao gồm việc khởi tạo các tham số toàn cục của hệ thống ABE (Khóa Công khai - PK và Khóa Chủ - MK), bảo vệ nghiêm ngặt Khóa Chủ, tạo Khóa Bí mật (SK) cho người dùng, và lưu trữ an toàn các "gói khóa" (key package) cần thiết cho việc giải mã dữ liệu. TA được triển khai như một dịch vụ riêng biệt, cách ly khỏi các thành phần khác của hệ thống EHR để đảm bảo an ninh tối đa cho các hoạt động nhạy cảm liên quan đến khóa.

– Chức năng:

– Khởi tạo Hệ thống ABE (/setup):

- Thực hiện việc tạo ra cặp khóa chính của hệ thống ABE: Khóa Công khai (PK) và Khóa Chủ (MK).
- Bảo vệ Khóa Chủ (MK): MK gốc được mã hóa bằng một Khóa Khách hàng Quản lý (Customer Managed Key - CMK) trên dịch vụ AWS Key Management Service (KMS). Ciphertext của MK (kết quả của việc mã hóa bằng KMS) sau đó được lưu trữ an toàn trong dịch vụ AWS Secrets Manager.
- Khóa Công khai (PK) được lưu trữ cục bộ trên TA và có thể được cung cấp cho các thành phần khác trong hệ thống (ví dụ: qua endpoint /get_public_key).

– Tạo Khóa Bí mật Người dùng (/keygen):

- Nhận yêu cầu tạo Khóa Bí mật (SK) từ Hệ thống Backend (ví dụ: Backend EHR Service), kèm theo danh sách các thuộc tính của người dùng (có thể bao gồm cả thuộc tính thời hạn).

– Để tạo SK, TA thực hiện quy trình an toàn:
  - Truy xuất ciphertext của MK từ AWS Secrets Manager.
  - Gửi ciphertext này đến AWS KMS để giải mã, nhận lại MK plaintext (chỉ tồn tại trong bộ nhớ tạm thời của TA).
  - Sử dụng MK plaintext và PK của hệ thống để tạo ra SK cho người dùng dựa trên các thuộc tính được cung cấp.
  - Trả về SK đã được serialize và mã hóa base64 cho Hệ thống Backend.
– Lưu trữ Gói Khóa (/store_key_package):
  - Cung cấp API để Hệ thống Backend gửi "gói khóa" (key_package) lên lưu trữ.
  - key_package chứa các thành phần mật mã cần thiết cho việc giải mã dữ liệu EHR sau này, bao gồm: IV và tag xác thực của mã hóa AES-GCM, cùng với metadata (chứa khóa đối xứng đã được mã hóa ABE, chính sách ABE, thông tin người dùng, timestamp) và chữ ký Hậu Lượng tử (PQC) của metadata đó.
  - TA lưu trữ key_package này (dưới dạng JSON) vào một kho lưu trữ cục bộ (ví dụ: file JSON trên server TA), liên kết với một record_id duy nhất.
– Truy xuất Gói Khóa (/get_key_package):
  - Cung cấp API để Hệ thống Backend yêu cầu truy xuất key_package đã lưu trữ, dựa trên record_id.
  - TA trả về key_package (dưới dạng chuỗi JSON) cho Backend để phục vụ quá trình giải mã dữ liệu EHR.
  - Bảo mật và Ghi nhật ký:
  - Tất cả các hoạt động nhạy cảm của TA nên được ghi nhật ký kiểm toán chi tiết.
  - Giao tiếp mạng với TA phải được bảo vệ bằng TLS 1.3.
  - Cơ chế xác thực mạnh mẽ nên được áp dụng cho các API của TA (ví dụ: JWT cho các dịch vụ nội bộ, hoặc Mutual TLS).
– Công nghệ và triển khai đề xuất:

- Triển khai TA như một microservice riêng biệt, sử dụng Flask và các thư viện mật mã (Charm-Crypto, PyCryptodome, OQS-Python).
- Chạy trong một container Docker để đảm bảo tính nhất quán và dễ triển khai.
- Khi triển khai lên cloud (ví dụ: AWS EC2), sử dụng IAM Roles để cấp quyền truy cập an toàn đến các dịch vụ AWS (KMS, Secrets Manager) thay vì lưu trữ access keys.
- Được bảo vệ bởi các biện pháp an ninh mạng nghiêm ngặt như Security Groups và Network ACLs, chỉ cho phép truy cập từ các thành phần hệ thống được tin cậy.

4.3. Cổng ủy quyền (Authorization Gateway):

– Xác thực: Người dùng xác thực thông qua JWT sử dụng Flask-JWT-Extended, sau khi đăng nhập bằng mật khẩu được băm hoặc OAuth2.

– API quản lý khóa:

- POST /auth/keygen: Nhận JWT, trích xuất thuộc tính người dùng, sau đó gửi yêu cầu tạo khóa đến TA. Nhận SKU từ TA và phân phối cho người dùng.
- POST /auth/revoke: Điểm cuối của quản trị viên để thông báo cho TA về việc thu hồi thuộc tính người dùng, để TA có thể quản lý vòng đời khóa (tái mã hóa nếu cần, mặc dù cơ chế thu hồi chi tiết hơn cần được xem xét).

– Quản lý vòng đời khóa:

- Tạo: Khóa bí mật chính (MSK/MK của TA) được tạo bởi TA trong quá trình khởi tạo hệ thống. Khóa bí mật người dùng (SKU) được tạo bởi TA dựa trên các thuộc tính của người dùng.
- Phân phối: SKU được phân phối an toàn cho người dùng thông qua TA và Cổng ủy quyền sau khi xác thực thành công. Người dùng chịu trách nhiệm bảo vệ SKU trên thiết bị của họ.
- Luân chuyển/Hủy: Các kế hoạch luân chuyển khóa định kỳ cho MSK của TA và cơ chế thu hồi/hủy khóa người dùng cần được TA quản lý.
- Giao tiếp an toàn: Tất cả giao tiếp được mã hóa qua TLS 1.3, bao gồm API nội bộ và truy cập đám mây. Sử dụng bộ mã hóa TLS mạnh và áp dụng chứng chỉ hợp lệ.

- Ghi nhật ký kiểm toán: Tất cả các hoạt động khóa và các sự kiện mã hóa/giải mã được ghi nhật ký với dấu thời gian, ID người dùng và hành động. Nhật ký này sẽ được bảo vệ tính toàn vẹn và không từ chối.
- Xử lý lỗi an toàn: Triển khai cơ chế xử lý lỗi không tiết lộ thông tin nhạy cảm.

4.4. Cơ sở dữ liệu siêu dữ liệu PostgreSQL:

– Bảng:

- users: id, username, email, password hash, role, department, created_at.
- ehr_files: id, filename, s3 key, owner id, policy, uploaded_at. Lưu ý: Các cột liên quan đến CTdk (khóa phiên AES đã mã hóa CP-ABE) và chữ ký số của nó đã được xóa khỏi bảng này. Chúng sẽ được lưu trữ bởi TA.
- Logic kiểm soát truy cập: Một hàm lưu trữ can_access_file(user_id, record_id) kiểm tra xem các thuộc tính người dùng có thỏa mãn chính sách tệp được lưu trữ (ở định dạng JSONB) hay không.
- Hiệu suất: Sử dụng lập chỉ mục GIN trên trường chính sách để kiểm tra truy cập hiệu quả.

4.5. Cổng API RESTful:

– Mô tả:

- Cổng API RESTful, được triển khai trong ehr.py, là thành phần trung tâm xử lý các yêu cầu từ người dùng liên quan đến Hồ sơ Sức khỏe Điện tử (EHR). Nó chịu trách nhiệm điều phối quy trình mã hóa khi tải lên và giải mã khi tải xuống, tương tác với Trusted Authority (TA) để quản lý các thành phần khóa, lưu trữ dữ liệu EHR đã mã hóa trên Amazon S3, và quản lý metadata của file trong cơ sở dữ liệu (ví dụ: PostgreSQL).

– Chức năng chính:

- Tải lên EHR (POST /api/ehr/upload):
- Tiếp nhận và Xác thực: Nhận yêu cầu từ người dùng bao gồm file EHR (plaintext), chuỗi chính sách truy cập ABE, và JWT để xác thực. Backend xác thực JWT, lấy user_id.
- Chuẩn bị Mã hóa: Đọc nội dung file, chuẩn hóa chuỗi chính sách ABE, tạo record_id và s3_key.
- Lấy Khóa Công khai (PK): Gọi TAClient.get_public_key() để lấy PK của hệ thống ABE từ TA.

- Thực hiện Mã hóa Lai Toàn diện: Gọi hàm abe.encrypt(PK, file_data, policy, user_id) từ thư viện abe_core_v2. Quá trình này (diễn ra bên trong abe.encrypt) bao gồm:
    - Tạo một khóa đối xứng (sym_key) tạm thời.
    - Mã hóa sym_key bằng CP-ABE theo policy đã cho, tạo ra abe_encrypted_key.
    - Dẫn xuất khóa AES từ sym_key và mã hóa file_data bằng AES-GCM, tạo ra encrypted_ehr_data (ciphertext AES), IV, và tag xác thực.
    - Tạo metadata (chứa policy, user_id, timestamp, abe_encrypted_key dạng hex) và ký metadata này bằng chữ ký Hậu Lượng tử (PQC) để tạo sig_pqc.
    - Kết quả là một "gói dữ liệu mã hóa hoàn chỉnh" (full_encrypted_package) chứa: iv, tag, data (là encrypted_ehr_data), metadata, và sig (là sig_pqc).
- Phân tách và Phân phối Lưu trữ:
    - encrypted_ehr_data (ciphertext AES) được tách ra từ full_encrypted_package.
    - Phần còn lại (iv, tag, metadata, sig_pqc) tạo thành key_package.
    - Backend gọi TAClient.store_key_package(record_id, key_package) để gửi key_package cho TA lưu trữ.
    - Backend tải encrypted_ehr_data lên Amazon S3 thông qua hàm upload_to_s3().
- Lưu Metadata Tham chiếu: Lưu thông tin như record_id, tên file, s3_key, policy gốc, và owner_id vào cơ sở dữ liệu PostgreSQL.
- Phản hồi: Trả về record_id và thông báo thành công cho người dùng.

– Tải xuống và Giải mã EHR (POST /api/ehr/download/<record_id>):

- Tiếp nhận và Xác thực: Nhận yêu cầu từ người dùng (đã xác thực JWT), bao gồm record_id và Khóa Bí Mật (SK) của người dùng (dưới dạng base64).
- Truy xuất Thông tin: Lấy metadata của file EHR từ PostgreSQL dựa trên record_id.
- Thu thập Thành phần Giải mã:
    - Gọi TAClient.get_key_package(record_id) để lấy key_package từ TA.

        - Gọi download_from_s3() để tải encrypted_ehr_data (ciphertext AES) từ S3.
        - Gọi TAClient.get_public_key() để lấy PK từ TA.
- Chuẩn bị Giải mã: Tái tạo full_package_for_decryption bằng cách kết hợp key_package và encrypted_ehr_data. Deserialize PK (từ TA) và SK (từ người dùng).
- Thực hiện Giải mã Toàn diện: Gọi hàm abe.decrypt(PK, SK, full_package_for_decryption). Quá trình này (diễn ra bên trong abe.decrypt) bao gồm:
    - Xác thực chữ ký PQC của metadata (trong key_package) bằng khóa công khai PQC của ABE engine. Nếu thất bại, dừng và báo lỗi.
    - Kiểm tra timestamp trong metadata.
    - Giải mã abe_encrypted_key (trong metadata) bằng CP-ABE (sử dụng PK hệ thống và SK của người dùng) để thu được sym_key gốc. Nếu thuộc tính không khớp, dừng và báo lỗi.
    - Dẫn xuất lại khóa AES từ sym_key và giải mã encrypted_ehr_data bằng AES-GCM (sử dụng IV và tag từ key_package). Nếu tag không khớp (lỗi toàn vẹn), dừng và báo lỗi.
- Trả Kết quả:
    - Nếu giải mã thành công, trả về nội dung file gốc (plaintext) cho người dùng.
    - Nếu thất bại ở bất kỳ bước nào, trả về thông báo lỗi thích hợp (ví dụ: 403 Access Denied, 404 Not Found, 500 Internal Server Error).

4.6. Lưu trữ đối tượng đám mây (ví dụ: AWS S3):

– Định dạng đường dẫn: s3://ehr-bucket/{user_id}/{record_id}.enc

– Bảo mật:

- Các chính sách IAM hạn chế quyền truy cập chỉ cho cổng API.
- Mã hóa phía máy chủ (SSE-KMS) được bật để lưu trữ.

4.7. Cơ chế thu hồi quyền truy cập: Phương pháp thu hồi dựa trên thời hạn.

– Khóa Bí mật (SK) Chứa Thuộc tính do Quản trị viên Gán:

- Khi một Khóa Bí mật (SK) được tạo cho người dùng bởi Trusted Authority (TA) (thông qua yêu cầu từ Hệ thống Backend), SK này sẽ chứa các thuộc tính được Quản trị viên Nền tảng gán cho người dùng đó.
- Các thuộc tính này có thể bao gồm vai trò (ví dụ: DOCTOR), chuyên môn (CARDIOLOGY). Việc đưa thuộc tính thời hạn này vào danh sách thuộc tính khi yêu cầu tạo SK là trách nhiệm của Hệ thống Backend, dựa trên quy trình quản lý người dùng.

– Chính sách truy cập (Policy) của dữ liệu EHR phải bao gồm điều kiện Thời hạn:
  - Khi người sở hữu dữ liệu (ví dụ: Ông An) mã hóa một tệp EHR thông qua Hệ thống Backend, chính sách truy cập ABE mà họ đặt ra phải được thiết kế để bao gồm thuộc tính thời hạn mong muốn nếu muốn kiểm soát truy cập dựa trên thời gian.
  - Ví dụ, một chính sách ABE có thể là: (DOCTOR AND CARDIOLOGY). Điều này có nghĩa là để giải mã được tệp này, SK của người dùng phải chứa cả ba thuộc tính này.
– Kiểm tra tính thời sự của dữ liệu (Metadata Timestamp):
  - Trong quá trình mã hóa, hàm abe.encrypt (cụ thể là ABECoreGCM.encrypt) ghi lại một dấu thời gian (timestamp) của thời điểm mã hóa vào metadata của "gói khóa" (key_package).
  - Khi giải mã (trong ABECoreGCM.decrypt), hệ thống thực hiện một bước kiểm tra:
  - if datetime.now(timezone.utc) > ts + timedelta(days=7): // ts là timestamp khi file được mã hóa
  - raise ValueError("Dữ liệu đã hết hạn: timestamp quá 7 ngày.")
  - Cơ chế này đảm bảo rằng dữ liệu không thể được giải mã nếu nó đã trở nên quá cũ so với thời điểm tạo ra (ví dụ, giới hạn truy cập dữ liệu trong vòng 7 ngày sau khi mã hóa). Đây là một biện pháp kiểm soát vòng đời của dữ liệu đã mã hóa, không trực tiếp liên quan đến việc thu hồi SK của người dùng.
– Thực thi Quyền Truy cập tại Thời điểm Giải mã:
– Một người dùng chỉ có thể giải mã thành công một tệp EHR nếu:
– Khớp thuộc tính ABE: Tất cả các thuộc tính được liệt kê trong chính sách ABE của tệp phải có mặt trong Khóa Bí mật (SK) của người dùng. Quá trình so khớp này được

thực hiện bởi thuật toán CP-ABE (self.abe.decrypt trong ABECoreGCM). Nếu không khớp, lỗi DecryptionPolicyError được trả về.

- Dữ liệu không quá cũ: Dữ liệu phải vượt qua kiểm tra timestamp metadata như mô tả ở mục 3.
- Luồng thu hồi quyền truy cập trong thực tế (Dựa trên quản lý SK và chính sách File):
- Kịch bản:
    - BS. Bích được Hệ thống Backend yêu cầu TA cấp SK có chứa các thuộc tính ['DOCTOR'].
    - Ông An (người sở hữu dữ liệu) mã hóa một hồ sơ y tế với chính sách ABE là (DOCTOR).
- Cơ chế Thu hồi Chính yếu - Quản lý Cấp phát SK:
    - Việc thu hồi quyền truy cập thực sự đối với BS. Bích (ví dụ, khi bà nghỉ việc hoặc hết thời hạn truy cập theo quy định) được thực thi chủ yếu thông qua quy trình quản lý và tái cấp phát Khóa Bí mật (SK) của Hệ thống Backend và TA.
    - Khi SK của BS. Bích không còn cho phép truy cập các file mới (do không khớp thuộc tính thời hạn trong chính sách của file mới), bà sẽ cần yêu cầu một SK mới.
    - Tại thời điểm này, Hệ thống Backend (dựa trên thông tin từ Quản trị viên Nền tảng) sẽ quyết định:
    - Nếu BS. Bích vẫn được phép truy cập: Backend sẽ yêu cầu TA cấp một SK mới với thuộc tính thời hạn được cập nhật.
    - Nếu BS. Bích không còn được phép truy cập (ví dụ: đã nghỉ việc): Backend sẽ từ chối yêu cầu cấp SK mới. Điều này ngăn chặn BS. Bích truy cập vào các file được bảo vệ bằng chính sách yêu cầu thuộc tính thời hạn mới.
- Phân tích Ưu điểm và Nhược điểm:
- Ưu điểm:
    - Không cần tái mã hóa dữ liệu để áp dụng "thời hạn" cho các file mới (chỉ cần người tạo file đặt chính sách ABE có chứa thuộc tính thời hạn).
    - Logic của TA và ABE engine hiện tại không cần thay đổi để hỗ trợ việc khớp thuộc tính thời gian nếu nó được định nghĩa trong chính sách ABE.

- Cơ chế kiểm tra metadata timestamp giúp hạn chế truy cập vào dữ liệu đã quá cũ.

– Nhược điểm và Rủi ro cần chấp nhận:
  - Độ trễ Thu hồi (Revocation Latency) đối với các file có chính sách cũ: Nếu một người dùng bị thu hồi quyền (ví dụ: nghỉ việc) nhưng SK hiện tại của họ vẫn chứa các thuộc tính (bao gồm cả thuộc tính thời hạn cũ) khớp với chính sách của các file đã được mã hóa trước đó, họ vẫn có thể truy cập các file đó cho đến khi thuộc tính thời hạn trong chính sách của file đó không còn "ý nghĩa" nữa (ví dụ, nếu các file mới đều yêu cầu một thuộc tính thời hạn mới hơn). Đây là "cửa sổ rủi ro".
  - Hiệu quả thu hồi phụ thuộc vào thiết kế Chính sách ABE của File: Việc thu hồi dựa trên thời hạn chỉ có tác dụng nếu người tạo dữ liệu chủ động đưa các thuộc tính thời hạn vào chính sách ABE của file. Nếu chính sách chỉ dựa trên vai trò, thì thuộc tính thời hạn trong SK sẽ không có tác dụng ngăn cản truy cập file đó.
  - Quản lý thuộc tính thời gian trong chính sách ABE có thể phức tạp: Người tạo dữ liệu cần ý thức được việc đặt đúng thuộc tính thời hạn trong chính sách.

4.8. Nguyên tắc thiết kế bảo mật:

– Phòng thủ chiều sâu (Defense-in-depth): Áp dụng nhiều lớp kiểm soát bảo mật (xác thực, kiểm soát truy cập, mã hóa, kiểm tra đầu vào, chữ ký số) để tăng cường khả năng chống chịu của hệ thống.

– Đặc quyền tối thiểu (Least Privilege): Đảm bảo mỗi thành phần và người dùng chỉ có các quyền tối thiểu cần thiết để thực hiện chức năng của mình.

4.9 Lựa chọn chữ ký số - đảm bảo toàn vẹn hậu lượng tử với CRYSTALS-Dilithium.

Trong kiến trúc của "MyHealth Vault", tính toàn vẹn và xác thực nguồn gốc của các siêu dữ liệu nhạy cảm là tối quan trọng. Thay vì chỉ ký lên một thành phần riêng lẻ như bản mã của khóa phiên, hệ thống áp dụng một phương pháp bảo vệ toàn diện hơn: **ký điện tử lên toàn bộ gói metadata** (bao gồm chính sách, định danh người dùng, dấu thời gian, và khóa phiên đã mã hóa). Một sự thay đổi trái phép trên bất kỳ thông tin ngữ cảnh nào cũng sẽ làm chữ ký bị vô hiệu, khiến cho toàn bộ quá trình giải mã thất bại. Để thực hiện việc này, dự án quyết định lựa chọn CRYSTALS-Dilithium. Một sự thay đổi trái phép trên đối tượng này có thể làm toàn bộ quá trình giải mã thất

bại hoặc bị qua mặt. Theo truyền thống, các thuật toán như ECDSA hoặc RSA-PSS được sử dụng. Tuy nhiên, với tầm nhìn dài hạn, dự án quyết định lựa chọn CRYSTALS-Dilithium, một thuật toán chữ ký số hậu-lượng tử đã được NIST chuẩn hóa. Lựa chọn này được dựa trên các phân tích sau:

4.2.1. Mô hình đe dọa và an ninh dài hạn.

- Dữ liệu y tế (EHR) có một đặc tính riêng biệt: giá trị của nó tồn tại trong một thời gian rất dài, có thể là toàn bộ cuộc đời của một bệnh nhân. Điều này đặt ra một yêu cầu bảo mật dài hạn nghiêm ngặt, vượt xa các loại dữ liệu thương mại thông thường.
- Nhóm em xác định mô hình đe dọa "Thu thập ngay, Phá vỡ sau" (Harvest Now, Decrypt Later) là một rủi ro hiện hữu. Kẻ tấn công có thể thu thập các bản tin đã được mã hóa và ký số ngày hôm nay. Với sự phát triển của máy tính lượng tử trong tương lai, các thuật toán kinh điển như RSA và ECDSA sẽ bị phá vỡ. Khi đó, không chỉ tính bảo mật của dữ liệu bị đe dọa, mà tính toàn vẹn và khả năng chống chối bỏ cũng sụp đổ. Kẻ tấn công có thể giả mạo chữ ký trên các hồ sơ y tế cũ, gây ra những hậu quả pháp lý và y khoa thảm khốc.
- Việc sử dụng Dilithium là một biện pháp phòng ngừa chiến lược. Bằng cách triển khai một thuật toán được cho là an toàn trước các cuộc tấn công từ cả máy tính cổ điển và máy tính lượng tử, chúng tôi đảm bảo rằng tính toàn vẹn của dữ liệu được ký hôm nay vẫn sẽ được duy trì trong nhiều thập kỷ tới.

4.9.2. Đánh đổi về hiệu năng và chi phí.

- Quyết định sử dụng Dilithium không phải là một lựa chọn không có đánh đổi. Nhóm em nhận thức rõ ràng về sự chênh lệch hiệu năng so với các thuật toán cổ điển:
- Kích thước Chữ ký: Đây là điểm đánh đổi lớn nhất. Một chữ ký Dilithium (cấp độ an ninh 2) có kích thước khoảng 2.4KB. Trong khi đó, một chữ ký ECDSA (sử dụng đường cong secp256r1) chỉ chiếm khoảng 70 bytes. Kích thước này lớn hơn gần 35 lần.
- Tác động:
    - Chi phí lưu trữ: Mỗi bản ghi trong cơ sở dữ liệu (lưu chữ ký của CTdk) sẽ tốn thêm khoảng 2.3KB dung lượng. Với hàng triệu hồ sơ bệnh án, chi phí lưu trữ tổng thể sẽ tăng lên.
    - Băng thông mạng: Việc truyền tải chữ ký giữa các microservice (ví dụ: từ TA đến Backend) và trong các quá trình sao lưu sẽ tốn nhiều băng thông hơn.
    - Tuy nhiên, sau khi cân nhắc, dự án chấp nhận sự đánh đổi này. Với một ứng dụng yêu cầu an ninh cao như EHR, việc đảm bảo tính toàn vẹn dài hạn là ưu tiên hàng đầu. Sự gia tăng về chi phí lưu trữ và băng thông được xem là hợp lý so với rủi ro an ninh mà nó giúp giảm thiểu.

4.9.3. Lựa chọn thư viện triển khai.

- Nguyên tắc cốt lõi trong mật mã học là không bao giờ tự phát minh hoặc tự triển khai các thuật toán. Để tích hợp Dilithium, nhóm em lựa chọn sử dụng thư viện liboqs (The Open Quantum Safe project).
- Lý do: liboqs là một dự án mã nguồn mở uy tín, cung cấp một bộ sưu tập các thuật toán mật mã hậu-lượng tử đã được các chuyên gia đánh giá. Nó cung cấp một API nhất quán để làm việc với các thuật toán này, bao gồm cả các cài đặt tham chiếu (reference implementation) sạch và đã được kiểm định từ dự án pqclean.
- Triển khai cụ thể: Nhóm em sẽ sử dụng binding Python của liboqs (oqs-python) để tích hợp thuật toán Dilithium vào các service viết bằng Flask của hệ thống. Điều này đảm bảo rằng nhóm em đang sử dụng một phiên bản cài đặt đã được kiểm thử rộng rãi, giảm thiểu rủi ro từ các lỗi triển khai.

4.10. Cơ chế chống thông đồng.

- Một trong những ưu điểm bảo mật quan trọng của lược đồ Mã hóa Dựa trên Thuộc tính Ciphertext-Policy (CP-ABE), cụ thể là biến thể Bethencourt, Sahai, và Waters (BSW07) được sử dụng trong hệ thống, là khả năng Chống Thông đồng (Collusion Resistance). Đặc tính này ngăn chặn việc nhiều người dùng (kể cả khi họ có các tập thuộc tính khác nhau) kết hợp các Khóa Bí mật (SK) của họ lại với nhau để tạo ra một khóa mới có khả năng giải mã những bản mã mà không ai trong số họ có thể giải mã riêng lẻ.
- Nguyên tắc hoạt động (Randomization):
    - Khả năng chống thông đồng trong lược đồ CP-ABE BSW07 đạt được bằng cách cá dạng bao gồm một thành phần chính D = g^((α + r_i)/β) và các thành phần phụ D_x = g^(r_i) * H(x)^(r_x,i) cho mỗi thuộc tính x mà người dùng sở hữu. Trong đó:
        - α và β là các thành phần của Khóa Chủ (MK).
        - g là một generator của nhóm mật mã.
        - r_i là số ngẫu nhiên duy nhất cho người dùng i.
        - H(x) là hàm băm ánh xạ thuộc tính x vào một phần tử của nhóm.
        - r_x,i cũng có thể là một yếu tố ngẫu nhiên liên quan đến thuộc tính x và người dùng i.
- Ngăn chặn Thông đồng:
    - Do mỗi Khóa Bí mật SK_i được "đóng dấu" bằng một giá trị ngẫu nhiên r_i riêng biệt, nhiên r_2) cố gắng kết hợp các phần của khóa riêng của họ,

họ không thể dễ dàng loại bỏ các thành r_1 và r_2 để tạo ra một khóa mới có thể giải mã một bản mã mà chính sách của nó yêu cầu một tập hợp thuộc tính mà cả A và B đều không đáp ứng riêng lẻ. Các yếu tố ngẫu nhiên này đảm bảo rằng các khóa không thể "trộn lẫn" một cách tùy tiện.

- Ý nghĩa ảo mật:
  - Khả năng chống thông đồng là cực kỳ quan trọng trong các hệ thống chia sẻ dữ liệu dựa trên thuộc tính. Nó đảm bảo rằng người dùng không thể gian lận hệ thống bằng cách hợp tác để truy cập trái phép vào thông tin mà họ không được quyền xem xét riêng lẻ, qua đó duy trì tính toàn vẹn của cơ chế kiểm soát truy cập.

5. Quá trình phát triển & kiểm thử.

- Quá trình phát triển và kiểm thử sẽ tuân thủ các nguyên tắc nghiêm ngặt để đảm bảo chất lượng và bảo mật của giải pháp, trong đó:
  - **Môi trường triển khai:** Phát triển và kiểm thử trong một môi trường cô lập (ví dụ: máy ảo cục bộ, vùng chứa Docker hoặc vùng chứa đám mây). Sử dụng kiểm soát phiên bản và mạng ảo để mô phỏng các tình huống thực tế. Đảm bảo các thư viện mật mã và framework được cập nhật.
  - **Kiểm thử chức năng:**
    - Tạo các bài kiểm thử đơn vị và tích hợp cho các chức năng cốt lõi.
    - Xác minh rằng mã hóa và giải mã (qua cơ chế mã hóa lai) tạo ra kết quả mong đợi.
    - Kiểm tra xem xác thực, trao đổi khóa và xử lý lỗi có hoạt động chính xác trong các đầu vào bình thường và trường hợp biên.
    - Minh họa tất cả các chức năng hệ thống trong một kịch bản demo hoặc giao diện người dùng.
    - Các kịch bản thử nghiệm sẽ xác nhận rằng hệ thống hoạt động theo thiết kế: Phân vùng truy cập, Giảm thiểu đặc quyền, Mã hóa lai được cải thiện hiệu suất mà không làm giảm độ bảo mật, Khả năng tương tác với các hệ thống lưu trữ khác nhau.
    - **Kiểm thử chức năng chữ ký số:** Xác minh việc tạo và xác minh chữ ký số **Dilithium2** cho **gói `metadata`** hoạt động chính xác trong quá trình tải lên và tải xuống. Đảm bảo rằng chữ ký được tạo ra đúng cách bởi Backend và được xác thực thành công trước khi tiến hành giải mã.
- **Kiểm thử bảo mật:**

- Thực hiện đánh giá mã và phân tích tĩnh (công cụ như SonarQube, Flawfinder/Bandit) để phát hiện các thực hành không an toàn (khóa cứng, thuật toán yếu, lỗi tiêm).
- Đối với các ứng dụng nối mạng, sử dụng phân tích động (ví dụ: OWASP ZAP, Burp Suite) để quét các lỗ hổng phổ biến.
- Cân nhắc fuzzing đầu vào hoặc thực hiện các bài kiểm tra thâm nhập cơ bản (ví dụ: cố gắng brute-force khóa hoặc mô phỏng tấn công trung gian).
- Các thử nghiệm bảo mật sẽ đặc biệt tập trung vào việc xác thực các chiến lược giảm thiểu cho các mối đe dọa đã xác định trong phần phân tích rủi ro (ví dụ: thử nghiệm để làm lộ thông tin nhạy cảm, cố gắng giả mạo dữ liệu hoặc thực hiện các cuộc tấn công từ chối dịch vụ).
- **Kiểm thử TA:** Đặc biệt tập trung vào kiểm thử các API của TA, bao gồm xác thực, ủy quyền, và các cuộc tấn công tiêm. Kiểm tra khả năng chịu lỗi của TA và cách nó xử lý các yêu cầu không hợp lệ. Kiểm tra xem liệu có thể truy cập Khóa chính (MK) của TA nếu không thông qua HSM/KMS hay không.
- **Kiểm thử chữ ký số:** Kiểm tra các trường hợp khi chữ ký số bị giả mạo hoặc không khớp với bản mã khóa phiên. Đảm bảo hệ thống phát hiện và phản ứng đúng với các bản mã khóa phiên có chữ ký không hợp lệ.
- Giảm thiểu mọi vấn đề được phát hiện.

- **Kiểm thử hiệu suất:**
  - Đo điểm chuẩn các hoạt động mật mã và thông lượng hệ thống. Ví dụ, sử dụng lệnh openssl speed để đo hiệu suất mã hóa/giải mã.
  - Đo độ trễ đầu cuối dưới tải (ví dụ: sử dụng Apache JMeter hoặc tập lệnh tùy chỉnh) để đảm bảo giải pháp đáp ứng mọi yêu cầu về hiệu quả.
  - Phân tích và tối ưu hóa các điểm nóng (ví dụ: sử dụng tăng tốc phần cứng hoặc hoạt động hàng loạt nếu cần).
  - Dựa trên các bài kiểm tra thực hiện trên máy tính cấu hình Intel Core i7, 16GB RAM, thời gian trung bình cho các hoạt động là: Thiết lập (Setup): 250 ms, Tạo khóa người dùng: 180 ms, Mã hóa (1KB dữ liệu): 120 ms (đã bao gồm mã hóa AES và CP-ABE), Giải mã (1KB dữ liệu): 150 ms (đã bao gồm giải mã CP-ABE và AES).

- **Triển khai:**
  - Chuẩn bị kế hoạch triển khai (vùng chứa/hình ảnh, dịch vụ đám mây hoặc máy chủ).

- Tài liệu các nền tảng hoặc thư viện cần thiết để người khác có thể tái tạo bản demo.
- Sử dụng cấu hình an toàn (vô hiệu hóa các cổng không cần thiết, thực thi bộ mã hóa TLS mạnh).
- Trình diễn thiết lập cuối cùng trong một kịch bản thực tế (ví dụ: dữ liệu được mã hóa truyền giữa máy khách và máy chủ).

6. Hướng phát triển và hoàn thiện trong tương lai.

6.1. Tăng cường Tính Sẵn sàng và Khả năng chịu lỗi (High Availability & Fault Tolerance)

Kiến trúc hiện tại vẫn tồn tại các điểm lỗi đơn (Single Point of Failure - SPOF). Nếu máy chủ EC2 chứa TA hoặc Droplet chứa Backend Service gặp sự cố, toàn bộ hệ thống sẽ ngừng hoạt động.

- **Triển khai TA với Auto Scaling Group và Load Balancer:**
  - Vấn đề: Dịch vụ TA là trái tim của hệ thống mật mã. Sự cố tại đây sẽ khiến mọi hoạt động sinh khóa và giải mã bị đình trệ.
  - Giải pháp: Di chuyển máy chủ TA vào một AWS Auto Scaling Group được cấu hình để duy trì ít nhất hai máy chủ chạy song song trên các Availability Zone (AZ) khác nhau. Đặt một Application Load Balancer (ALB) phía trước nhóm này.
  - Lợi ích:
    - Tự động phục hồi: Nếu một máy chủ TA gặp lỗi, Auto Scaling Group sẽ tự động chấm dứt nó và khởi tạo một máy chủ mới để thay thế.
    - Phân tải: ALB sẽ phân phối các yêu cầu từ Backend Service đến các máy chủ TA, tránh quá tải cho một máy chủ duy nhất.
    - Không gián đoạn: Backend Service sẽ luôn giao tiếp với địa chỉ cố định của ALB, không cần quan tâm đến địa chỉ IP của từng máy chủ TA cụ thể.
- **Phân tải cho Backend Service:**
  - Vấn đề: Một Droplet duy nhất sẽ trở thành nút thắt cổ chai khi lưu lượng truy cập tăng cao.
  - Giải pháp: Sử dụng dịch vụ DigitalOcean Load Balancer. Triển khai nhiều Droplet giống hệt nhau (chứa Nginx và Backend Service) và đặt chúng sau Load Balancer.

- Lợi ích: Tăng khả năng xử lý đồng thời nhiều yêu cầu, cải thiện tốc độ phản hồi cho người dùng và đảm bảo dịch vụ không bị gián đoạn nếu một trong các Droplet gặp sự cố.

6.2. Xây dựng Hệ thống Giám sát, Ghi log và Cảnh báo Tập trung

Việc kiểm tra log thủ công trên từng máy chủ là không khả thi trong môi trường sản xuất. Một hệ thống giám sát tập trung là cực kỳ cần thiết để chủ động phát hiện sự cố và phân tích hành vi hệ thống.

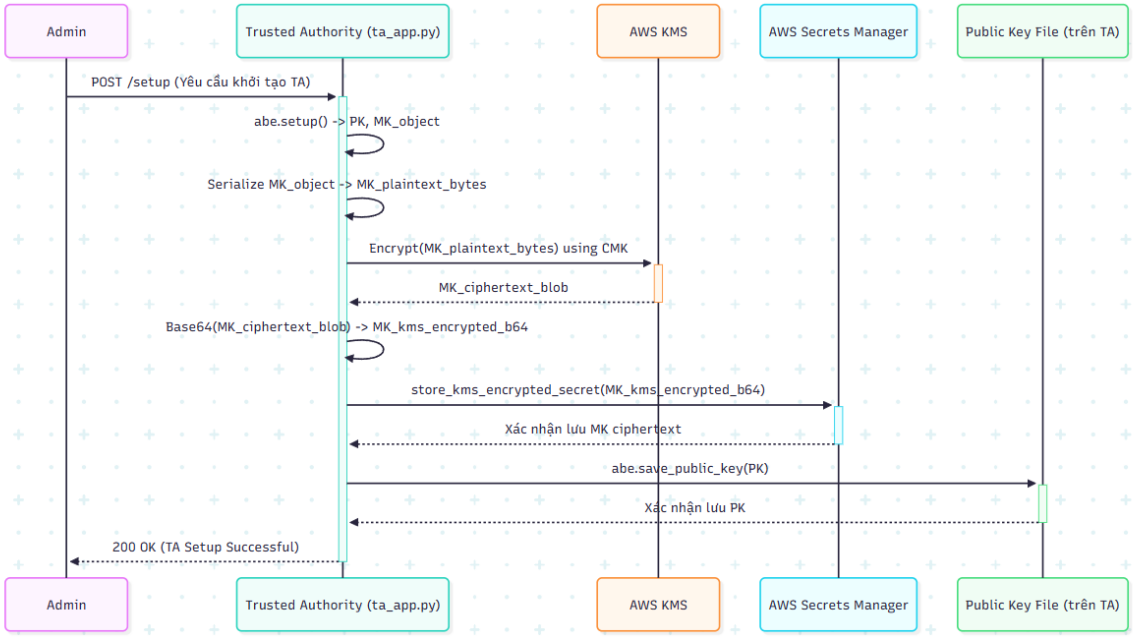
- **Centralized Logging (Ghi log Tập trung):**
  - Giải pháp: Triển khai một stack ghi log tập trung như EFK (Elasticsearch, Fluentd, Kibana).
    - Fluentd: Được cài đặt trên tất cả các máy chủ (cả EC2 của TA và Droplet của Backend) để thu thập log từ ứng dụng (Gunicorn), web server (Nginx) và hệ điều hành.
    - Elasticsearch: Lưu trữ và lập chỉ mục tất cả log được gửi về từ Fluentd.
    - Kibana: Cung cấp giao diện web mạnh mẽ để tìm kiếm, phân tích và trực quan hóa log, giúp truy vết lỗi và phân tích hành vi người dùng một cách nhanh chóng.
- **Metrics Monitoring and Alerting (Giám sát Chỉ số và Cảnh báo):**
  - Giải pháp: Sử dụng bộ đôi Prometheus và Grafana.
    - Prometheus: Thu thập các chỉ số vận hành theo thời gian thực như: tải CPU, sử dụng bộ nhớ, độ trễ của các yêu cầu API, số lượng lỗi HTTP 5xx, v.v.
    - Grafana: Tạo các dashboard trực quan để theo dõi sức khỏe của toàn bộ hệ thống.
    - Alertmanager: Tích hợp với Prometheus để gửi cảnh báo tự động (qua Email, Slack, Telegram) đến đội ngũ vận hành khi có chỉ số bất thường (ví dụ: "Độ trễ giải mã trung bình vượt quá 500ms" hoặc "CPU máy chủ TA > 90% trong 5 phút").

6.3. Tự động hóa Triển khai với CI/CD (Continuous Integration/Continuous Deployment)

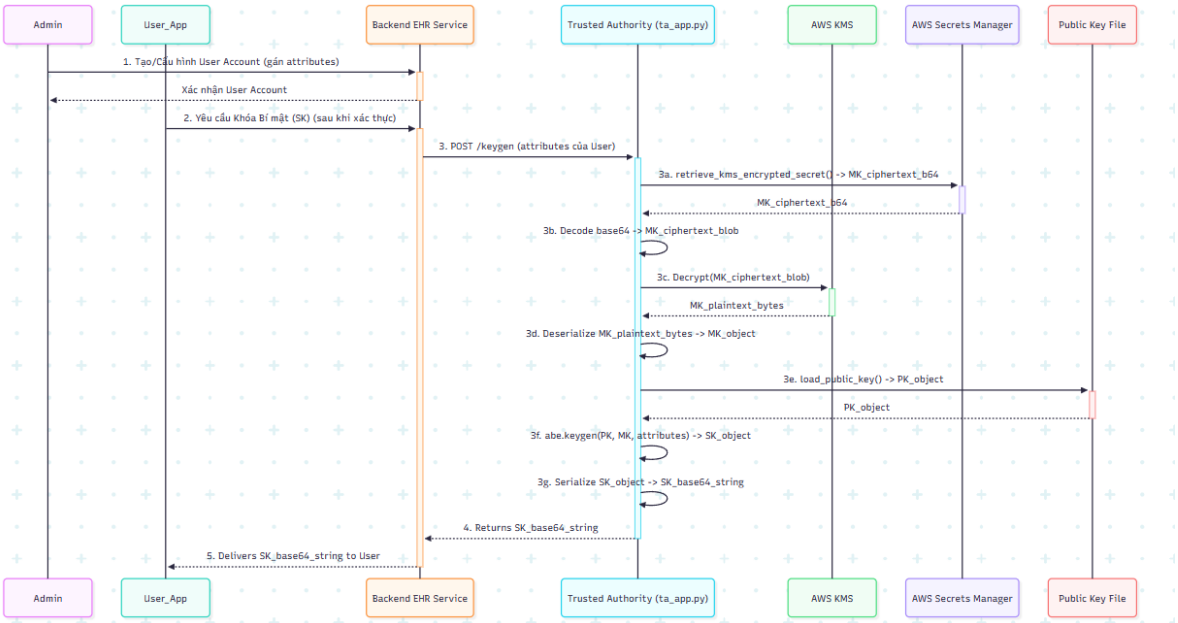
Quy trình triển khai thủ công hiện tại tốn thời gian và dễ xảy ra lỗi. Việc áp dụng CI/CD sẽ giúp chuẩn hóa và tăng tốc độ phát triển.

- Giải pháp: Xây dựng một đường ống (pipeline) CI/CD sử dụng các công cụ như GitHub Actions hoặc GitLab CI.
    - Continuous Integration (CI): Mỗi khi có một thay đổi được đẩy lên repository (ví dụ: git push), pipeline sẽ tự động:
        1. Chạy các bài kiểm thử đơn vị (unit tests) để đảm bảo logic không bị phá vỡ.
        2. Phân tích mã nguồn tĩnh để kiểm tra chất lượng và các lỗ hổng bảo mật cơ bản.
        3. Nếu thành công, xây dựng các ảnh Docker mới cho TA và Backend Service.
    - Continuous Deployment (CD): Sau khi CI thành công, pipeline sẽ tự động triển khai các ảnh Docker mới lên một môi trường kiểm thử (staging). Sau khi được phê duyệt, việc triển khai lên môi trường sản xuất (production) có thể được thực hiện chỉ bằng một cú nhấp chuột, giúp giảm thiểu rủi ro và thời gian chết.

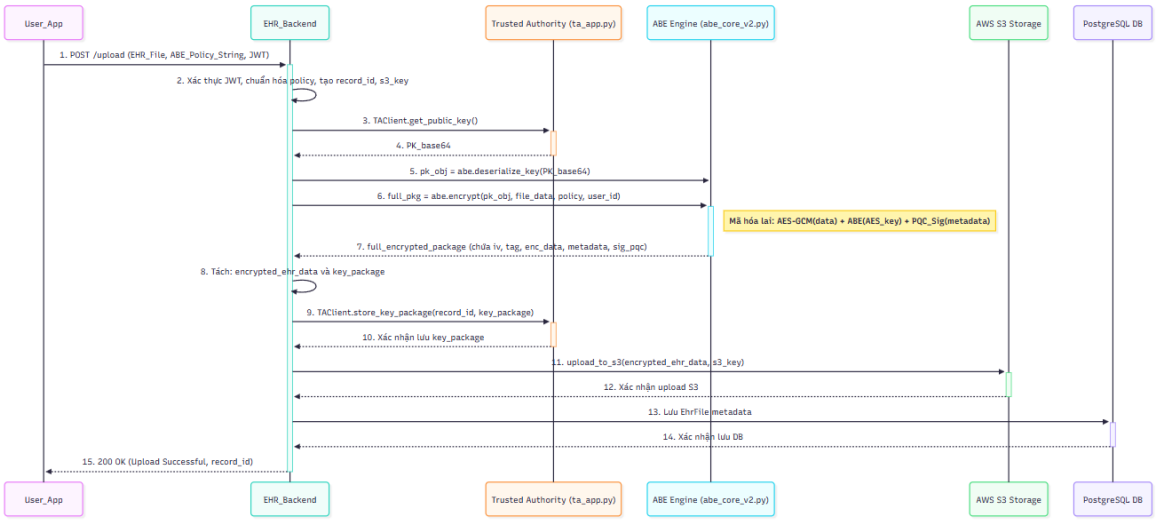
7. Kiến trúc hệ thống.



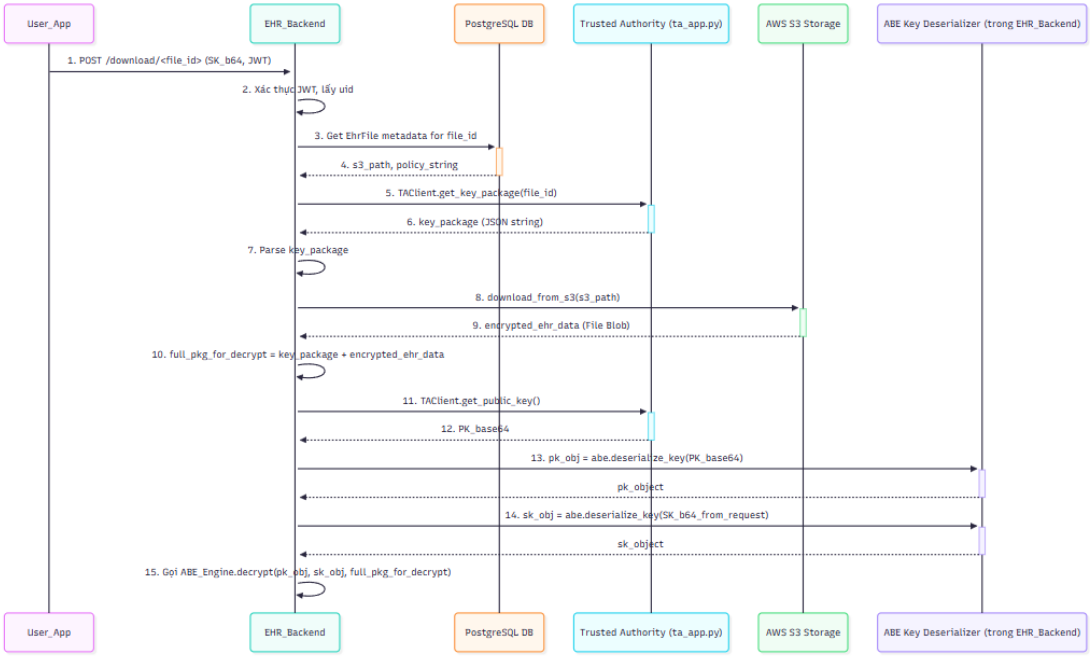
Hình 7 – Khởi tạo hệ thống (TA Setup)



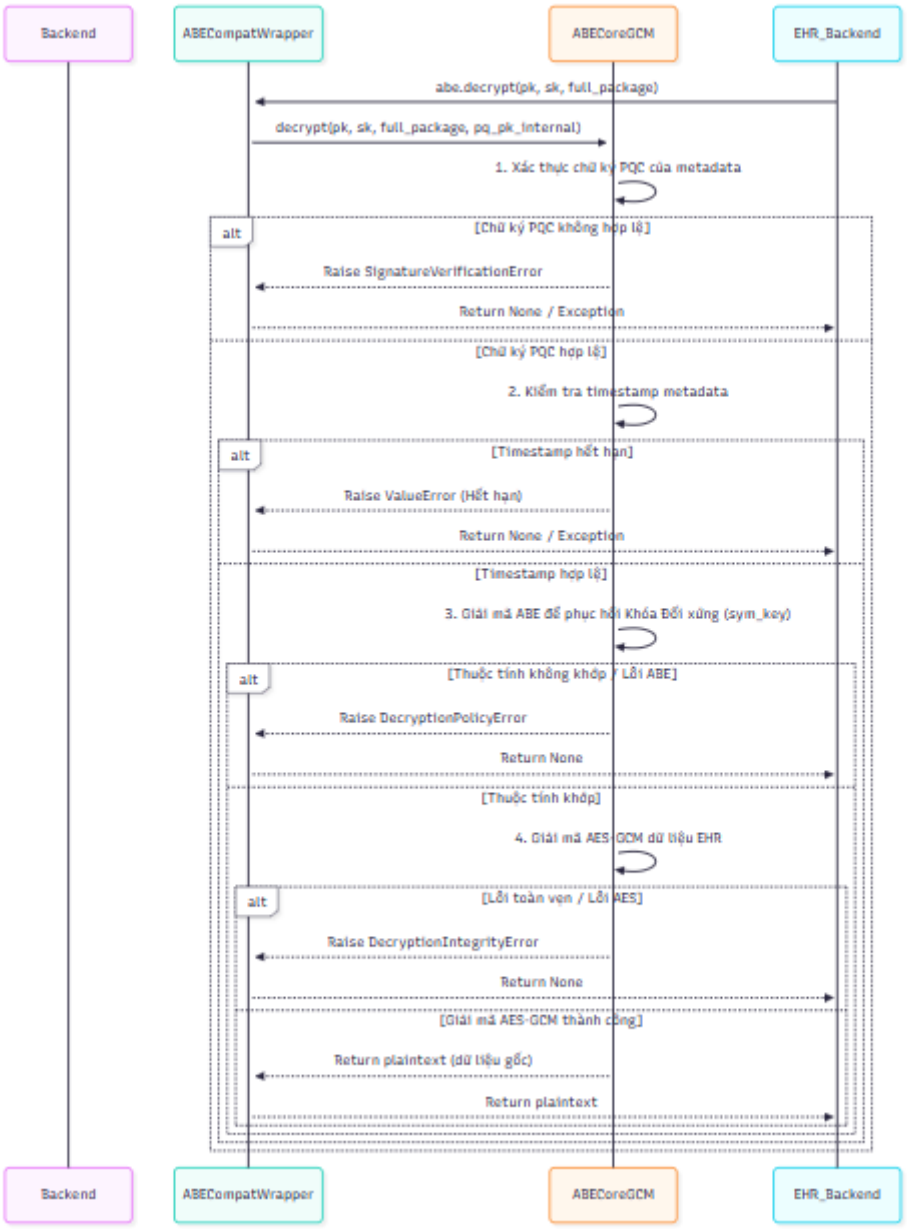
Hình 8 - Đăng ký & cấp phát khóa bí mật



Hình 9 - Tải lên & Mã hóa EHR



Hình 10 - Tải xuống & Chuẩn bị giải mã EHR



Hình 11 - Giải mã

7.1. Tổng quan kiến trúc.

Hệ thống Backend được chia làm 2 thành phần chính, bao gồm:

– API Gateway (Flask Application - đóng vai trò là điểm tiếp nhận trung tâm tất cả các yêu cầu từ phía client (frontend). Nhiệm vụ chính của API Gateway bao gồm:

- **Xác thực người dùng:** Kiểm tra và đảm bảo các yêu cầu từ client đều được xác thực hợp lệ, ví dụ kiểm tra token hoặc phiên đăng nhập (session).
- **Quản lý quy trình nghiệp vụ người dùng:**
  - Xử lý đăng nhập, đổi mật khẩu cho người dùng.
  - Cấp phát Secret Key (SK) dùng trong mã hóa dữ liệu hoặc truy cập thông tin.

    - Quản lý các thao tác upload và download hồ sơ y tế điện tử (EHR) từ người dùng.
- **Điều phối gọi tới các dịch vụ chuyên biệt.** API Gateway sẽ gọi tới các microservice chuyên trách như:
    - TA Service để xử lý các tác vụ liên quan đến mã hóa CP-ABE và quản lý khóa.
    - S3 Handler để xử lý việc lưu trữ và truy xuất hồ sơ EHR trên hệ thống lưu trữ đám mây (ví dụ Amazon S3).

– TA Service (Flask Microservice) - Đây là microservice riêng biệt, chuyên trách toàn bộ logic CP-ABE.

- **Chức năng chính:**
    - Setup: Khởi tạo hệ thống mã hóa, tạo các tham số và khóa hệ thống cần thiết cho CP-ABE.
    - Key Generation (keygen): Tạo các khóa cá nhân cho người dùng dựa trên các thuộc tính (ví dụ: vai trò, chức danh, quyền truy cập).
    - Store Key: Lưu trữ khóa cá nhân một cách an toàn.
    - Retrieve Key: Cung cấp khóa cá nhân khi có yêu cầu hợp lệ từ người dùng hoặc hệ thống.
- **Quản lý khóa bảo mật.**
    - MSK (Master Secret Key): Đây là khóa bí mật gốc và quan trọng nhất trong hệ thống CP-ABE, dùng để tạo khóa cá nhân và đảm bảo tính bảo mật toàn hệ thống.
    - MSK sẽ được lưu trữ ở nơi cực kỳ an toàn, thường là trong các thiết bị phần cứng chuyên dụng gọi là HSM (Hardware Security Module) hoặc các dịch vụ quản lý khóa đám mây như: AWS KMS (Key Management Service), Google Cloud KMS, Azure Key Vault.
    - Việc sử dụng HSM hoặc KMS giúp bảo vệ MSK khỏi các truy cập trái phép và đảm bảo tuân thủ các chuẩn bảo mật cao.
- **Public Key (PK):**
    - Đây là khóa công khai được sử dụng để mã hóa dữ liệu theo chính sách CP-ABE.

- PK được công bố rộng rãi, cho phép cả API Gateway và client tải về sử dụng để mã hóa thông tin, giúp dữ liệu chỉ được giải mã bởi người có khóa cá nhân phù hợp với chính sách đã định.

7.2. Mô hình dữ liệu.

Bảng 1 - User

Tên cột	Kiểu dữ liệu	Chú thích
id	SERIAL PK	Khóa chính
username	VARCHAR(50)	Đăng nhập, duy nhất
email	VARCHAR(100)	Duy nhất
password_hash	TEXT	Bcrypt hash
role	VARCHAR(50)	Doctor, Nurse, Admin, Emergency, Patient
department	VARCHAR(50)	Dept hoặc patient_id
created_at	TIMESTAMP	Tự động CURRENT_TIMESTAMP
sk_downloaded	BOOLEAN	Đánh dấu đã tải SK một lần

Bảng 2 - EHR_File

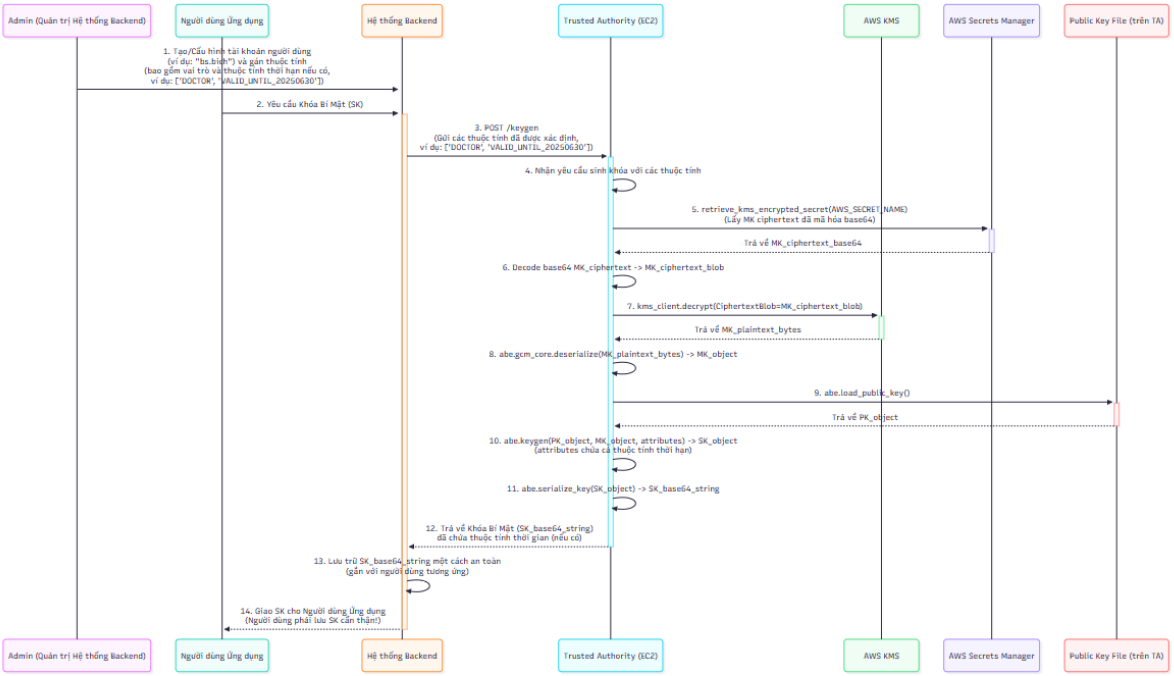
Tên cột	Kiểu dữ liệu	Chú thích
id	UUID/SERIAL	Mã bản ghi
filename	VARCHAR	Tên gốc
s3_key	VARCHAR	Đường dẫn object trong S3

owner_id	INT FK	Tham chiếu users.id
policy	JSONB	Chính sách CP-ABE
uploaded_at	TIMESTAMP	CURRENT_TIMESTAMP

7.3. Quy trình mã hóa dữ liệu EHR sử dụng Hybrid CP-ABE + AES.

TA Service (Trusted Authority Service) đóng vai trò trung tâm trong việc triển khai và vận hành. Nó được xây dựng dưới dạng Flask microservice, cung cấp các chức năng cốt lõi như sau:

1. *Khởi tạo hệ thống (setup).*



Hình 12 - Khởi tạo hệ thống

– Mục đích:

- Tạo khóa (Secret Key - SK) duy nhất cho mỗi người dùng, dựa trên tập hợp các thuộc tính được quản lý bởi Hệ thống Backend. Các thuộc tính này có thể bao gồm vai trò, quyền hạn và các thuộc tính chỉ định thời hạn hiệu lực của khóa (ví dụ: VALID_UNTIL_YYYYMMDD). Khóa bí mật này cho phép người dùng giải mã các bản mã (ciphertext) nếu tập thuộc tính của họ (bao gồm cả việc còn trong thời hạn hiệu lực) thỏa mãn chính sách truy cập được định nghĩa khi dữ liệu được mã hóa.

- Luồng hoạt động chi tiết:
    - Quản lý thuộc tính người dùng (Bởi Admin trên Hệ thống Backend - Bước chuẩn bị):
    - Quản trị viên của hệ thống Backend thực hiện việc tạo tài khoản cho người dùng (ví dụ: "bs.bich").
    - Trong quá trình này, Quản trị viên gán các thuộc tính cần thiết cho người dùng, bao gồm các thuộc tính vai trò (ví dụ: DOCTOR, CARDIOLOGY) và các thuộc tính quy định thời hạn hiệu lực cho khóa bí mật sẽ được tạo (ví dụ: VALID_UNTIL_20250630). Các thuộc tính này được lưu trữ và quản lý bởi Hệ thống Backend.
    - Yêu cầu tạo khóa từ người dùng ứng dụng: Người dùng cuối, thông qua ứng dụng, thực hiện một hành động yêu cầu cấp khóa (ví dụ: sau khi đăng nhập lần đầu hoặc khi cần cấp lại khóa).
- Backend Yêu cầu TA tạo khóa:
    - Hệ thống Backend, sau khi xác thực người dùng, thu thập các thuộc tính đã được định nghĩa cho người dùng đó (bao gồm cả thuộc tính thời hạn).
    - Backend gửi một yêu cầu POST đến API endpoint /keygen của Trusted Authority (TA). Yêu cầu này chứa danh sách đầy đủ các thuộc tính.
    - TA Nhận Yêu cầu: TA nhận yêu cầu sinh khóa từ Backend, cùng với danh sách thuộc tính.
- Truy xuất và Giải mã Master Key (MK):
    - TA truy xuất chuỗi ciphertext đã mã hóa base64 của Master Key từ AWS Secrets Manager.
    - TA giải mã base64 chuỗi này, sau đó gửi ciphertext blob của MK đến AWS KMS để giải mã bằng Customer Managed Key (CMK) đã được cấu hình.
    - AWS KMS trả về phiên bản plaintext (dạng bytes) của Master Key. MK gốc chỉ tồn tại trong bộ nhớ của TA trong quá trình xử lý yêu cầu.
    - TA khôi phục đối tượng Master Key từ dạng bytes.
    - Tải khóa công khai (PK): TA tải khóa công khai (PK) của hệ thống ABE từ nơi lưu trữ cục bộ.
- Tạo khóa (SK) với Thuộc tính Thời hạn:

- TA sử dụng PK, MK (đã giải mã) và toàn bộ danh sách thuộc tính (bao gồm VALID_UNTIL_YYYYMMDD nếu có) để gọi hàm abe.keygen().
- Một khóa (SK) được tạo ra, trong đó các thuộc tính (kể cả thuộc tính thời hạn) được gắn trực tiếp vào cấu trúc của khóa.
- Serialize và Trả về SK: SK được serialize và mã hóa base64, sau đó TA trả về chuỗi SK này cho Hệ thống Backend.

– Backend Xử lý và Giao SK:
- Hệ thống Backend nhận SK từ TA và có thể thực hiện các bước lưu trữ an toàn (ví dụ: trong database của Backend, gắn với thông tin người dùng).
- Backend giao SK cho Người dùng Ứng dụng. Người dùng có trách nhiệm lưu trữ khóa này một cách cẩn thận.

– Nguyên tắc Giải mã và Kiểm soát Thời hạn:
- Khi người dùng cố gắng giải mã một bản mã, hệ thống ABE sẽ kiểm tra xem các thuộc tính trong SK của người dùng (bao gồm cả thuộc tính VALID_UNTIL_YYYYMMDD) có thỏa mãn chính sách truy cập của bản mã hay không.
- Ví dụ: Một bản mã có chính sách (DOCTOR AND VALID_UNTIL_20250630). Người dùng có SK chứa thuộc tính ['DOCTOR', 'VALID_UNTIL_20250630'] sẽ giải mã được nếu ngày hiện tại chưa vượt quá 2025-06-30. Nếu thuộc tính thời hạn trong SK là VALID_UNTIL_20250531, người dùng sẽ không giải mã được bản mã đó sau ngày 31/05/2025, ngay cả khi họ có thuộc tính DOCTOR.

– Sử dụng lại các thành phần hệ thống:
- Quy trình này dựa trên khóa công khai (PK) và Khóa Chủ (MK) đã được thiết lập an toàn trong bước /setup, với MK được bảo vệ bởi AWS KMS.

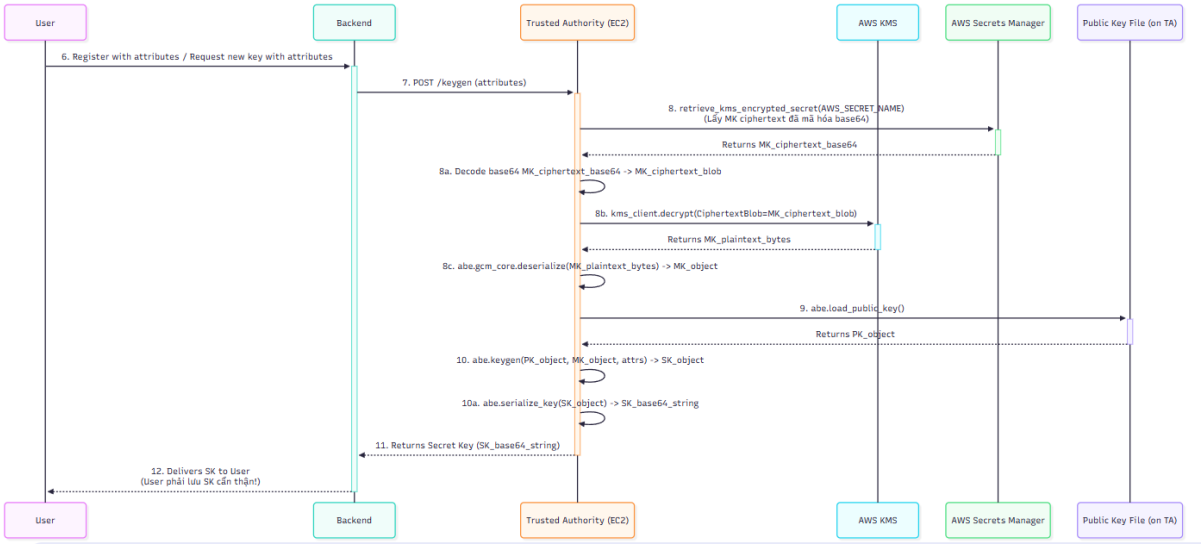
Bảng 3 - Hàm setup()

```
#25/6 setup & keygen
@app.route('/setup', methods=['POST'])
def setup():
    """
```

```
    Thiết lập hệ thống: tạo PK và MK.
    Mã hóa MK bằng KMS, sau đó lưu ciphertext (base64) vào Secrets Manager.
    Lưu PK ra file cục bộ.
    """
    if not AWS_KMS_KEY_ID: # Kiểm tra lại phòng trường hợp app vẫn chạy dù thiếu config
        app.logger.error("TA setup failed: AWS_KMS_KEY_ID not configured.")
        return jsonify({"error": "TA setup failed: KMS Key not configured"}), 500
    try:
        pk, mk_object = abe.setup()

        # Lưu PK vào file như cũ
        abe.save_public_key(pk, filename_prefix="keys", directory=".")

        # 1. Serialize Master Key (MK) thành bytes thô
        #    Sử dụng gcm_core.serialize vì ABECompatWrapper.serialize_key đã bao gồm base64
        mk_plaintext_bytes = abe.gcm_core.serialize(mk_object)

        # 2. Mã hóa MK bytes bằng AWS KMS
        kms_client = boto3.client('kms', region_name=AWS_REGION)
        encrypt_response = kms_client.encrypt(
            KeyId=AWS_KMS_KEY_ID,
            Plaintext=mk_plaintext_bytes
        )
        mk_ciphertext_blob = encrypt_response['CiphertextBlob'] # Đây là bytes
```

```
        # 3. Chuyển MK ciphertext (đã mã hóa bằng KMS) thành chuỗi base64
                                                    mk_kms_encrypted_b64_str =
base64.b64encode(mk_ciphertext_blob).decode('utf-8')

        # 4. Lưu chuỗi base64 của MK ciphertext vào AWS Secrets Manager
                                        store_kms_encrypted_secret(AWS_SECRET_NAME,
mk_kms_encrypted_b64_str)

       app.logger.info("TA setup completed successfully. MK encrypted with KMS and
stored in Secrets Manager.")
        return jsonify({"message": "TA setup completed successfully."}), 200
    except Exception as e:
        app.logger.error(f"TA setup failed: {e}")
        traceback.print_exc()
        return jsonify({"error": "TA setup failed", "details": str(e)}), 500
```

2. Sinh khóa người dùng (keygen).



Hình 13 - Quá trình sinh khóa

– Mục đích:

- Tạo khóa (Secret Key - SK) duy nhất cho mỗi người dùng dựa trên tập hợp các thuộc tính (attributes) mà họ sở hữu. khóa này cho phép người dùng giải mã các bản mã (ciphertext) nếu tập thuộc tính của họ thỏa mãn chính sách truy cập (access policy) được định nghĩa khi dữ liệu được mã hóa.

– Luồng hoạt động chi tiết:

- Yêu cầu từ Người dùng/Backend: Người dùng (thông qua một ứng dụng Backend) gửi yêu cầu đến API endpoint /keygen của Trusted Authority (TA). Yêu cầu này bao gồm một danh sách các thuộc tính cụ thể mà người dùng đó sở hữu (ví dụ: ['DOCTOR', 'CARDIOLOGY', 'HOSPITAL_A']).

– Truy xuất Master Key Ciphertext:

- TA nhận yêu cầu và tiến hành truy xuất phiên bản đã được mã hóa của Khóa Chủ (Master Key - MK).
- TA gọi hàm retrieve_kms_encrypted_secret() để lấy chuỗi ciphertext (đã được mã hóa base64) của MK từ dịch vụ AWS Secrets Manager. Chuỗi này là kết quả của việc MK gốc được mã hóa bằng một Customer Managed Key (CMK) trên AWS KMS.

– Giải mã Master Key bằng AWS KMS:

- TA giải mã chuỗi base64 nhận được từ Secrets Manager để thu được CiphertextBlob của MK.

- TA gửi CiphertextBlob này đến dịch vụ AWS KMS, yêu cầu giải mã bằng CMK đã được chỉ định.
- AWS KMS thực hiện giải mã và trả về Plaintext của MK (dưới dạng bytes) cho TA. Master Key gốc chỉ tồn tại trong bộ nhớ của TA trong thời gian xử lý yêu cầu này.
- Khôi phục đối tượng Master Key: TA chuyển đổi Plaintext (bytes) của MK trở lại thành đối tượng Master Key (mk_object) mà thư viện mã hóa ABE (Charm-Crypto) có thể sử dụng.

– Tải khóa công khai (PK): TA tải khóa công khai (pk_object) của hệ thống ABE từ nơi lưu trữ đã được thiết lập trước đó (ví dụ: từ một file cục bộ trên server TA như keys_public.key).

– Tạo khóa (SK):

- Sử dụng pk_object, mk_object (Master Key đã được giải mã an toàn) và danh sách thuộc tính (attrs) do người dùng cung cấp, TA gọi hàm cốt lõi của thư viện ABE:
- sk_object = abe.keygen(pk_object, mk_object, attrs)
- Kết quả là một khóa (sk_object) được tạo ra, gắn liền với các thuộc tính của người dùng.

– Serialize và Trả về SK:

- sk_object được serialize (chuyển đổi thành dạng bytes) và sau đó mã hóa base64 để dễ dàng truyền tải.
- TA trả về chuỗi base64 của khóa này cho Backend.
- Backend Giao SK cho Người dùng: Backend nhận SK từ TA và chuyển giao cho người dùng cuối. Người dùng có trách nhiệm lưu trữ khóa của mình một cách an toàn.

– Nguyên tắc Giải mã:

- khóa (SK) này cho phép người dùng giải mã các bản mã nếu và chỉ nếu tập thuộc tính được liên kết với SK của họ thỏa mãn logic của chính sách truy cập được đính kèm với dữ liệu đã mã hóa.
- Ví dụ: Một người dùng sở hữu các thuộc tính ['DOCTOR', 'CARDIOLOGY'] sẽ có khả năng giải mã các bản mã được mã hóa với chính sách truy cập như (DOCTOR AND CARDIOLOGY) hoặc (DOCTOR OR ONCOLOGY) (nếu

thuộc tính DOCTOR đủ để thỏa mãn một phần của chính sách). Biểu thức logic cụ thể phụ thuộc vào cách chính sách được định nghĩa khi mã hóa.

– Sử dụng lại các thành phần hệ thống:
  - Quá trình này sử dụng lại khóa công khai (PK) và dựa vào Khóa Chủ (MK) đã được thiết lập an toàn trong bước /setup (với MK được bảo vệ bởi AWS KMS và lưu trữ trong AWS Secrets Manager).

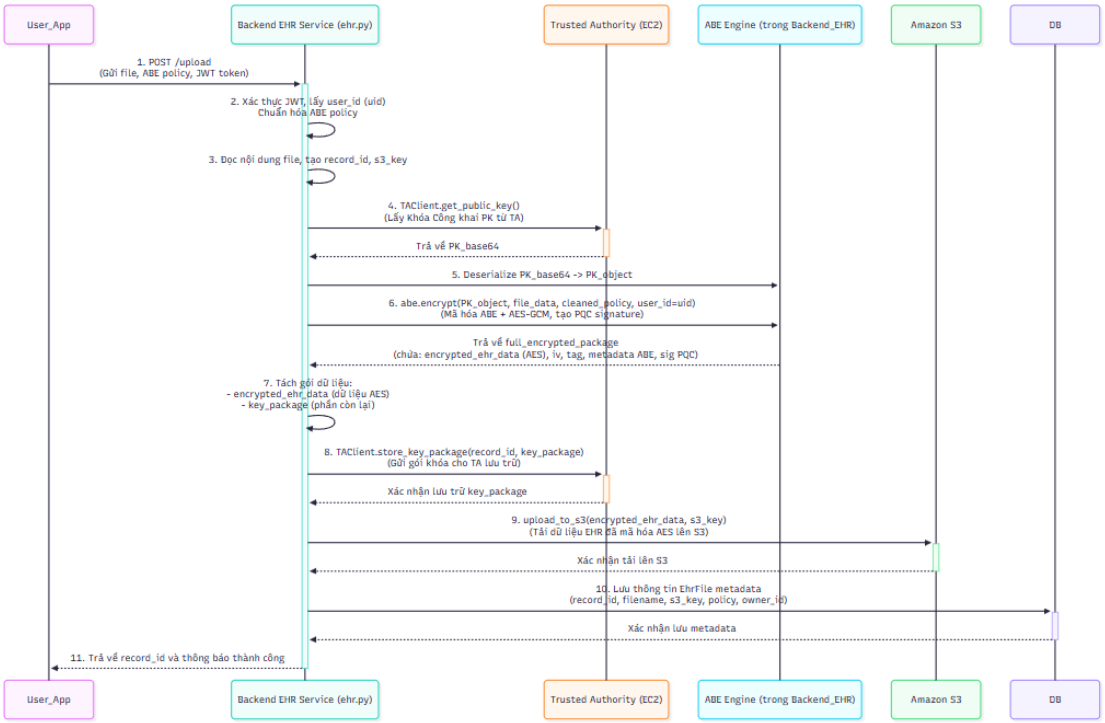
Bảng 4 - Hàm keygen()

```
@app.route('/keygen', methods=['POST'])
def keygen():
    """
    Tạo khóa bí mật (SK) cho người dùng.
    Lấy MK ciphertext từ Secrets Manager, giải mã bằng KMS, rồi sử dụng.
    """
    if not AWS_KMS_KEY_ID: # Kiểm tra cấu hình
        app.logger.error("Key generation failed: AWS_KMS_KEY_ID not configured.")
        return jsonify({"error": "Key generation failed: KMS Key not configured"}), 500
    try:
        data = request.get_json()
        if not data or 'attributes' not in data:
            return jsonify({"error": "Missing 'attributes' in request body"}), 400

        attrs = data.get('attributes', [])

        # 1. Lấy chuỗi base64 của MK ciphertext từ AWS Secrets Manager
                                                    mk_kms_encrypted_b64_str =
retrieve_kms_encrypted_secret(AWS_SECRET_NAME)
```

```
    if not mk_kms_encrypted_b64_str:
        app.logger.error(f"Master key not found or empty in Secrets Manager: {AWS_SECRET_NAME}")
        return jsonify({"error": "Key generation failed: Master key not configured or accessible"}), 500

    # 2. Decode base64 để lấy MK ciphertext blob (dạng bytes)
    mk_ciphertext_blob = base64.b64decode(mk_kms_encrypted_b64_str)

    # 3. Giải mã MK ciphertext blob bằng AWS KMS
    kms_client = boto3.client('kms', region_name=AWS_REGION)
    decrypt_response = kms_client.decrypt(
        CiphertextBlob=mk_ciphertext_blob
        # EncryptionContext={'Purpose': 'ABE Master Key Protection'}
    )
    mk_plaintext_bytes = decrypt_response['Plaintext'] # Đây là bytes thô của MK

    # 4. Deserialize MK plaintext bytes thành đối tượng MK của Charm
    #    Sử dụng gcm_core.deserialize vì nó nhận bytes thô
    mk_object = abe.gcm_core.deserialize(mk_plaintext_bytes)

    # Load PK từ file (như cũ)
    pk = abe.load_public_key(filename='keys_public.key', directory='.')

    # Sinh khóa SK cho người dùng (như cũ, nhưng với mk_object đã giải mã an toàn)
```

```
        sk = abe.keygen(pk, mk_object, attrs)


        # Serialize và trả về SK dạng base64 string (như cũ)
        sk_b64_bytes = abe.serialize_key(sk)
        app.logger.info(f"Successfully generated SK for attributes: {attrs}")
        return jsonify({
            "sk": sk_b64_bytes.decode('utf-8')
        }), 200
    except Exception as e:
        app.logger.error(f"Key generation failed: {e}")
        traceback.print_exc()
        return jsonify({"error": "Key generation failed", "details": str(e)}), 500
```

3. Luồng tải dữ liệu lên (Hàm upload).



Hình 14 - Luồng tải dữ liệu lên

– Mục đích:

- Cho phép người dùng đã được xác thực có thể tải lên một file hồ sơ sức khỏe điện tử (EHR). File này sẽ được mã hóa bằng một lược đồ lai (ABE + AES-GCM), với các thành phần khóa được quản lý bởi Trusted Authority (TA) và dữ liệu đã mã hóa được lưu trữ trên Amazon S3. Một chính sách truy cập dựa trên thuộc tính (ABE policy) sẽ được người dùng định nghĩa và gắn liền với dữ liệu.

– Luồng hoạt động:
  - Yêu cầu tải lên: Người dùng, thông qua ứng dụng, gửi yêu cầu POST đến endpoint /api/ehr/upload của Backend EHR Service. Yêu cầu này chứa file cần tải lên, một chuỗi định nghĩa chính sách truy cập ABE (ABE policy), và được xác thực bằng JWT.
– Chuẩn bị dữ liệu:
  - Backend EHR Service xác thực JWT và lấy user_id.
  - Chính sách ABE được chuẩn hóa (ví dụ: loại bỏ khoảng trắng thừa, chuyển về chữ thường).
  - Nội dung file được đọc vào bộ nhớ. Một record_id duy nhất (UUID) và một s3_key (đường dẫn lưu trữ trên S3) được tạo ra.
  - Lấy khóa công khai (PK): Backend gọi đến Trusted Authority (TA) để lấy khóa công khai (PK) của hệ thống ABE. PK này được deserialize để sử dụng.
– Thực hiện Mã hóa Lai:
  - Backend sử dụng ABE Engine (thư viện abe_core_v2) để gọi hàm abe.encrypt(PK, file_data, cleaned_policy, user_id).
– Quá trình này bao gồm:
  - Tạo một khóa đối xứng (symmetric key) ngẫu nhiên.
  - Mã hóa khóa đối xứng này bằng ABE theo cleaned_policy (sử dụng PK).
  - Dẫn xuất một khóa AES từ khóa đối xứng.
  - Mã hóa nội dung file gốc (file_data) bằng AES-GCM sử dụng khóa AES vừa dẫn xuất, tạo ra encrypted_ehr_data, IV (Initialization Vector), và tag xác thực.

- Tạo metadata chứa thông tin ngữ cảnh (policy, user_id, timestamp, khóa đối xứng đã mã hóa ABE) và ký metadata này bằng chữ ký Hậu Lượng tử (PQC) để chống tấn công lạm dụng/phát lại.
- Kết quả là một "gói dữ liệu mã hóa hoàn chỉnh" (full_encrypted_package).

– Phân tách và lưu trữ thành phần:

- full_encrypted_package được tách thành hai phần:
  - encrypted_ehr_data: Phần dữ liệu đã được mã hóa bằng AES.
  - key_package: Phần còn lại, chứa IV, tag AES, metadata ABE (bao gồm khóa đối xứng đã mã hóa ABE), và chữ ký PQC.
- Gửi gói Khóa đến TA: key_package được gửi đến TA để lưu trữ an toàn, liên kết với record_id.
- Tải dữ liệu Mã hóa lên S3: encrypted_ehr_data được tải lên Amazon S3 dưới s3_key đã tạo.
- Lưu Metadata: Thông tin metadata của file (bao gồm record_id, tên file gốc, s3_key, chính sách ABE gốc, owner_id) được lưu vào cơ sở dữ liệu của Backend.
- Phản hồi cho người dùng: Backend trả về record_id và một thông báo xác nhận việc tải lên và lưu trữ thành công cho người dùng.

– Kết quả:

- Dữ liệu EHR gốc được bảo vệ bằng mã hóa mạnh. Khóa giải mã dữ liệu (khóa đối xứng) được bảo vệ bởi ABE và chỉ có thể được phục hồi bởi những người dùng sở hữu khóa (SK) có thuộc tính thỏa mãn chính sách ABE đã định nghĩa. Dữ liệu mã hóa và các thành phần khóa được lưu trữ tách biệt, tăng cường tính bảo mật.

Bảng 5 - Hàm upload

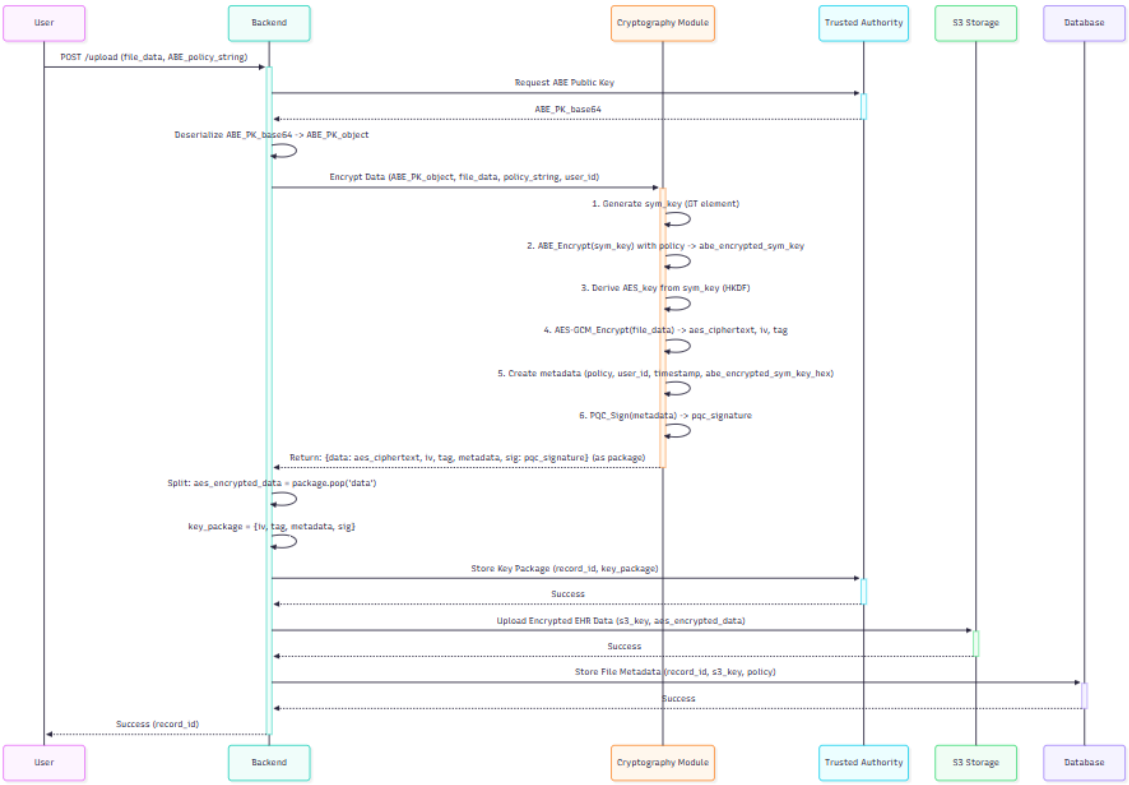
```
@ehr_bp.route('/upload', methods=['POST'])
@jwt_required()
def upload():
    uid = get_jwt_identity()
```

```
    file = request.files.get('file')
    policy_from_form = request.form.get('policy')

    if not file or not policy_from_form:
        return jsonify({"msg": "file và policy là bắt buộc"}), 400

    cleaned_policy = re.sub(r'\s+', ' ', policy_from_form).strip().lower()
     current_app.logger.info(f"User {uid} uploading with Cleaned Policy for ABE: '{cleaned_policy}'")

    data = file.read()
    record_id = str(uuid.uuid4())
    s3_key = f"{uid}/{record_id}.aes.enc" # Đuôi file chỉ chứa dữ liệu AES

    try:
        pk_base64 = TAClient.get_public_key()
        pk = abe.deserialize_key(pk_base64)

        # 1. Mã hóa để tạo ra gói dữ liệu hoàn chỉnh
        full_encrypted_package = abe.encrypt(pk, data, cleaned_policy, user_id=uid)

        if full_encrypted_package is None:
            raise ValueError("Quá trình mã hóa tổng thể thất bại.")

        # 2. Tách gói dữ liệu
```

```
        encrypted_ehr_data = full_encrypted_package.pop('data')
        # Phần còn lại là key_package
        key_package = full_encrypted_package

        # 3. Gửi key_package cho TA để lưu trữ
        TAClient.store_key_package(record_id, key_package)

        # 4. Tải dữ liệu EHR đã mã hóa AES lên S3
        upload_to_s3(encrypted_ehr_data, s3_key)

        # 5. Lưu thông tin vào database (thêm lại s3_key)
        ef = EhrFile(
            record_id=record_id,
            filename=file.filename,
            s3_key=s3_key, # Lưu lại key của S3
            policy=policy_from_form,
            owner_id=uid
        )
        db.session.add(ef)
        db.session.commit()

         return jsonify({"record_id": record_id, "message": "File uploaded and stored successfully"}), 201

    except requests.exceptions.RequestException as e:
```

```
        db.session.rollback()
        current_app.logger.error(f"TA connection failed during upload for user {uid}: {e}")
        return jsonify({"msg": "Không thể kết nối đến máy chủ lưu trữ khóa (TA)."}), 503
    except Exception as e:
        db.session.rollback()
        current_app.logger.error(f"Upload process failed for user {uid}: {e}")
        traceback.print_exc()
        return jsonify({"msg": "Upload process failed", "error": str(e)}), 500
```

4. Mã hóa dữ liệu EHR và bảo vệ khóa đối xứng AES.



Hình 15 - Mã hóa và upload dữ liệu

– Tạo khóa đối xứng tạm thời (sym_key):

- Một khóa đối xứng ngẫu nhiên, tạm thời (sym_key) được tạo ra. Khóa này thuộc về nhóm mật mã đường cong elliptic (cụ thể là một phần tử ngẫu nhiên của nhóm GT từ thư viện Charm-Crypto) và sẽ được sử dụng gián tiếp để mã hóa dữ liệu EHR.

– Mã hóa khóa đối xứng sym_key bằng CP-ABE:
  - Khóa đối xứng sym_key được mã hóa bằng thuật toán CP-ABE (Ciphertext-Policy Attribute-Based Encryption), sử dụng khóa công khai (PK) của hệ thống và chính sách truy cập (ABE policy) do người dùng định nghĩa khi yêu cầu tải lên.
  - Kết quả của bước này là abe_encrypted_key – đây là phiên bản đã được mã hóa của sym_key. Chỉ những người dùng sở hữu Khóa Bí mật (SK) có các thuộc tính thỏa mãn chính sách ABE mới có thể giải mã abe_encrypted_key để thu lại sym_key gốc.

– Dẫn xuất khóa AES và mã hóa dữ liệu EHR bằng AES-GCM:
  - Từ sym_key gốc (phiên bản plaintext, trước khi được mã hóa bằng ABE), một khóa AES-256 (aes_key) được dẫn xuất an toàn thông qua hàm _derive_key. Hàm này sử dụng thuật toán HKDF (HMAC-based Key Derivation Function) với hàm băm SHA-512, đảm bảo khóa AES có độ dài và tính ngẫu nhiên phù hợp cho việc mã hóa.
  - Một Initialization Vector (IV) ngẫu nhiên, có độ dài 12 bytes, được tạo ra, cần thiết cho chế độ hoạt động GCM của AES.
  - Dữ liệu EHR gốc (plaintext) được mã hóa bằng thuật toán AES ở chế độ GCM (Galois/Counter Mode) sử dụng aes_key và iv. Chế độ AES-GCM được chọn vì nó cung cấp đồng thời cả tính bảo mật (encryption) và tính toàn vẹn/xác thực (integrity/authenticity) cho dữ liệu được mã hóa.

– Kết quả của quá trình mã hóa AES-GCM là:
  - ciphertext: Dữ liệu EHR đã được mã hóa.
  - tag: Một thẻ xác thực (authentication tag) được tạo ra bởi AES-GCM, dùng để kiểm tra tính toàn vẹn của ciphertext trong quá trình giải mã.

Bảng 6 - Hàm encrypt()

```
def encrypt(self, pk, plaintext, policy, user_id, timestamp, pqc_signer_object):
    """Mã hóa dữ liệu bằng ABE + AES-GCM, ký metadata chống misuse/replay."""
    sym_key = self.group.random(GT)
    abe_encrypted_key = self.abe.encrypt(pk, sym_key, policy)

    if abe_encrypted_key is None:
        raise ValueError("ABE encryption failed. Check policy syntax.")

    # Mã hóa dữ liệu bằng AES-GCM
    iv = get_random_bytes(12)
    aes_key = self._derive_key(sym_key)
    cipher = AES.new(aes_key, AES.MODE_GCM, nonce=iv)
    ciphertext, tag = cipher.encrypt_and_digest(
        plaintext if isinstance(plaintext, bytes) else plaintext.encode('utf-8')
    )

    # Ký metadata bảo vệ context (chống misuse/replay)
    metadata = {
        'policy': policy,
        'user_id': user_id,
        'timestamp': timestamp,
        'abe_key_hex': objectToBytes(abe_encrypted_key, self.group).hex()
    }
    sig = pqc_signer_object.sign(json.dumps(metadata, sort_keys=True).encode())
```

```
    return {
        'iv': iv,
        'tag': tag,
        'data': ciphertext,
        'metadata': metadata,
        'sig': sig
    }
```

5. Tạo Metadata, ký PQC, và hoàn thiện gói dữ liệu mã hóa.

- Tạo Metadata ngữ cảnh:
- Một đối tượng metadata được xây dựng để chứa các thông tin quan trọng liên quan đến ngữ cảnh của việc mã hóa, bao gồm:
    - policy: Chuỗi chính sách ABE đã được áp dụng.
    - user_id: Định danh của người dùng đã thực hiện thao tác mã hóa.
    - timestamp: Dấu thời gian (ISO 8601 UTC) ghi lại thời điểm mã hóa.
    - abe_key_hex: Phiên bản abe_encrypted_key (khóa đối xứng đã mã hóa bằng ABE) được chuyển đổi sang dạng chuỗi hexadecimal để dễ dàng lưu trữ và truyền tải.
- Ký Metadata bằng chữ ký Hậu Lượng tử (PQC Signature):
    - Toàn bộ đối tượng metadata (sau khi được serialize thành một chuỗi JSON chuẩn hóa) được ký điện tử bằng một thuật toán chữ ký Hậu Lượng tử (PQC), chẳng hạn như Dilithium2, được cung cấp bởi thư viện OQS.
    - Chữ ký này (sig) được tạo ra bằng cách sử dụng khóa ký PQC (pqc_signer_object) được quản lý bởi lớp ABECompatWrapper.
- Việc ký metadata này nhằm mục đích:
    - Đảm bảo tính toàn vẹn của metadata (không bị thay đổi sau khi tạo).
    - Xác thực nguồn gốc của metadata.
    - Chống lại các tấn công lạm dụng hoặc phát lại (replay attacks) đối với các thành phần khóa.

- Cung cấp khả năng chống lại các mối đe dọa từ máy tính lượng tử đối với tính hợp lệ của chữ ký trong tương lai.

– Đóng gói dữ liệu hoàn chỉnh (full_encrypted_package):

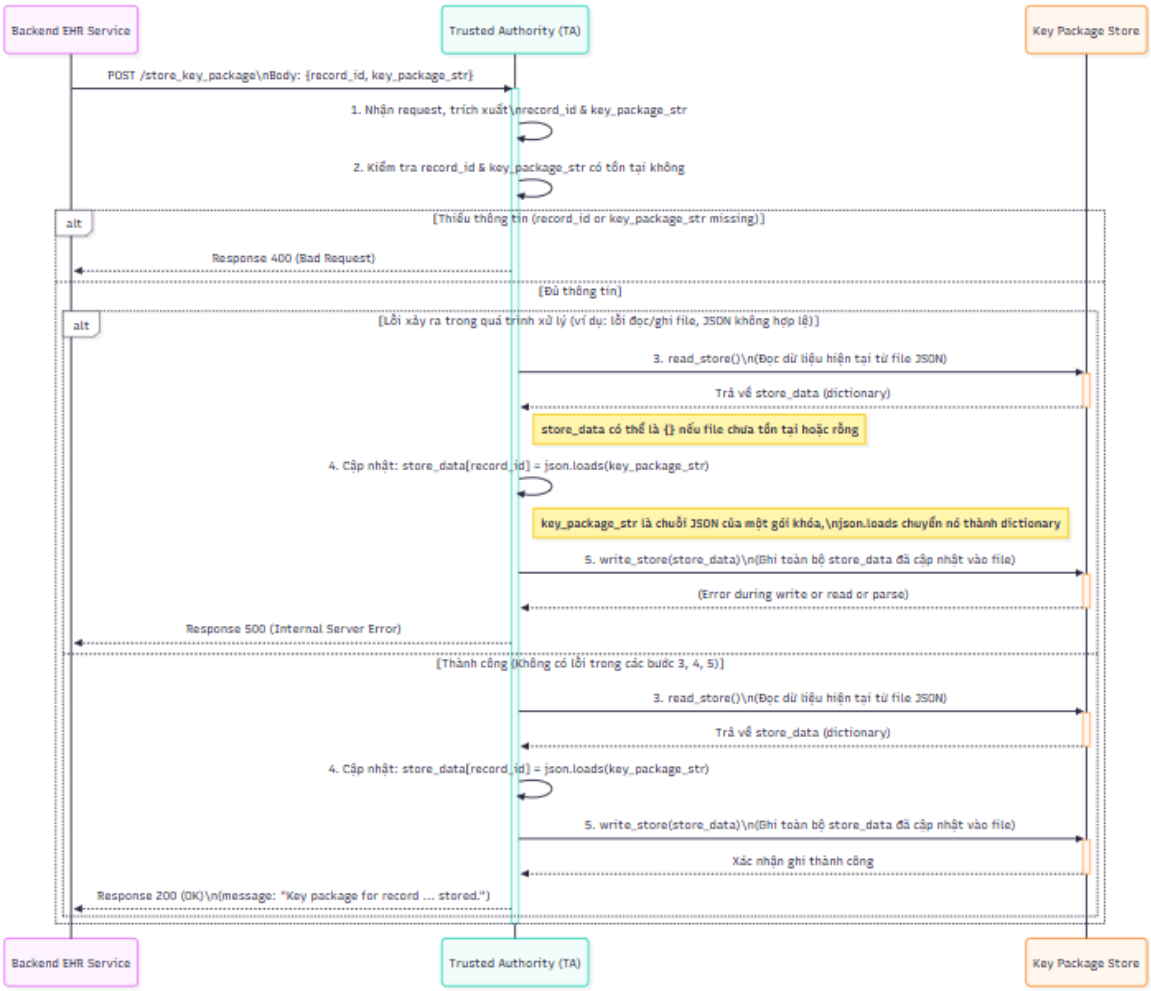
- Tất cả các thành phần mật mã đã tạo ra được tập hợp lại thành một cấu trúc dữ liệu (dictionary) duy nhất, gọi là "gói dữ liệu mã hóa hoàn chỉnh". Gói này bao gồm:
- iv: Initialization Vector của AES-GCM.
- tag: Thẻ xác thực của AES-GCM.
- data: Dữ liệu EHR đã được mã hóa bằng AES-GCM (ciphertext).
- metadata: Đối tượng metadata chứa thông tin ngữ cảnh và abe_key_hex.
- sig: Chữ ký PQC của metadata.
- Gói dữ liệu này sau đó được trả về từ hàm abe.encrypt để Backend EHR Service tiếp tục xử lý (tách data để lưu lên S3 và gửi phần còn lại – key_package – cho Trusted Authority).

Bảng 7 – Gọi encryp trong Wrapper

```
class ABECompatWrapper:
    def encrypt(self, pk, plaintext, policy, user_id):
        """
        Gói hàm encrypt của ABECoreGCM. Tự động tạo timestamp.
        """
        try:
            timestamp = datetime.now(timezone.utc).isoformat()
            logging.info(f"Đang mã hóa cho user '{user_id}' với chính sách: '{policy}'")
                return self.gcm_core.encrypt(pk, plaintext, policy, user_id, timestamp, self.pqc_signer)
        except Exception as e:
            logging.error(f"Mã hóa thất bại trong wrapper: {e}")
            traceback.print_exc()
```

```
        return None
```

6. Lưu trữ AES-GCM Key (store_key_package).



Hình 16 - Sơ đồ Luồng Lưu trữ Gói Khóa (Key Package) tại TA

– Mục đích:

- Cho phép hệ thống Backend EHR Service gửi và lưu trữ key_package lên Trusted Authority (TA). Gói khóa này chứa các thành phần mật mã cần thiết (IV, tag xác thực AES, metadata ABE đã được ký bằng PQC - bao gồm khóa đối xứng đã mã hóa ABE) để giải mã một file EHR cụ thể sau này.

– Luồng hoạt động:

- Yêu cầu từ Backend: Sau khi Backend EHR Service thực hiện mã hóa lai một file EHR, nó sẽ tách ra encrypted_ehr_data (để lưu lên S3) và

key_package. Backend gửi một yêu cầu POST đến endpoint /store_key_package của TA. Body của request chứa:

- record_id: Mã định danh duy nhất của hồ sơ EHR.
- key_package: Một chuỗi JSON biểu diễn "gói khóa".

– Xử lý tại TA:

- TA nhận request và kiểm tra tính hợp lệ của dữ liệu đầu vào (record_id và key_package không được rỗng).
- TA đọc nội dung hiện tại của file lưu trữ gói khóa (ví dụ: secrets/key_package_store.json) bằng hàm read_store(). File này chứa một dictionary ánh xạ record_id tới các đối tượng gói khóa.
- Chuỗi JSON key_package từ request được chuyển đổi thành đối tượng Python (dictionary).
- Đối tượng gói khóa này được thêm mới hoặc cập nhật vào dictionary dữ liệu đã đọc, sử dụng record_id làm khóa.
- Toàn bộ dictionary dữ liệu (đã cập nhật) được ghi lại vào file lưu trữ bằng hàm write_store().

– Phản hồi:

- Nếu thành công, TA trả về một thông báo xác nhận (HTTP 200 OK).
- Nếu có lỗi xảy ra (ví dụ: thiếu dữ liệu, lỗi đọc/ghi file), TA trả về mã lỗi HTTP tương ứng (400 Bad Request hoặc 500 Internal Server Error) cùng với chi tiết lỗi.

– Lưu trữ:

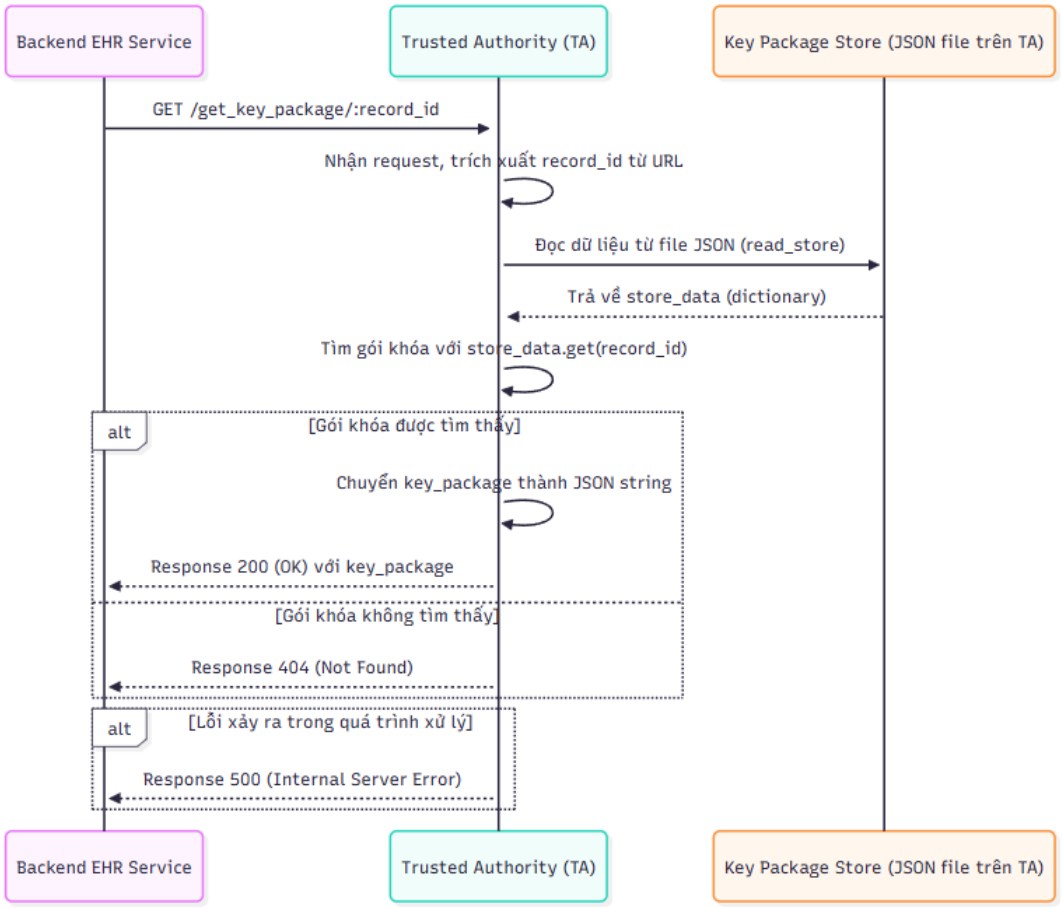
- Gói khóa được lưu trữ dưới dạng đối tượng JSON trong một file trên server TA, được quản lý bởi các hàm read_store() và write_store().

Bảng 8 - Hàm store_key_package

```
def store_key_package(record_id: str, key_package: dict):
    """
    Gửi gói khóa (key package) đến TA để lưu trữ.
    LƯU Ý: Yêu cầu endpoint /store_key_package phải được tạo bên phía ta_app.py.
```

```
    """
    # Convert bytes to base64 for JSON serialization
    serializable_package = {}
    for key, value in key_package.items():
        if isinstance(value, bytes):
            serializable_package[key] = base64.b64encode(value).decode('utf-8')
        else:
            serializable_package[key] = value

    res = requests.post(
        f"{current_app.config['TA_BASE_URL']}/store_key_package",
        json={
            "record_id": record_id,
             "key_package": json.dumps(serializable_package) # Gửi dưới dạng chuỗi JSON
        },
        verify=False
    )
    res.raise_for_status()
         current_app.logger.info(f"Successfully stored key package for record_id '{record_id}' on TA.")
```

7. Truy xuất ciphertext (get_ctdk).



Hình 17 - Truy xuất CTdk

– Mục đích:

- Cho phép hệ thống Backend EHR Service truy xuất lại "gói khóa" (key_package) đã được lưu trữ trước đó trên Trusted Authority (TA), dựa trên một record_id cụ thể. Gói khóa này cần thiết cho quá trình giải mã file EHR.

– Luồng hoạt động:

- Yêu cầu từ Backend: Khi Backend EHR Service cần giải mã một file EHR, nó gửi một yêu cầu GET đến endpoint /get_key_package/<record_id> của TA, với <record_id> là mã định danh của hồ sơ EHR cần truy xuất gói khóa.

– Xử lý tại TA:

- TA nhận request và trích xuất record_id từ đường dẫn URL.
- TA đọc nội dung của file lưu trữ gói khóa (ví dụ: secrets/key_package_store.json) bằng hàm read_store().

- TA tìm kiếm gói khóa trong dictionary dữ liệu đã đọc bằng cách sử dụng record_id làm khóa.

– Phản hồi:

- Tìm thấy: Nếu gói khóa tương ứng với record_id được tìm thấy, TA chuyển đổi đối tượng gói khóa đó thành một chuỗi JSON. Sau đó, TA trả về chuỗi JSON này cho Backend trong một đối tượng JSON (HTTP 200 OK, ví dụ: {"key_package": "chuỗi_json_của_gói_khóa"}).
- Không tìm thấy: Nếu không tìm thấy gói khóa nào cho record_id đã cho, TA trả về lỗi HTTP 404 Not Found.
- Lỗi khác: Nếu có lỗi xảy ra trong quá trình xử lý (ví dụ: lỗi đọc file), TA trả về lỗi HTTP 500 Internal Server Error.

– Sử dụng bởi Backend:

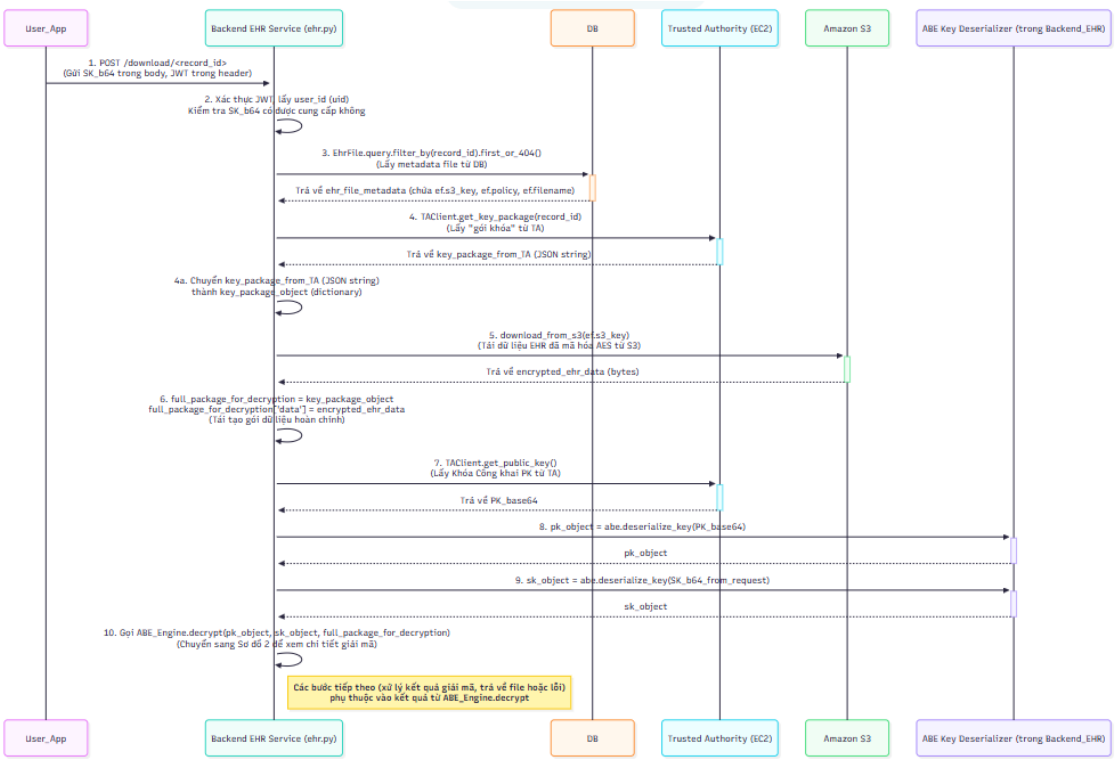
- Sau khi nhận được chuỗi JSON của gói khóa từ TA, Backend EHR Service sẽ chuyển đổi nó trở lại thành đối tượng Python. Các thành phần trong gói khóa này (IV, tag, metadata ABE, sig PQC) sau đó được sử dụng cùng với khóa công khai (PK), khóa (SK) của người dùng, và dữ liệu EHR đã mã hóa AES (tải từ S3) để thực hiện quá trình giải mã.

Bảng 9 - Hàm get_key_package

```
def get_key_package(record_id: str) -> dict:
    """
    Lấy lại gói khóa từ TA.
    LƯU Ý: Yêu cầu endpoint /get_key_package/<record_id> phải được tạo bên phía ta_app.py.
    """
    res = requests.get(
        f"{current_app.config['TA_BASE_URL']}/get_key_package/{record_id}",
        verify=False
    )
```

```
    res.raise_for_status()
    key_package_str = res.json()['key_package']
    serializable_package = json.loads(key_package_str)

    # Convert base64 back to bytes for known bytes fields
    key_package = {}
    bytes_fields = ['iv', 'tag', 'encrypted_key', 'signature', 'sig', 'ct_key', 'data']

    for key, value in serializable_package.items():
        if key in bytes_fields and isinstance(value, str):
            try:
                key_package[key] = base64.b64decode(value)
            except:
                key_package[key] = value  # Keep original if decode fails
        else:
            key_package[key] = value

    return key_package
```

8. Luồng tải dữ liệu xuống và giải mã.

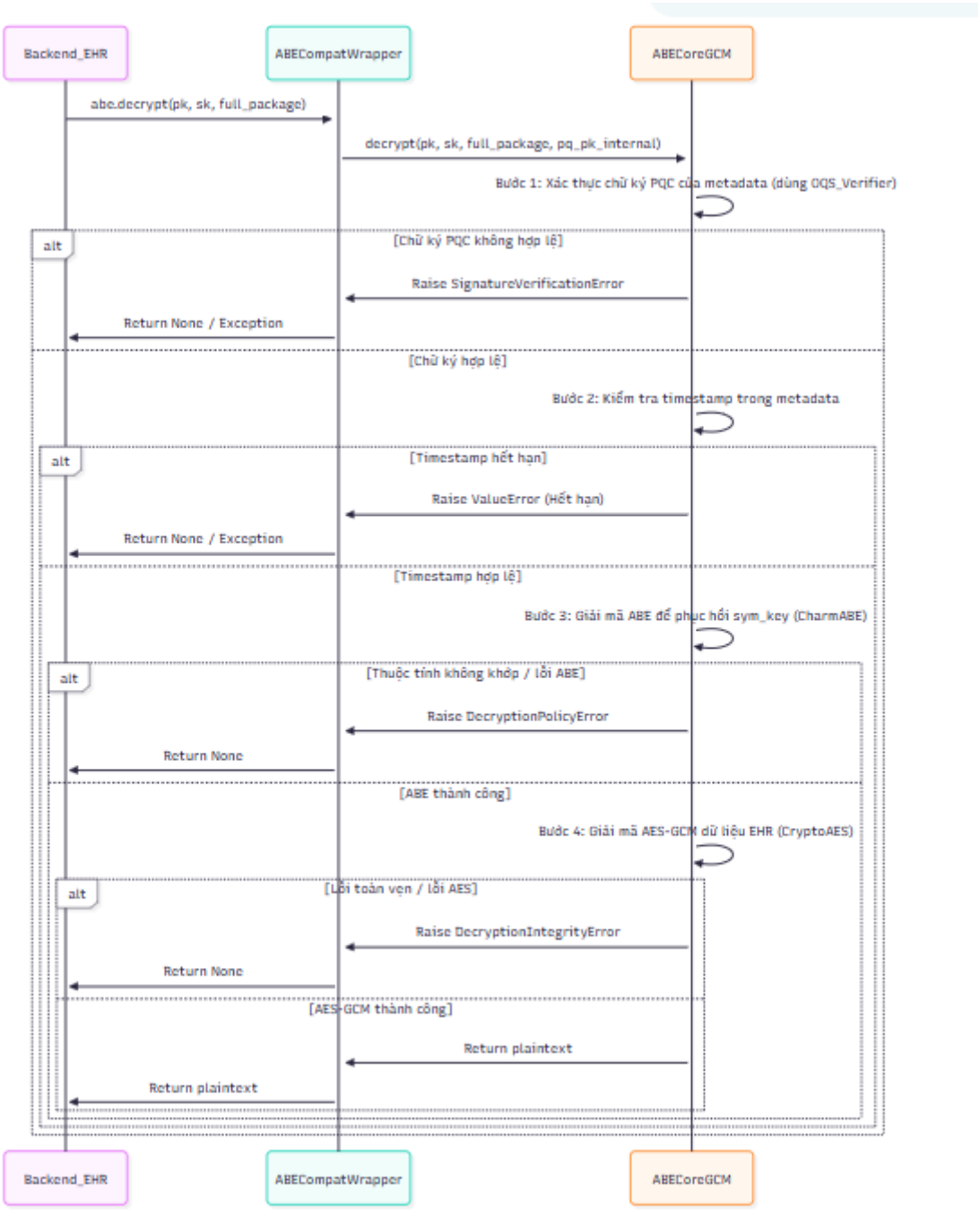


Hình 18 - Luồng tải dữ liệu

– Yêu cầu tải xuống từ người dùng (User_App -> Backend_EHR):
  - Người dùng, thông qua ứng dụng, gửi yêu cầu POST đến endpoint /api/ehr/download/<record_id> của Backend EHR Service.
  - Yêu cầu này phải được xác thực bằng JWT (JSON Web Token).
  - Body của request chứa khóa (SK) của người dùng, đã được mã hóa base64 (sk_b64).
– Xác thực và chuẩn bị tại Backend EHR Service (ehr.download):
  - Backend xác thực JWT và lấy user_id (uid) của người dùng.
  - Truy vấn cơ sở dữ liệu (DB) để lấy thông tin metadata của file EHR (đối tượng EhrFile) dựa trên record_id được cung cấp. Thông tin này bao gồm s3_key (đường dẫn đến file trên S3) và chính sách truy cập gốc (ef.policy).
  - Kiểm tra xem sk_b64 có được cung cấp trong request không.
– Thu thập các thành phần giải mã (ehr.download):
  - Lấy Key Package từ Trusted Authority (TA):
    - Backend gọi phương thức TAClient.get_key_package(record_id).

    - TAClient gửi request GET đến endpoint /get_key_package/<record_id> của TA.
    - TA đọc từ nơi lưu trữ của mình (ví dụ: file JSON key_package_store.json) và trả về key_package (dưới dạng chuỗi JSON) cho Backend.
    - Backend chuyển đổi chuỗi JSON này thành đối tượng Python (dictionary). key_package này chứa iv, tag (của AES-GCM), metadata (bao gồm policy, user_id gốc, timestamp, abe_key_hex), và sig (chữ ký PQC của metadata).

– Tải dữ liệu EHR đã Mã hóa AES từ S3:
    - Backend gọi hàm download_from_s3(ef.s3_key).
    - Hàm này sử dụng boto3.client('s3') để tải nội dung file từ bucket S3 được chỉ định, dựa trên s3_key. Kết quả là encrypted_ehr_data (dữ liệu EHR đã được mã hóa bằng AES-GCM, dạng bytes).
– Lấy khóa công khai (PK) từ TA:
    - Backend gọi TAClient.get_public_key().
    - TAClient gửi request GET đến endpoint /get_public_key của TA.
    - TA trả về khóa công khai (PK) của hệ thống ABE dưới dạng chuỗi base64 (pk_base64).
– Chuẩn bị dữ liệu và khóa cho ABE Engine (ehr.download):
    - Tái tạo gói dữ liệu Hoàn chỉnh: encrypted_ehr_data (từ S3) được thêm vào key_package (từ TA) để tạo thành full_package_for_decryption mà hàm abe.decrypt mong đợi.
– Deserialize Khóa:
    - pk_base64 (từ TA) được deserialize thành đối tượng khóa công khai (pk_object) bằng abe.deserialize_key().
    - sk_b64 (từ request của người dùng) được deserialize thành đối tượng khóa Bí Mật (sk_object) bằng abe.deserialize_key().



Hình 19 - Giải mã EHR

- Thực hiện Giải mã Toàn diện (Gọi abe.decrypt - ABECompatWrapper.decrypt -> ABECoreGCM.decrypt):
- Backend gọi hàm plaintext = abe.decrypt(pk_object, sk_object, full_package_for_decryption).
- Bên trong ABECompatWrapper.decrypt, nó gọi self.gcm_core.decrypt(pk, sk, ct, self.pq_pk). ct ở đây là full_package_for_decryption, và self.pq_pk là khóa công khai PQC của ABE engine (dùng để xác minh chữ ký).
- Các bước chi tiết bên trong ABECoreGCM.decrypt:

- **Bước 1:** Xác thực Chữ ký Metadata PQC:
  - Metadata từ ct['metadata'] được chuẩn hóa thành chuỗi JSON.
  - Chữ ký PQC ct['sig'] được xác thực bằng khóa công khai PQC (pq_verification_key - chính là self.pq_pk của wrapper).
  - Nếu xác thực thất bại, SignatureVerificationError được raise.
- Bước 2: Kiểm tra Timestamp Hết hạn:
  - timestamp từ ct['metadata']['timestamp'] được kiểm tra. Nếu dữ liệu đã quá thời gian quy định (ví dụ: 7 ngày), lỗi ValueError được raise.
- Bước 3: Phục hồi khóa Đối xứng sym_key bằng Giải mã ABE:
  - abe_key_hex từ ct['metadata']['abe_key_hex'] được chuyển đổi từ hex sang bytes, rồi thành đối tượng abe_key_obj.
  - Hàm self.abe.decrypt(pk, sk, abe_key_obj) (của Charm CP-ABE) được gọi. Hàm này sử dụng khóa công khai (pk), khóa Bí Mật của người dùng (sk), và abe_key_obj (chứa khóa đối xứng đã mã hóa ABE).
  - Nếu các thuộc tính trong sk của người dùng thỏa mãn chính sách được nhúng trong abe_key_obj, hàm này sẽ trả về sym_key gốc (plaintext của khóa đối xứng).
  - Nếu không thỏa mãn, nó trả về None (hoặc tương đương), và DecryptionPolicyError được raise.
- Bước 4: Giải mã dữ liệu AES-GCM:
  - Khóa AES (aes_key) được dẫn xuất lại từ sym_key (đã giải mã thành công) bằng hàm self._derive_key().
  - Một đối tượng giải mã AES-GCM (cipher) được khởi tạo với aes_key và ct['iv'] (IV từ gói khóa).
  - Hàm cipher.decrypt_and_verify(ct['data'], ct['tag']) được gọi. ct['data'] là encrypted_ehr_data (từ S3), và ct['tag'] là thẻ xác thực AES-GCM (từ gói khóa).
  - Hàm này sẽ giải mã ct['data'] và đồng thời kiểm tra ct['tag']. Nếu tag không khớp (dữ liệu đã bị thay đổi hoặc khóa/IV sai), ValueError (dẫn đến DecryptionIntegrityError) được raise.

    - Nếu thành công, hàm trả về dữ liệu gốc đã giải mã (plaintext dạng bytes).

– Xử lý Kết quả Giải mã và Trả về cho Người dùng (ehr.download):

- Giải mã Thành công: Nếu plaintext không phải là None (tức là tất cả các bước giải mã đều thành công), Backend tạo một HTTP Response chứa plaintext. Headers được thiết lập để trình duyệt tải file về (Content-Disposition: attachment).
- Giải mã Thất bại (Access Denied): Nếu plaintext là None (thường do DecryptionPolicyError từ ABE), Backend trả về lỗi HTTP 403 Access Denied, thông báo rằng thuộc tính của người dùng không thỏa mãn chính sách.
- Các Lỗi Khác: Các ngoại lệ khác (ví dụ: FileNotFoundError từ S3, requests.exceptions.RequestException khi gọi TA, hoặc các lỗi ABEError khác) được bắt và trả về mã lỗi HTTP tương ứng (404, 503, 500).

8. Triển khai & vận hành.

8.1. Môi trường triển khai (Deployment Environment).

Hệ thống được phân chia thành hai phần chính trên hai nền tảng đám mây khác nhau:

A. Phía Amazon Web Services (AWS) - Lõi Bảo mật và Quản lý Khóa.

Phần hạ tầng đòi hỏi bảo mật cao nhất, đặc biệt là việc quản lý khóa, được đặt trên AWS để tận dụng các dịch vụ chuyên dụng.

- **Trusted Authority (TA):**
    - Nền tảng: Triển khai trên một máy chủ ảo Amazon EC2.
    - Mạng: Được đặt trong một mạng con riêng (private subnet) của Amazon VPC và được bảo vệ bởi Security Group, chỉ cho phép kết nối từ máy chủ Backend.
- **Quản lý Khóa Chủ (MK):**
    - AWS Key Management Service (KMS): Một Khóa Khách hàng Quản lý (CMK - Customer Managed Key) được tạo riêng để mã hóa và bảo vệ Khóa Chủ của hệ thống ABE.

- AWS Secrets Manager: Dùng để lưu trữ phiên bản đã mã hóa bằng KMS của Khóa Chủ (MK), giúp TA truy xuất một cách an toàn.

B. Phía DigitalOcean - Cổng Tương tác và Logic Nghiệp vụ

Phần giao tiếp trực tiếp với người dùng và xử lý các logic nghiệp vụ chính được đặt trên DigitalOcean.

- **Backend EHR Service (API Gateway):**
  - Nền tảng: Triển khai trên một DigitalOcean Droplet.
  - Web Server / Reverse Proxy: Sử dụng Nginx để xử lý các kết nối HTTPS từ người dùng (với chứng chỉ SSL từ Let's Encrypt) và chuyển tiếp yêu cầu đến ứng dụng Flask (chạy bằng Gunicorn).
- **Lưu trữ Dữ liệu:**
  - Dữ liệu EHR đã mã hóa: Được lưu trữ trên dịch vụ Amazon S3 để đảm bảo độ tin cậy và khả năng mở rộng. Droplet được cấp quyền truy cập S3 thông qua một cặp IAM access key được cấu hình an toàn.
  - Siêu dữ liệu (Metadata): Thông tin người dùng và metadata của các tệp EHR được lưu trữ an toàn trong một cơ sở dữ liệu PostgreSQL và Amazon S3.

8.2. Quy trình triển khai.

Quy trình triển khai được thực hiện tuần tự như sau:

1. **Cấu hình Hạ tầng AWS:**
   - Tạo VPC, private subnet, và máy chủ EC2 cho TA.
   - Tạo CMK trong AWS KMS và một secret tương ứng trong AWS Secrets Manager.
   - Cấu hình Security Group để cho phép Droplet Backend kết nối vào EC2 của TA.
   - Thực hiện Setup lần đầu: Tạo cặp khóa PK/MK và lưu trữ chúng một cách an toàn.
2. **Triển khai Trusted Authority (TA) lên AWS:**

- Kết nối vào máy chủ EC2, cài đặt các thư viện Python cần thiết (Flask, charm-crypto, oqs, boto3, gunicorn).
- Cấu hình các biến môi trường cho việc kết nối AWS.
- Chạy ứng dụng TA bằng Gunicorn.

3. **Cấu hình Hạ tầng DigitalOcean:**
   - Tạo Droplet cho Backend Service và cấu hình DNS.
   - Chuẩn bị sẵn sàng CSDL PostgreSQL và lấy chuỗi kết nối.
4. **Triển khai Backend Service và Nginx lên DigitalOcean:**
   - Kết nối vào Droplet, cài đặt các thư viện Python, Nginx và Gunicorn.
   - Cấu hình các biến môi trường cho ứng dụng Backend (chuỗi kết nối CSDL, URL của TA Service, thông tin S3).
   - Cấu hình Nginx làm reverse proxy và thiết lập SSL với Certbot (Let's Encrypt).
   - Chạy ứng dụng Backend bằng Gunicorn và khởi động Nginx.

8.3. Vận hành và thực thi kịch bản.

Sau khi triển khai thành công, nhóm tiến hành vận hành hệ thống để kiểm chứng các kịch bản:

1. **Khởi tạo Người dùng và Cấp khóa:**
   - Quản trị viên tạo tài khoản và gán thuộc tính cho người dùng trong CSDL PostgreSQL.
   - Khi người dùng yêu cầu, Backend trên DigitalOcean sẽ giao tiếp với TA trên AWS để tạo và cấp khóa bí mật (SK).
2. **Thực thi Kịch bản Tải lên (Upload):**
   - Bệnh nhân gửi tệp và chính sách đến Backend Service trên DigitalOcean (như Hình 25).
   - Backend giao tiếp với TA trên AWS để lấy khóa công khai.

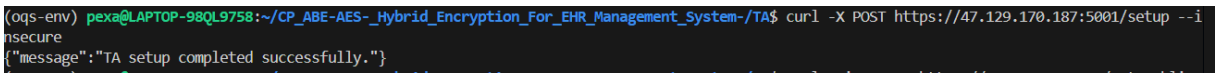
- Sau khi thực hiện mã hóa lai, Backend gửi gói khóa (key package) về cho TA trên AWS để lưu trữ, đồng thời tải tệp EHR đã mã hóa lên Amazon S3.
- Hệ thống trả về thông báo thành công (như Hình 26).

3. **Thực thi Kịch bản Tải xuống (Download):**
   - Bác sĩ gửi yêu cầu tải tệp cùng với khóa bí mật (SK) của mình đến Backend.
   - Backend lấy gói khóa từ TA (trên AWS) và tệp mã hóa từ S3.
   - Toàn bộ quá trình giải mã (xác thực chữ ký Dilithium, giải mã ABE, giải mã AES) được thực hiện trên Droplet của DigitalOcean.
   - Nếu thành công, tệp gốc được trả về cho Bác sĩ (như Hình 28). Nếu không, lỗi truy cập bị từ chối được trả về (như Hình 29).

9. Kết quả & Nhận xét.

```
2025-06-25 15:15:27,979 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,008 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,033 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,058 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,083 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,107 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,132 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,157 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,182 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,206 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
2025-06-25 15:15:28,234 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'
    Đang thực hiện lần lặp thứ 1000/1000...
2025-06-25 15:15:28,259 - INFO - Đang mã hóa cho user 'perf_user_001' với chính sách: '((DOCTOR AND HOSPITALA) OR (DOCTOR AND CARDIOLOGY)) AND N
URSE'

[4] Hoàn thành đo hiệu năng.

--- Kết quả Mã hóa ---
    Tổng số lần mã hóa thành công: 1000/1000
    Tổng thời gian mã hóa: 18.957173 giây
    Thời gian mã hóa trung bình: 0.018957 giây/lần
    Thông lượng mã hóa (ước tính): 52.750480918995166 lần/giây

--- Kết quả Giải mã ---
    Tổng số lần giải mã thành công: 1000/1000
    Tổng thời gian giải mã: 7.302507 giây
    Thời gian giải mã trung bình: 0.007303 giây/lần
    Thông lượng giải mã (ước tính): 136.9392761781016 lần/giây
(oqs-env) pexa@LAPTOP-98QL9758:~/CP_ABE-AES-_Hybrid_Encryption_For_EHR_Management_System-/TA$
```

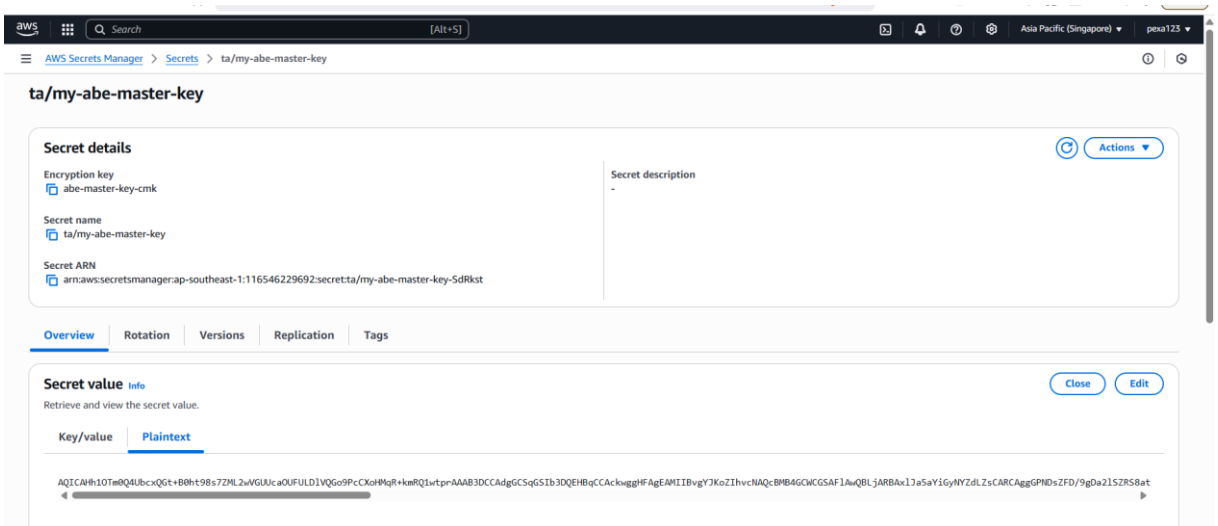
Hình 20 - Trung bình thời gian mã hóa và giải mã



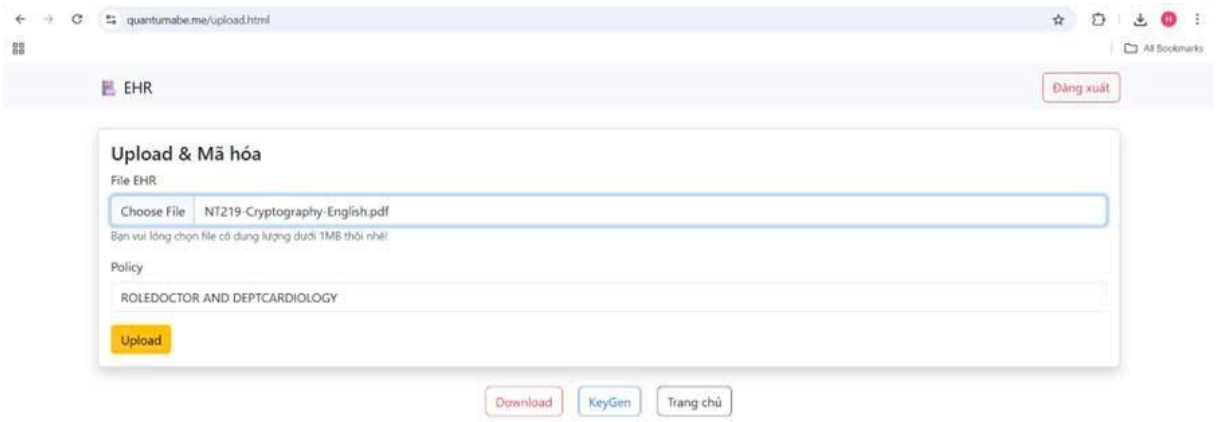
Hình 21 - setup()

Hình 22 – Deploy và get publickey

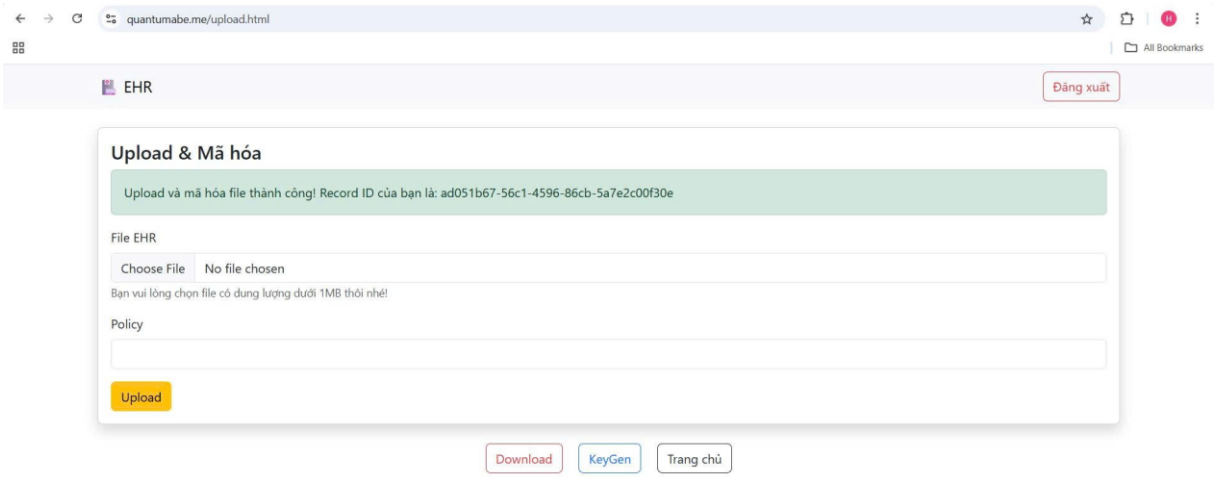
Hình 23 - get secret key



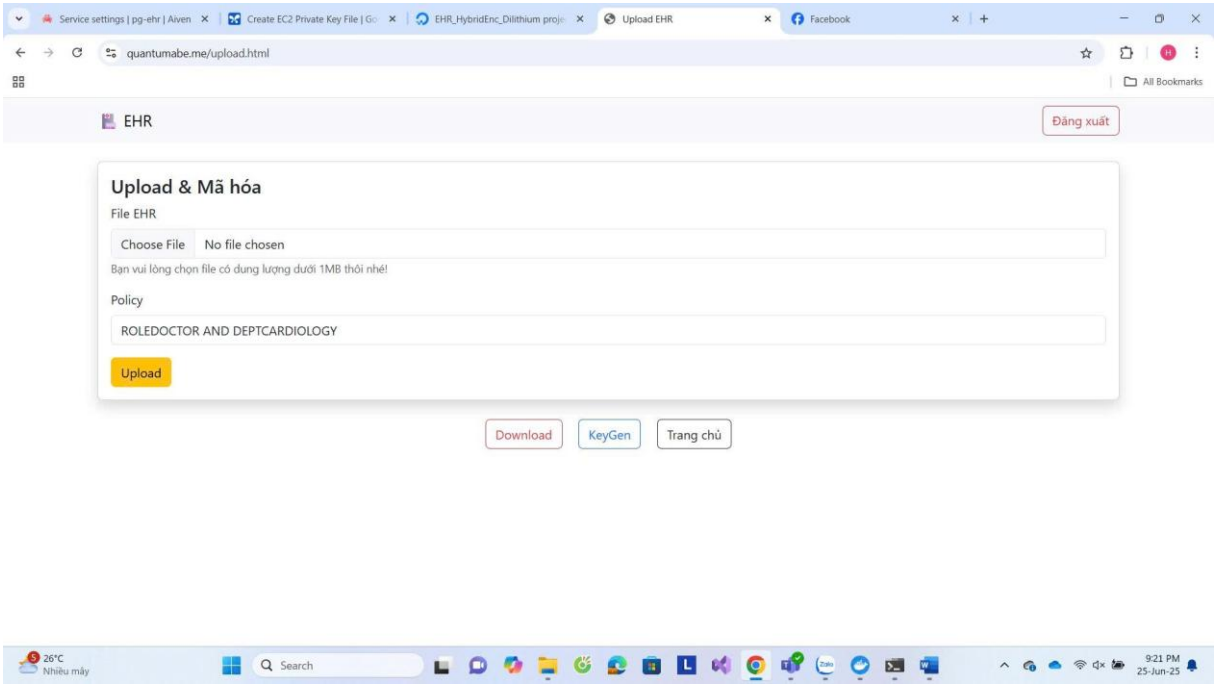
Hình 24 - Lưu master key



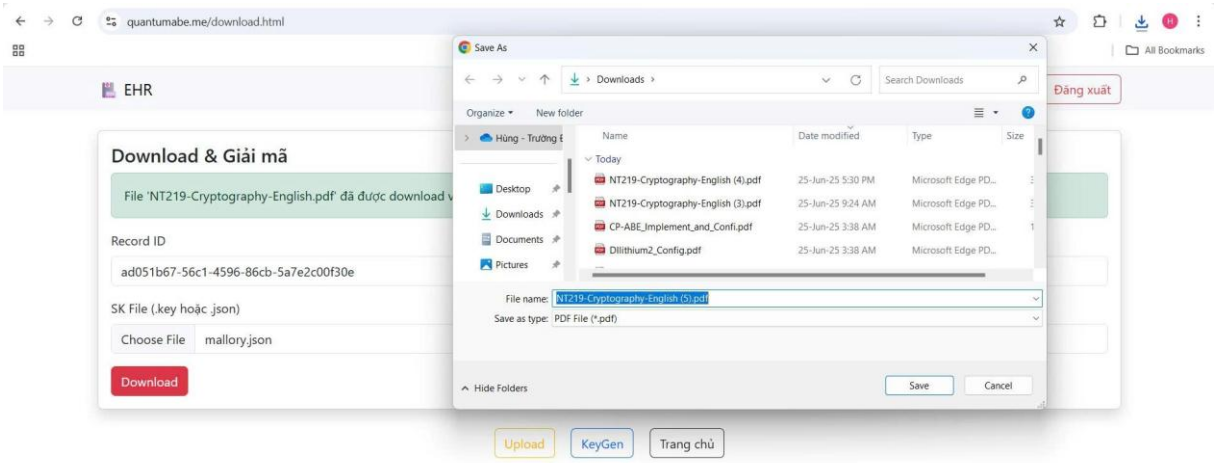
Hình 25 - Người dùng A có role là Doctor và department là Department là Cardiology upload file



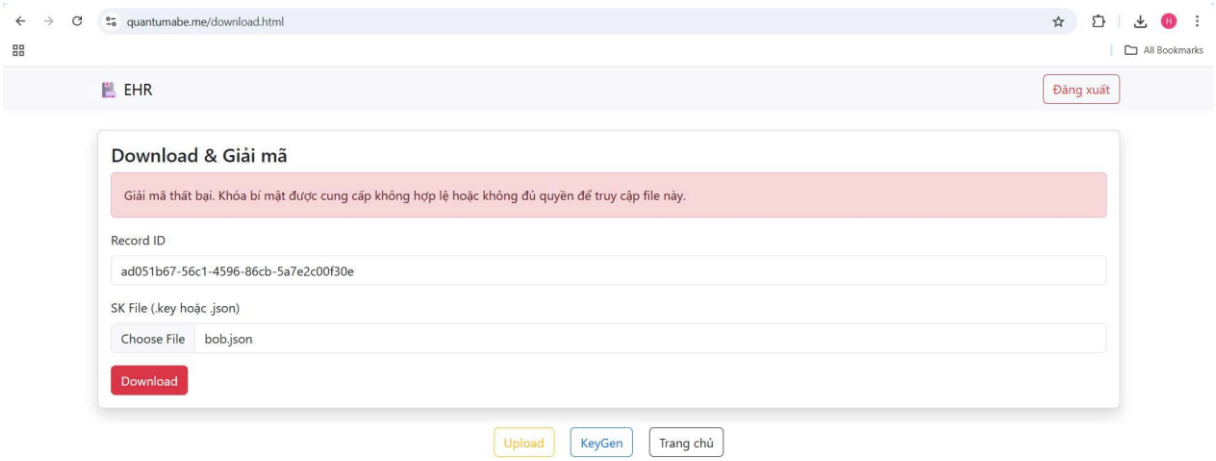
Hình 26 - Người dùng A upload file thành công



Hình 27 - Người dùng B



Hình 28 - Người dùng B có cùng role và department với người dùng A tải file được



Hình 29 - Người dùng C khác thuộc tính với người dùng A nên không tải được file về