

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG
MÔN MẬT MÃ HỌC



BÁO CÁO ĐỒ ÁN CUỐI KỲ

**HỆ THỐNG ĐÁNH GIÁ ĐIỂM TÍN DỤNG LIÊN NGÂN HÀNG &
TỔ CHỨC TÀI CHÍNH ỦNG DỤNG HOMOMORPHIC
ENCRYPTION**

Chủ đề: Application of Homomorphic Encryption for Financial Services

Link repository của đồ án: https://github.com/Hacles/NT219_Project

Nhóm thực hiện: Nhóm 11

Nguyễn Việt Tùng	23521746	Trưởng nhóm
Sit Khải Đông	23520299	Thành viên

Giảng viên hướng dẫn: TS. Nguyễn Ngọc Tự

Thời gian thực hiện: 3/2025 – 6/2025

Thành phố Hồ Chí Minh, ngày 22 tháng 6 năm 2025

LỜI CẢM ƠN

Các thành viên trong nhóm 11 muốn gửi lời cảm ơn chân thành sâu sắc tới giảng viên hướng dẫn TS. Nguyễn Ngọc Tự đã tận tình hỗ trợ nhóm để hoàn thành đồ án này. Qua đồ án này, nhóm đã tích lũy được nhiều kiến thức bổ ích về chuyên môn cũng như rèn luyện kỹ năng làm việc nhóm.

Mặc dù các thành viên trong nhóm đã cố gắng hoàn thành đồ án một cách hoàn thiện nhất nhưng do thiếu sót về kinh nghiệm cũng như một số hạn chế khác nên đồ án này cũng không thể tránh khỏi các sai sót. Nhóm rất mong nhận được sự cảm thông, chia sẻ và góp ý từ thầy.

Nhóm 11 xin chân thành cảm ơn!

TÓM TẮT

Hệ thống này được thiết kế nhằm giải quyết bài toán đánh giá điểm tín dụng giữa các ngân hàng và tổ chức tài chính trong bối cảnh hiện đại, nơi bảo mật và quyền riêng tư của dữ liệu khách hàng được đặt lên hàng đầu. Thông qua việc ứng dụng các kỹ thuật mã hóa đồng cấu (Homomorphic Encryption) tiên tiến, hệ thống cho phép nhiều bên cùng hợp tác, trao đổi và xử lý thông tin tín dụng mà không cần tiết lộ dữ liệu gốc.

Kiến trúc hệ thống bao gồm nhiều thành phần độc lập như Cơ quan chứng thực (CA), các ngân hàng, tổ chức tài chính và các dịch vụ liên ngân hàng, tất cả được kết nối thông qua một hạ tầng Internet bảo mật. Mỗi ngân hàng sở hữu khóa giải mã riêng và phối hợp với các bên còn lại để tạo ra khóa mã hóa và các khóa phụ dùng chung. Tổ chức tài chính có thể thực hiện các phép tính theo công thức độc quyền trên dữ liệu đã mã hóa mà không cần truy cập trực tiếp vào nội dung ban đầu; kết quả được trả về dưới dạng mã hóa. Dữ liệu chỉ được giải mã khi có sự đồng thuận của tất cả ngân hàng liên quan, đảm bảo tính bảo mật, minh bạch và phân quyền trong xử lý thông tin.

Hệ thống tuân thủ các nguyên tắc bảo mật như truyền tải dữ liệu an toàn qua giao thức TLS, sử dụng các thuật toán mã hóa mạnh như AES, ECDSA, và lưu trữ dữ liệu trong cơ sở dữ liệu hỗ trợ mã hóa toàn bộ (TDE). Toàn bộ quy trình – từ xác thực, truyền nhận dữ liệu đến tính toán điểm tín dụng – đều được thiết kế để đảm bảo các yếu tố: bảo mật (Confidentiality), toàn vẹn (Integrity), xác thực (Authentication), sẵn sàng (Availability) và không thể chối bỏ (Non-repudiation).

Với sự kết hợp giữa công nghệ mã hóa đồng cấu, kiến trúc phân tán và các tiêu chuẩn bảo mật hiện đại, hệ thống không chỉ nâng cao hiệu quả hợp tác giữa các tổ chức tài chính mà còn bảo vệ tối đa quyền riêng tư và lợi ích của khách hàng trong kỷ nguyên số.

Mục lục

Chương 1: Tổng quan	5
I. Giới thiệu đề tài.....	5
II. Kịch bản ứng dụng.....	5
III. Tài sản, chủ sở hữu tài sản và các bên liên quan.....	5
IV. Rủi ro.....	7
V. Mục tiêu:.....	8
Chương 2: Kiến trúc hệ thống	9
I. Các thuật toán mã hóa, giao thức và tiêu chuẩn sử dụng.....	9
II. Kiến trúc hệ thống.....	13
III. Thuật toán của tổ chức tài chính:.....	14
Chương 3: Triển khai thực tế.....	16
I. Xây dựng hệ thống chung.....	16
II. Các bước triển khai:.....	16
III. Triển khai chi tiết:.....	17
IV. Mục tiêu của quy trình.....	34
Chương 4: Kết quả và nhận xét.....	35
Tài liệu tham khảo.....	36

Chương 1: Tổng quan

I. Giới thiệu về tài

Trong kỷ nguyên số, bảo mật và quyền riêng tư dữ liệu khách hàng là ưu tiên hàng đầu của các tổ chức tài chính. Đề tài “Application of Homomorphic Encryption for Financial Services” nhằm nghiên cứu cách thức công nghệ Homomorphic Encryption (HE) có thể giúp các ngân hàng và tổ chức tài chính xử lý, phân tích dữ liệu nhạy cảm như điểm tín dụng mà không cần giải mã. Điều này không chỉ tăng cường bảo mật, mà còn mở ra hướng tiếp cận mới cho việc chia sẻ và xử lý dữ liệu liên ngân hàng một cách an toàn, hiệu quả trong môi trường hiện đại.

II. Kịch bản ứng dụng

Trong lĩnh vực tài chính, nhiều ngân hàng muốn cùng đánh giá điểm tín dụng của một khách hàng để ra quyết định cho vay hoặc đề xuất sản phẩm. Tuy nhiên, mỗi bên chỉ nắm giữ một phần dữ liệu và không ai có đầy đủ thông tin để tự đánh giá chính xác. Vì vậy, một tổ chức trung gian (như công ty xếp hạng tín dụng) được giao nhiệm vụ tổng hợp và tính toán điểm tín dụng. Tuy nhiên, do tuân thủ các quy định bảo mật như GDPR, CCPA và chính sách nội bộ, các ngân hàng không thể chia sẻ dữ liệu khách hàng dưới dạng thô. Giải pháp là sử dụng mã hóa đồng hình (Homomorphic Encryption) – cho phép thực hiện tính toán trực tiếp trên dữ liệu đã mã hóa. Nhờ đó, tổ chức trung gian có thể xử lý và đưa ra điểm tín dụng cuối cùng mà không cần truy cập dữ liệu gốc, đảm bảo cả hiệu quả và tính riêng tư.

III. Tài sản, chủ sở hữu tài sản và các bên liên quan

Tài sản & chủ sở hữu tài sản

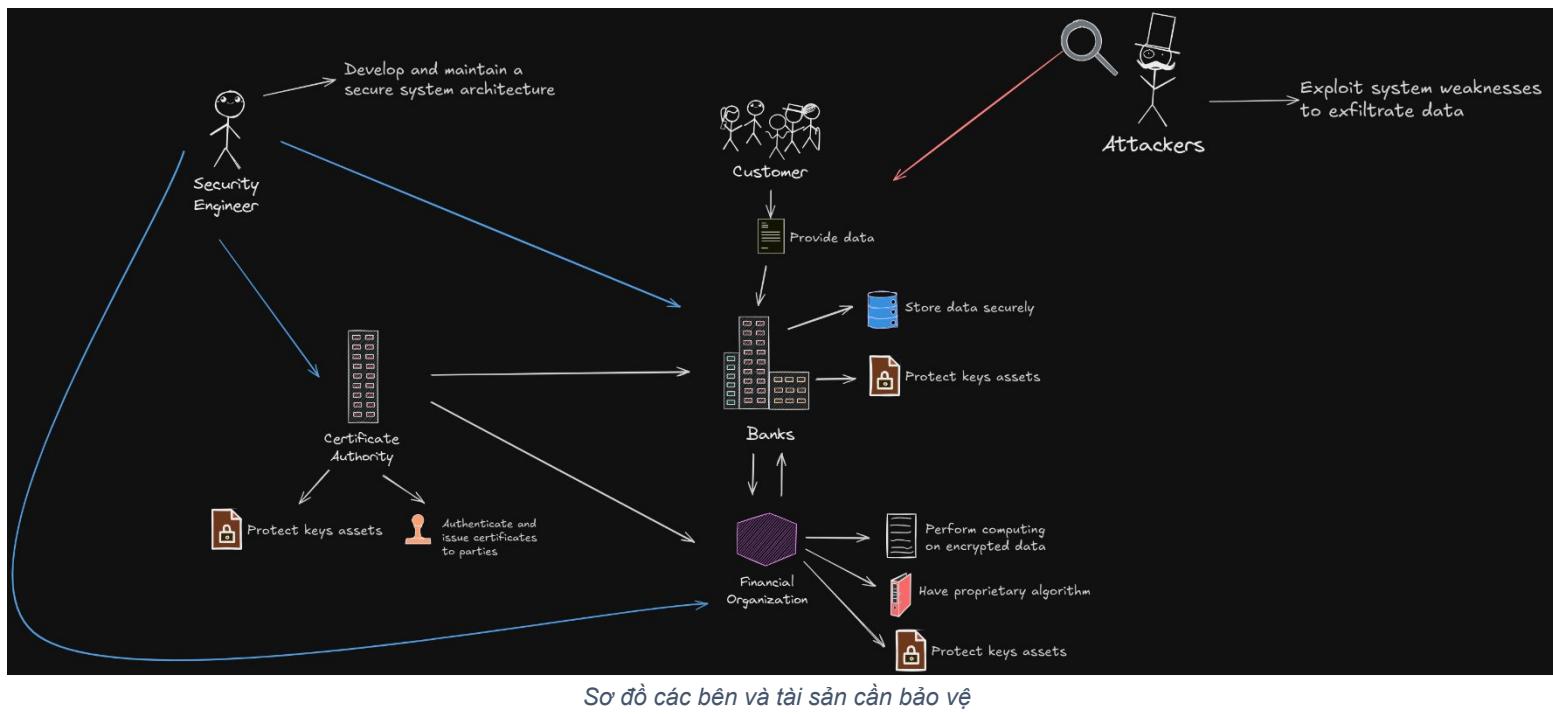
- Dữ liệu khách hàng:
 - Do khách hàng của các ngân hàng tạo ra.
 - Ngân hàng là bên sở hữu hợp pháp, chịu trách nhiệm lưu trữ và bảo vệ thông tin này.
- Khóa mật mã:
 - Khóa ECC riêng tư: Mỗi node trong hệ thống đều sở hữu, có trách nhiệm bảo vệ.
 - Khóa mã hóa đồng cấu (Riêng tư): Mỗi ngân hàng đều sở hữu, có trách nhiệm bảo vệ.

- Khóa mã hóa cơ sở dữ liệu: Mọi ngân hàng đều sở hữu, có trách nhiệm bảo vệ.
 - Khóa mã hóa đồng cấu (Công khai): Mọi ngân hàng đều dùng chung, dùng để mã hóa.
 - Khóa nhân đồng cấu (Công khai): Mọi ngân hàng và tổ chức tài chính đều biết, dùng để tính toán.
- **Chứng chỉ số (Certificate):**
- Root Certificate: Chứng chỉ chứng thực của CA, ai cũng sở hữu
 - TLS Certificate: Chứng chỉ của Server, do CA cấp, để thiết lập HTTPS và ký thông tin
- **Cơ sở hạ tầng:**
- Các node ngân hàng, tổ chức tài chính, CA.
 - Cơ sở dữ liệu, phần mềm xử lý mã hóa đồng hình (HE).
 - Hạ tầng Internet kết nối giữa các node.
- **Công thức đánh giá điểm tài chính: Độc quyền của tổ chức tài chính.**

Các bên liên quan

- **Ngân hàng thành viên (ACB, MSB...)**
 - Gửi dữ liệu khách hàng đã mã hóa, tham gia tạo khóa và giải mã
 - Vai trò trong hệ thống: Data Custodian, System Node Operator
- **Tổ chức tài chính trung gian (FE Credit)**
 - Nhận dữ liệu mã hóa, đánh giá điểm tín dụng, trả về kết quả
 - Vai trò trong hệ thống: Evaluator, Data Processor
- **Certificate Authority (Ngân hàng Nhà nước)**
 - Cấp và xác thực chứng chỉ cho các bên
 - Vai trò: Trust Anchor, Security Authority
- **Khách hàng**
 - Dữ liệu của họ được xử lý, là người chịu ảnh hưởng trực tiếp
 - Vai trò trong hệ thống: End-User, Data Owner
- **Tin tặc**
 - Mối đe dọa, tìm cách tấn công và đánh cắp dữ liệu
 - Vai trò trong hệ thống: Attacker
- **Kỹ sư bảo mật**
 - Xây dựng hệ thống giao tiếp bảo mật giữa các node
 - Vai trò trong hệ thống: Security Engineer, System Architect

❖ Sơ đồ:



IV. Rủi ro

Mối đe dọa	Mô tả ngắn	Ví dụ trong hệ thống của em
S – Spoofing (Giả mạo)	Kẻ tấn công mạo danh một thực thể hợp lệ.	<ul style="list-style-type: none"> - Giả mạo ngân hàng để gửi dữ liệu giả mạo đến tổ chức tài chính. - Mạo danh CA để cấp chứng chỉ giả. - Giả mạo tổ chức tài chính để trả về kết quả sai lệch.
T – Tampering (Thay đổi dữ liệu)	Dữ liệu bị thay đổi khi truyền hoặc lưu trữ.	<ul style="list-style-type: none"> - Sửa nội dung dữ liệu mã hóa trong quá trình truyền giữa ngân hàng và tổ chức tài chính. - Thay đổi khóa mã hóa hoặc chứng chỉ trong kho lưu trữ ngân hàng.
R – Repudiation (Chối bỏ hành vi)	Bên tham gia phủ nhận hành động của mình.	<ul style="list-style-type: none"> - Ngân hàng từ chối việc đã gửi dữ liệu cho tổ chức tài chính. - Tổ chức tài chính từ chối trách nhiệm nếu xảy ra rò rỉ dữ liệu. - CA từ chối việc đã cấp chứng chỉ.
I – Information Disclosure (Rò rỉ thông tin)	Lộ thông tin nhạy cảm cho bên không được phép.	<ul style="list-style-type: none"> - Attacker giải mã được dữ liệu khách hàng nếu khoá bị lộ. - Lộ khóa riêng từ bất kỳ node nào (Bank, CA, Financial Org). - TLS bị tấn công nếu chứng chỉ không hợp lệ.
D – Denial of Service (DoS) (Tù chối dịch vụ)	Gây gián đoạn hoạt động hệ thống.	<ul style="list-style-type: none"> - Flood CA bằng yêu cầu cấp chứng chỉ giả mạo. - Làm gián đoạn kênh HTTPS giữa các node.

Mối đe dọa	Mô tả ngắn	Ví dụ trong hệ thống của em
		<ul style="list-style-type: none"> - Tấn công khiến tổ chức tài chính không thể tính toán điểm tín dụng.
E – Elevation of Privilege (<i>Tăng quyền truy cập trái phép</i>)	Kẻ tấn công leo thang quyền để kiểm soát hệ thống.	<ul style="list-style-type: none"> - Attacker chiếm quyền root trong máy chủ ngân hàng. - Truy cập trái phép vào hệ thống CA để ký chứng chỉ giả. - Truy cập trái phép vào tổ chức tài chính để thay đổi thuật toán đánh giá. - Một ngân hàng có thể vượt quyền, tự mã hóa/giải mã. - Tổ chức tài chính có thể giải mã thông tin.

V. Mục tiêu:

Thiết kế hệ thống bảo mật toàn diện từ đầu đến cuối:

- Dữ liệu khách hàng được lưu trữ an toàn trong cơ sở dữ liệu mã hóa.
- Tất cả các bên tham gia hệ thống bắt buộc phải sở hữu chứng chỉ số hợp lệ để xác thực danh tính.
- Mọi dữ liệu trao đổi trong hệ thống đều phải được ký số để đảm bảo tính toàn vẹn và xác minh nguồn gốc.
- Tất cả dữ liệu đầu vào phải được xác thực trước khi xử lý nhằm ngăn chặn giả mạo hoặc dữ liệu bất hợp pháp.
- Chỉ những thực thể có quyền hạn phù hợp mới được phép truy cập thông tin, đảm bảo nguyên tắc phân quyền và kiểm soát truy cập chặt chẽ.

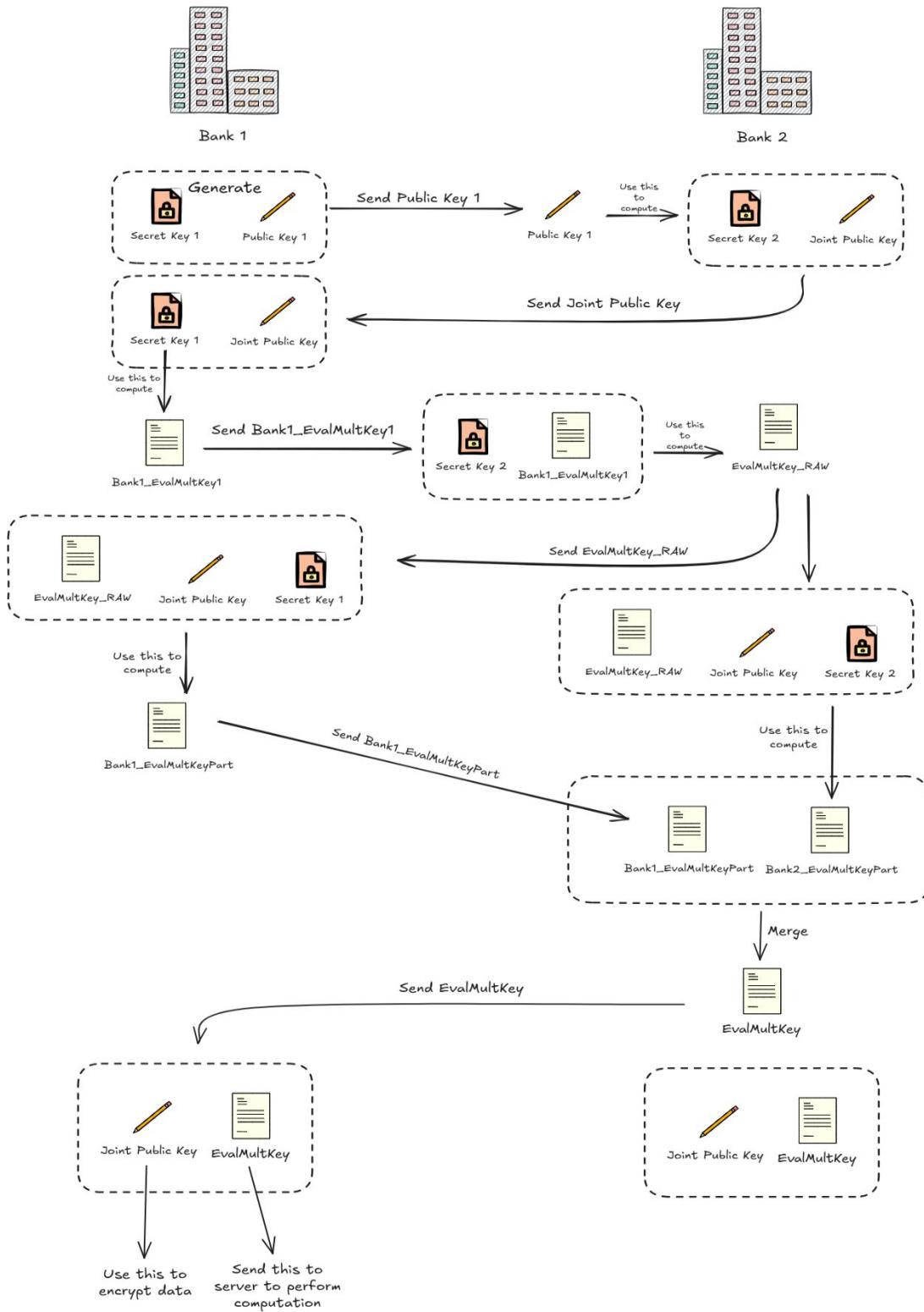
Chương 2: Kiến trúc hệ thống

I. Các thuật toán mã hóa, giao thức và tiêu chuẩn sử dụng

1. Mã hóa đồng cấu (Homomorphic Encryption)

- Là kỹ thuật cho phép thực hiện các phép tính toán trên dữ liệu đã mã hóa. Kết quả sau tính toán vẫn ở dạng mã hóa, và khi được giải mã, kết quả tương đương với việc thực hiện các phép toán đó trên dữ liệu gốc.
- Đây là thuật toán mã hóa bất đối xứng (asymmetric).
- Với bài toán đánh giá điểm tín dụng, cần phải sử dụng các công thức tính toán số thực phức tạp, nhóm em sử dụng thuật toán CKKS (Cheon – Kim – Kim – Song). Đây là một thuật toán FHE cho phép thực hiện các phép toán trên số thực, với một sai số chấp nhận được.
- Tuy nhiên, một thách thức đặt ra là: nếu tất cả các bên sử dụng chung một khóa giải mã (secret key), điều gì sẽ xảy ra nếu một ngân hàng "chơi xấu" và tiết lộ khóa đó cho tổ chức tài chính? Điều này sẽ khiến toàn bộ dữ liệu khách hàng từ các ngân hàng khác bị rò rỉ, vi phạm nghiêm trọng quyền riêng tư. Để giải quyết vấn đề này, nhóm em áp dụng thuật toán Multiparty CKKS, cụ thể là Threshold CKKS – trong đó mỗi bên (ngân hàng) giữ một khóa giải mã riêng. Chỉ khi tất cả các bên hợp tác, quá trình giải mã mới có thể diễn ra. Nhờ đó, hệ thống vẫn đảm bảo được cả tính bảo mật, tính riêng tư và khả năng tính toán hiệu quả.
- Lược đồ mã hóa của thuật toán:

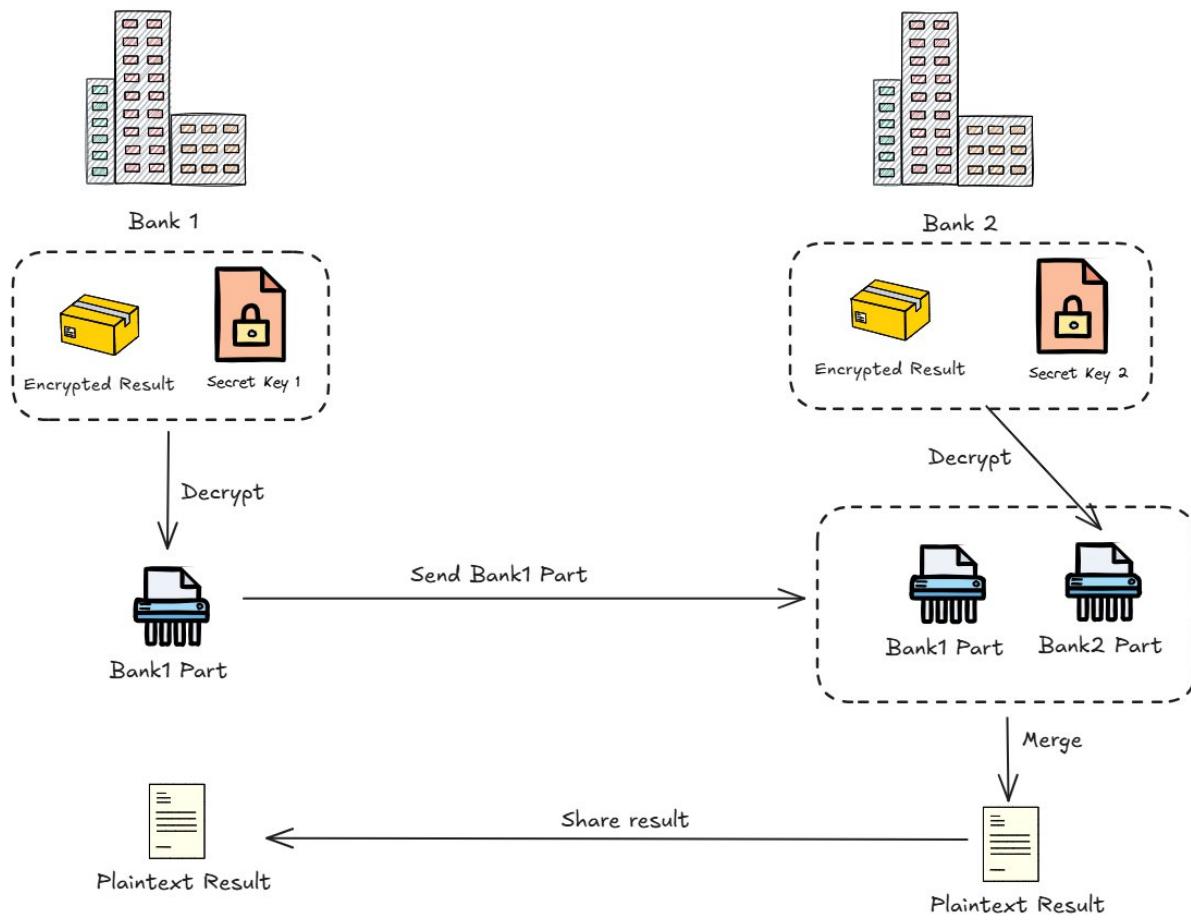
Multiparty CKKS Encrypt Flow (applied similarly across n banks)



Sơ đồ mã hóa của Threshold CKKS

- Lược đồ giải mã của thuật toán:

Multiparty CKKS Decrypt Flow (applied similarly across n banks)



Sơ đồ giải mã của Threshold CKKS

2. Elliptic Curve Digital Signature Algorithm (ECDSA)

- Là một thuật toán chữ ký số bất đối xứng (digital signature algorithm), sử dụng mật mã đường cong elliptic (ECC).
- Chức năng chính:
 - Ký số: đảm bảo tính xác thực và toàn vẹn của dữ liệu.
 - Xác minh chữ ký: kiểm tra dữ liệu có bị thay đổi hay không và có đúng người gửi hay không.
- Nhóm em chọn vì hiệu suất cao, độ an toàn mạnh mẽ, và kích thước khóa ngắn hơn so với RSAPSS.

3. Advanced Encryption Standard (AES)

- AES là một thuật toán mã hóa đối xứng, dùng để mã hóa và giải mã dữ liệu với cùng một khóa.
- Chức năng chính:
 - Mã hóa dữ liệu của cơ sở dữ liệu khi nằm trên ổ cứng.
 - Mã hóa dữ liệu khi truyền tải qua Internet.

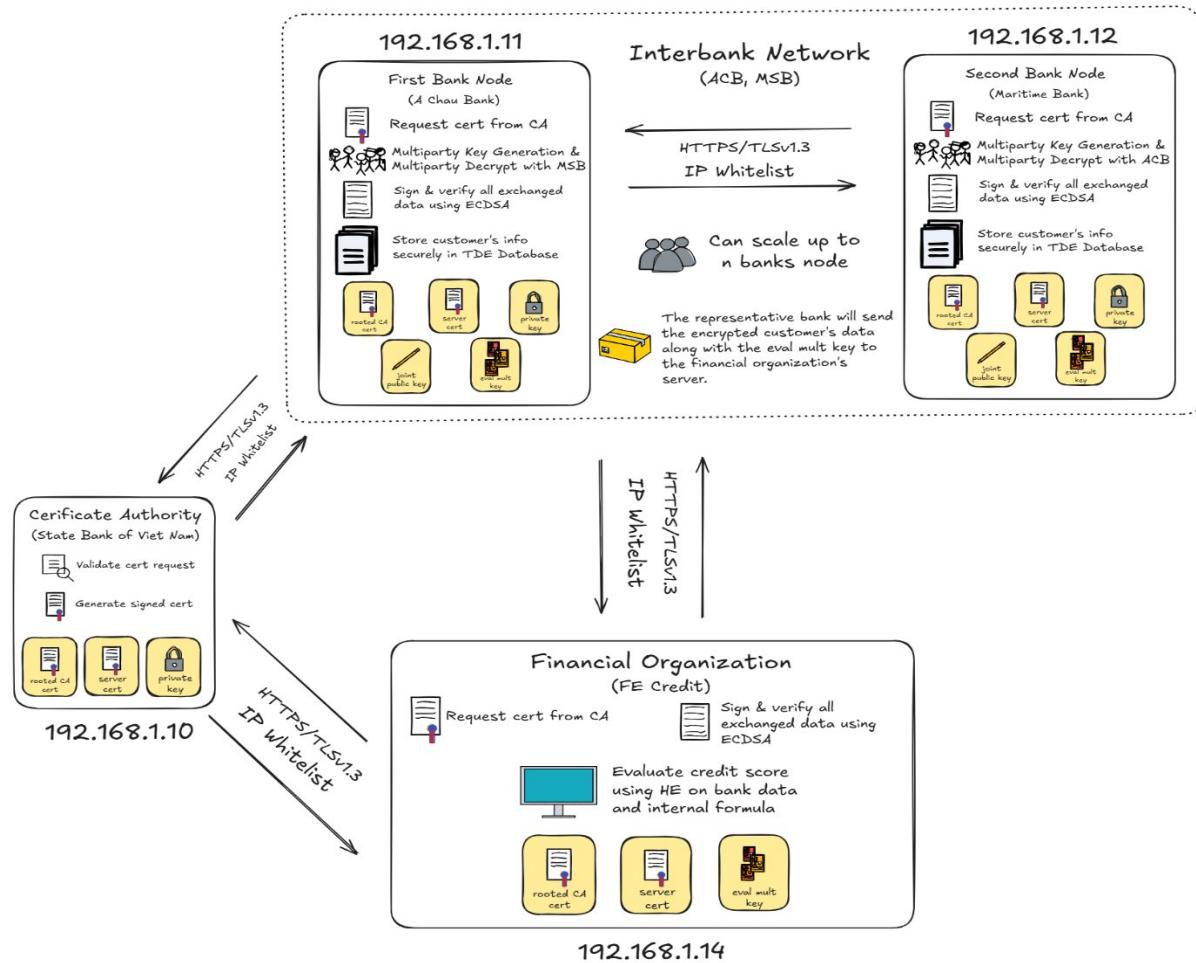
4. HyperText Transfer Protocol Secure (HTTPS)

- Là phiên bản bảo mật của HTTP – giao thức truyền tải dữ liệu trên Internet.
- HTTPS sử dụng TLS để mã hóa dữ liệu giữa máy khách và máy chủ, giúp:
 - Bảo mật: Dữ liệu không bị đọc trộm khi truyền.
 - Xác thực: Đảm bảo máy chủ là thật (dùng chứng chỉ số X.509).
 - Toàn vẹn: Ngăn dữ liệu bị thay đổi trên đường truyền.

5. Chứng chỉ số X.509

- Chứng chỉ số X.509 là một loại chứng chỉ điện tử dùng để xác thực danh tính của một thực thể (máy chủ, người dùng, dịch vụ) trong hệ thống bảo mật sử dụng hệ thống mã hóa khóa công khai (PKI).
- Chức năng trong hệ thống: Dùng để định danh các node cho HTTPS và chữ ký số.
- Nhóm em sử dụng ECC (P-256) và SHA-256/SHA-384 để tạo chứng chỉ số.

II. Kiến trúc hệ thống



Sơ đồ kiến trúc hệ thống

- Nhóm em xây dựng một hệ thống gồm 4 node:

1. Certificate Authority (Ngân hàng Nhà Nước)

- Chịu trách nhiệm cấp chứng chỉ cho các ngân hàng và tổ chức tài chính.
- Chỉ cho phép những IP trong danh sách kết nối tới.
- Đóng vai trò là gốc tin cậy (trust anchor) cho toàn bộ hệ thống.

2. Ngân hàng 1 (ACB)

- Bảo vệ dữ liệu của khách hàng với cơ sở dữ liệu mã hóa.
- Sử dụng chứng chỉ do Ngân hàng Nhà Nước cấp.
- Chỉ cho phép những IP trong danh sách kết nối tới.
- Luôn luôn ký và xác minh những dữ liệu trao đổi bằng ECDSA.
- Giao tiếp với Ngân hàng 2 để mã hóa và giải mã dữ liệu với Multiparty CKKS.

3. Ngân hàng 2 (MSB)

- Bảo vệ dữ liệu của khách hàng với cơ sở dữ liệu mã hóa.
 - Sử dụng chứng chỉ do Ngân hàng Nhà Nước cấp.
 - Chỉ cho phép những IP trong danh sách kết nối tới.
 - Luôn luôn ký và xác minh những dữ liệu trao đổi bằng ECDSA.
 - Giao tiếp với Ngân hàng 1 để mã hóa và giải mã dữ liệu với Multiparty CKKS.
 - Đại diện tổng hợp dữ liệu mã hóa và gửi tới tổ chức tài chính để thực hiện tính toán.
 - Đại diện nhận dữ liệu từ tổ chức tài chính.
- 4. Tổ chức tài chính (FE Credit)**
- Sử dụng chứng chỉ do Ngân hàng Nhà Nước cấp.
 - Luôn luôn ký và xác minh những dữ liệu trao đổi bằng ECDSA.
 - Chỉ cho phép những IP trong danh sách kết nối tới.
 - Sử dụng thuật toán đánh giá độc quyền để đánh giá điểm tín dụng trên dữ liệu mã hóa.
 - Trả về điểm tín dụng ở dạng mã hóa.

III. Thuật toán của tổ chức tài chính:

- PoC.py:

$$\text{FICO_Sim} = \frac{1}{A+1} \left[(S_{\text{payment}} \cdot W_1)^2 + \sqrt{S_{\text{util}} \cdot W_2 + 3 \cdot (S_{\text{behavioral}} \cdot W_7)^2} + \frac{(S_{\text{length}} \cdot W_3) + (S_{\text{credit_mix}} \cdot W_4)^2}{B+1} + \log(1 + S_{\text{inquiries}} \cdot W_5 + S_{\text{income_stability}} \cdot W_6) \right]$$

☞ **Trong đó:**

- $W_1 = 0.35$
- $W_2 = 0.30$
- $W_3 = 0.15$
- $W_4 = 0.10$
- $W_5 = 0.05$
- $W_6 = 0.03$
- $W_7 = 0.02$
- $A = S_{\text{util}} + S_{\text{inquiries}}^2$
- $B = \sqrt{S_{\text{credit_mix}} + S_{\text{income_stability}}} + 1$

Thành phần	Ký hiệu	Khoảng giá trị S	Giải thích ngắn
Lịch sử thanh toán	S_{payment}	0.0 – 1.0	1 = luôn trả đúng hạn
Dư nợ / Hạn mức	S_{util}	0.0 – 1.0	0 = sử dụng thấp, 1 = sử dụng toàn bộ
Tuổi tín dụng	S_{length}	0.0 – 1.0	Tính theo tuổi tín dụng tương đối
Loại tín dụng	$S_{\text{credit_mix}}$	0.0 – 1.0	1 = danh mục tín dụng đa dạng
Yêu cầu tín dụng	$S_{\text{inquiries}}$	0.0 – 1.0	1 = rất nhiều yêu cầu tín dụng gần đây
Ôn định thu nhập	$S_{\text{income_stability}}$	0.0 – 1.0	1 = thu nhập ổn định lâu dài
Hành vi tài chính	$S_{\text{behavioral}}$	0.0 – 1.0	1 = hành vi tài chính tốt

- Kết quả thực nghiệm, thể hiện tốc độ tính toán, mã hóa, giải mã và độ lệch chuẩn của CKKS so với tính toán trực tiếp trên 1000 lần chạy (File PoC_benchmark.py em để trong thư mục Testing):

```
== Benchmark Summary ==
Average Encryption Time: 0.779s
Average Computation Time: 6.674s
Average Decryption Time: 0.181s
Average Difference: 0.085263
Maximum Difference: 0.264889
```

- Công thức này có sai số hoàn toàn chấp nhận được (0,085.../850 điểm), phù hợp để sử dụng với CKKS.
- Tuy nhiên, khi triển khai thực tế, vì nhóm có một bạn rời giữa chừng nên bọn em chỉ có 2 người, đủ tài nguyên để triển khai 2 bên ngân hàng. Khi này, noise budget của CKKS không đủ để chịu tải công thức trên:

```
Merging all partial decryptions...
How many partial decryptions to merge?: 2
Path to partial decryption #1: ./InterbankService/Received/MSB_partialDecryption.txt
Path to partial decryption #2: Keys/ACB_partialDecryption.txt
Traceback (most recent call last):
  File "/home/acb/NT219-Project/Banks/HEModule/multipartyDecrypt.py", line 85, in <module>
    result_ptxt = cc.MultipartyDecryptFusion(part_decryptions)
RuntimeError: /root/openfhe-python-packager/build/openfhe-development/src/pke/lib/encoding/ckkspackedencoding.cpp:1.462:Decode(): The decryption failed because the approximation error is too high. Check the parameters.
(venv) acb@acb-VMware-Platform:~/NT219-Project/Banks/HEModule$
```

Nên bọn em buộc phải sử dụng công thức đơn giản hơn:

Công thức FICO_Sim đơn giản hóa:

$$\text{FICO_Sim} = S_1 \cdot W_1 + S_2 \cdot W_2 + S_3 \cdot W_3 + S_4 \cdot W_4 + S_5 \cdot W_5 + S_6 \cdot W_6 + S_7 \cdot W_7$$

Chương 3: Triển khai thực tế

I. Xây dựng hệ thống chung

- Nhóm em sử dụng 3 máy ảo và 1 máy thật, chạy Ubuntu 24.04. Tất cả được cài IP tĩnh, và kết nối vào một cùng một mạng:
 - Ngân hàng Nhà Nước (CA): 192.168.1.10
 - ACB: 192.168.1.11
 - MSB: 192.168.1.12
 - FE Credit: 192.168.1.14
- Vai trò của từng thành phần:
 - Ngân hàng nhà nước (CA): Nơi cấp chứng chỉ xác thực cho các bên.
 - ACB, MSB: Đại diện cho các ngân hàng, là chủ sở hữu dữ liệu để tính toán.
 - FE CREDIT: Nơi thực hiện các phép tính đã được mã hóa từ các bước mã hóa đồng cấu ở phía ngân hàng và trả về dữ liệu ở dạng mã hóa cho ngân hàng.
- Giao tiếp giữa các thành phần:
 - Protocol: TCP/IP được sử dụng để đảm bảo giao tiếp ổn định giữa các máy.
 - Bảo mật: Dữ liệu được truyền qua giao thức HTTPS, kèm theo mã hóa đồng cấu (Homomorphic Encryption).
 - Chứng thực: Các thành phần trong hệ thống sử dụng chữ ký số và chứng chỉ (Certificate) để đảm bảo tính toàn vẹn và xác thực của dữ liệu.

II. Các bước triển khai:

a. **Tạo khóa lần đầu cho các ngân hàng**

- Mở server để cấp chứng chỉ xác thực các bên.
- Đầu tiên, ngân hàng MSB sẽ xây dựng hai loại khóa: secret key và public key. Sau đó ngân hàng sẽ xin cấp chứng chỉ để xác thực, tiếp đến sẽ gửi khóa công khai lần đầu cho bên ACB để gộp khóa, khóa có tên là publicKey_MSB.
- Đồng thời FE Credit cũng phải tạo chứng chỉ cho riêng mình nhờ Ngân hàng Nhà Nước để xác thực đúng nơi tính toán
- Sau đó, Ngân hàng ACB sẽ lại gửi Joint Public Key cho bên MSB để tiếp tục bước tiếp theo.

b. **Tạo khóa nhân đánh giá**

- Sau khi hợp nhất khóa (Joint PublicKey) thì bên ACB đang sở hữu 2 tài nguyên đó là secret key của riêng họ và jointPublicKey, từ hai tài nguyên này ngân hàng sẽ tạo khóa nhân đánh giá và tạo ra evaltMultKey1, và gửi đến ngân hàng MSB
- Ngân hàng MSB nhận được file evaltMultKey1 lại thực hiện tạo khóa nhân của riêng họ dựa trên secret key của họ kết hợp với evaltMultKey1 từ ngân hàng ACB và tạo ra một khóa nhân khác của riêng nó, gọi là evalMultKey_raw, gửi lại cho ACB

c. Hợp nhất khóa đánh giá

- Ngân hàng ACB, MSB tạo từng mảnh của khóa nhân cuối cùng từ secret key của riêng họ, evalMultKey_raw và Joint Public Key
- ACB hợp nhất các khóa, tạo ra evalMultKey_merged.
- Giờ mỗi bên đều sở hữu Joint Public Key và evalMultKey_merged.

d. Mã hóa và tính toán

- Trường hợp mã hóa dữ liệu trên chỉ test trên cùng một người dùng với các tham số để thực hiện tính toán điểm tín dụng là duy nhất và các tham số không được trùng nhau (ví dụ ông A có tham số tuổi tín dụng ở ngân hàng A rồi thì không được có lại tham số tuổi tín dụng nữa).
- Sau khi hợp nhất khóa đánh giá và khóa công khai chung thì việc phân phối khóa sẽ diễn ra như sau:
 - o JointPublicKey: Được dùng để mã hóa dữ liệu từ phía ngân hàng
 - o evalMultKey_merged: Được gửi đến máy chủ (FE Credit) để thực hiện tính toán đồng cầu dựa trên dữ liệu đã mã hóa.

e. Gửi kết quả tính toán

- Máy chủ sử dụng các khóa để tính toán điểm tín dụng dựa trên dữ liệu được mã hóa và gửi lại kết quả cho từng ngân hàng để giải mã.

III. Triển khai chi tiết:

a. Certificate Authority (Ngân hàng Nhà Nước)

- File server.py + runServer.sh: File này triển khai một server CA (Certificate Authority) đơn giản sử dụng FastAPI, cho phép các tổ chức gửi yêu cầu cấp phát chứng chỉ số (CSR) và nhận lại chứng chỉ đã được ký bởi CA.
 - Middleware kiểm tra IP
 - + Mỗi request đến server sẽ được kiểm tra IP.
 - + Nếu IP không nằm trong danh sách cho phép, trả về lỗi 403 Forbidden.
 - API /submit-csr (POST)
 - + Nhận 2 file upload:

- csr: File Certificate Signing Request (CSR).
 - config: File cấu hình (dùng để ký chứng chỉ với các extension như SAN).
- + Quy trình:
- Lưu tạm 2 file này vào hệ thống.
 - Tạo file chứng chỉ tạm.
 - Gọi lệnh OpenSSL để ký CSR thành chứng chỉ số, sử dụng file cấu hình để thêm các extension cần thiết.
 - Trả về file chứng chỉ đã ký cho client.
 - Nếu có lỗi khi chạy OpenSSL hoặc lỗi khác, trả về mã lỗi 500 cùng thông báo chi tiết.
 - Dọn dẹp file tạm sau khi xử lý xong.

b. Các ngân hàng (Banks): 4 module

1. Thư mục Certificate

- File requestCert.py: Xin cấp chứng chỉ từ Server
 - Đầu tiên, tạo một file config tạm cho chứng chỉ, gồm: Quốc gia, Tỉnh/Thành phố, tên tổ chức và tên thường gọi. Ngoài ra, kèm theo SAN là địa chỉ IP (rất quan trọng, để sử dụng cho HTTPS vì máy khách sẽ chỉ tin khi server dùng IP này).

```
# === 1. Tạo file config tạm có SAN ===
with tempfile.NamedTemporaryFile("w", suffix=".cnf", delete=False) as config_file:
    config_path = config_file.name
    config_file.write(f"""
[req]
default_bits      = 2048
distinguished_name = req_distinguished_name
req_extensions    = req_ext
prompt            = no

[req_distinguished_name]
C = {country}
ST = {state}
O = {org}
CN = {commonname}

[req_ext]
subjectAltName = IP:{public_ip}
""")
```

- Tạo secret key và CSR

```
# === 3. Tạo EC Private Key ===
subprocess.run([
    "openssl", "ecparam", "-name", "prime256v1",
    "-genkey", "-noout", "-out", key_path
], check=True)

# === 4. Tạo CSR từ private key + config ===
subprocess.run([
    "openssl", "req", "-new", "-key", key_path,
    "-out", csr_path,
    "-config", config_path
], check=True)
```

- Gửi CSR qua HTTPS

```
# === 5. Gửi CSR qua HTTPS ===
with open(csr_path, "rb") as f:
    csr_data = f.read()
SERVER_URL = f"https://{{server_domain}}:443/submit-csr"
response = requests.post(
    SERVER_URL,
    files = {
        "csr": (f"{{commonname}}.csr", open(csr_path, "rb"), "application/pkcs10"),
        "config": (f"{{commonname}}.cnf", open(config_path, "rb"), "text/plain")
    },
    verify="./RootCA.crt",
    timeout=(10, 300)
)
```

- Nhận chứng chỉ, lưu chứng chỉ và secret key vào file

```

# === 6. Nhận và lưu cert nếu thành công ===
if response.status_code == 200:
    client_key_path = f"{commonname}.key"
    client_crt_path = f"{commonname}.crt"

    with open(client_key_path, "wb") as f:
        f.write(open(key_path, "rb").read())

    with open(client_crt_path, "wb") as f:
        f.write(response.content)

    print(f"Certificate request complete!")
    print(f"Private key: {client_key_path}")
    print(f"Certificate: {client_crt_path}")
else:
    print(f"Certificate request failed: {response.status_code}")
    print(response.text)

# === 7. Cleanup ===
os.remove(csr_path)
os.remove(config_path)

```

2. Database Module

- Setup PostgreSQL TDE:

- Vấn đề: Nếu lưu trữ cơ sở dữ liệu ở dạng bình thường, nếu hệ thống của ngân hàng bị tấn công, kẻ xấu có thể đọc dữ liệu khách hàng trái phép => Cần phải có một biện pháp mã hóa dữ liệu khi lưu trữ trên ổ đĩa, và chỉ có thể được đọc bởi người có thẩm quyền.
- Giải pháp: Áp dụng mã hóa dữ liệu ở tầng lưu trữ (Transparent Data Encryption – TDE) giúp bảo vệ toàn bộ nội dung cơ sở dữ liệu ngay trên ổ đĩa. Với TDE trong PostgreSQL, dữ liệu được mã hóa tự động bằng AES khi ghi xuống và giải mã khi được truy xuất bởi tiến trình hợp lệ. Khóa giải mã không lưu trực tiếp trong cấu hình mà được cung cấp thông qua một script bảo mật (như providekey.sh), chỉ cho phép hệ thống có quyền hợp lệ khởi động được cơ sở dữ liệu. Nhờ đó, ngay cả khi kẻ tấn công truy cập vào ổ cứng hoặc sao chép file dữ liệu vật lý, nội dung cũng sẽ ở dạng mã hóa – không thể đọc được nếu không có khóa chính xác. Đây là lớp phòng vệ quan trọng để bảo vệ thông tin khách hàng trong các hệ thống tài chính như ngân hàng.
- Triển khai: Em đã note chi tiết trong file kèm theo. Đây là ý chính:

- + Tạo một chuỗi hex random làm khóa
- + Setup file khóa chỉ có duy nhất user root có thể đọc được. User postgres chỉ có thể thực thi, không thể đọc!

```

nano providekey.sh
sudo chmod +x /usr/local/bin/providekey.sh
sudo chown root:root /usr/local/bin/providekey.sh
sudo chmod 500 /usr/local/bin/providekey.sh
sudo adduser --disabled-password --gecos "" postgres
sudo visudo
postgres ALL=(ALL) NOPASSWD: /usr/local/bin/providekey.sh
sudo mkdir -p /usr/local/postgres
sudo chown postgres:postgres /usr/local/postgres
sudo chmod 775 /usr/local/postgres

```

- + Cài mật khẩu cho cơ sở dữ liệu. Phải có mật khẩu mới truy cập được vào cơ sở dữ liệu.

- Kết quả: Em thử đọc file cứng trên ổ đĩa của bảng customer:

```

postgres@acb-VMware-Virtual-Platform:~$ psql -d acb_db
Password for user postgres:
psql (12.3_TDE_1.0)
Type "help" for help.

acb_db=# SELECT oid, relname FROM pg_class WHERE relname = 'customer';
      oid      | relname
-----+-----
      16422    | customer
(1 row)

acb@acb-VMware-Virtual-Platform:~$ sudo su
root@acb-VMware-Virtual-Platform:/home/acb# cd /usr/local/postgres/base
root@acb-VMware-Virtual-Platform:/usr/local/postgres/base# cd /usr/local/postgres/base
root@acb-VMware-Virtual-Platform:/usr/local/postgres/base# sudo find . -type f -name '16422'
./16384/16422
root@acb-VMware-Virtual-Platform:/usr/local/postgres/base# nano ./16384/16422

```

```

localain.tfoi          root@acb-VMware-Virtual-Platform:/usr/local/postgres/base . /16384/16422

```

⇒ Không thể đọc được!

- File `getCustomerInfo.py`: Truy vấn và lấy thông tin điểm tín dụng của khách hàng từ cơ sở dữ liệu PostgreSQL đã được mã hóa bằng TDE (Transparent Data Encryption). Sử dụng tên khách hàng để thực hiện truy vấn, yêu cầu người dùng nhập mật khẩu

```

passw = getpass("Get PostgreSQL password: ")

```

```

def get_credit_scores_by_name():
    try:
        conn = psycopg2.connect(**conn_info)
        cursor = conn.cursor()
        name = input("Input customer name: ").strip()
        # Truy vấn Lấy CustomerID
        cursor.execute("SELECT CustomerID FROM Customer WHERE Name = %s", (name,))
        result = cursor.fetchone()

        if not result:
            print("Không tìm thấy khách hàng:", name)
            return

        customer_id = result[0]

        # Truy vấn bảng Data
        cursor.execute("""
            SELECT Spayment, Sutil, Slength, Screditmix,
            Sinquiries, Sincomestability, Sbehaviorial
            FROM Data
            WHERE CustomerID = %s
        """, (customer_id,))
        data = cursor.fetchone()

        if data:
            print(f"Chi số tín dụng của '{name}':")
            fields = ["Spayment", "Sutil", "Slength", "Screditmix",
                      "Sinquiries", "Sincomestability", "Sbehaviorial"]
            for field, value in zip(fields, data):
                print(f" {field}: {value}")
        else:
            print(f"Không có dữ liệu tín dụng cho khách hàng '{name}'")

        cursor.close()

```

3. Homomorphic Encryption (HE) Module: Sử dụng thư viện openfhe-python

- CryptoContext dùng chung cho tất cả các file (kể cả server)

```

# Initialize CKKS parameters
# CKKS is a scheme that supports approximate arithmetic on encrypted real numbers
parameters = fhe.CCParamsCKKSRSN()
# Set the maximum depth of multiplication operations allowed
parameters.SetMultiplicativeDepth(15)
# Set the scaling factor size for CKKS encoding
parameters.SetScalingModsize(59)
# Set the number of slots for batch processing
parameters.SetBatchSize(1)

# Create crypto context with the specified parameters
# The crypto context manages all cryptographic operations
crypto_context = fhe.GenCryptoContext(parameters)

# Enable required features for the crypto context
# PKE: Public Key Encryption - enables basic encryption/decryption
crypto_context.Enable(fhe.PKESchemeFeature.PKE)
# LEVELED SHE: Leveled Homomorphic Encryption - enables operations with limited depth
crypto_context.Enable(fhe.PKESchemeFeature.LEVELED SHE)
# ADVANCED SHE: Advanced Homomorphic Encryption - enables more complex operations
crypto_context.Enable(fhe.PKESchemeFeature.ADVANCEDSHE)
crypto_context.Enable(fhe.PKESchemeFeature.MULTIPARTY)

```

- File keyGenerator.py: Tạo và xuất cặp khóa mã hóa đồng hình (bình thường) cho từng ngân hàng

```

# Generate the key pair (public and private keys)
keys = crypto_context.KeyGen()

# Serialize the public key to a file
# The public key is used for encryption and can be shared publicly
if not fhe.SerializeToFile(f'{key_dir}/{bank_name}_publicKey.txt', keys.publicKey, fhe.BINARY):
    raise Exception("Error writing serialization of the public key")
print("The public key has been serialized.")

# Serialize the private key to a file
# The private key is used for decryption and must be kept secure
if not fhe.SerializeToFile(f'{key_dir}/{bank_name}_privateKey.txt', keys.secretKey, fhe.BINARY):
    raise Exception("Error writing serialization of the private key")
print("The private key has been serialized.")

# Print summary of generated files
print("\nKeys have been generated and exported:")
print(f"- {key_dir}/{bank_name}_publicKey.txt")      # Used for encryption
print(f"- {key_dir}/{bank_name}_privateKey.txt")      # Used for decryption

```

- File calculateJointKey.py: Dựa vào khóa công khai đầu vào, tính ra khóa riêng tư cho chủ thẻ và đóng góp vào khóa công khai chung

```

# 2. Deserialize publicKey của bên trước
print(f"Loading latest public key from: {prev_file}")
publicKey, result = fhe.DeserializePublicKey(prev_file, fhe.BINARY)
if not result:
    raise Exception("Cannot deserialize chosen public key.")

# 3. Tạo cặp khóa mới dựa trên publicKey trước đó
print("Generating our contribution to joint public key...")
keyPair = cc.MultipartyKeyGen(publicKey)

# 4. Lưu lại public key và secret key mới
# Cần phải bao gồm hợp đồng?
is_aggregator = input("Are you the aggregator party? (y/n): ").strip().lower()
if is_aggregator == 'y':
    pub_path = os.path.join(key_dir, f"{bank_name}_publicKey.txt")
else:
    pub_path = os.path.join(key_dir, f"jointPublicKey.txt")
priv_path = os.path.join(key_dir, f"{bank_name}_privateKey.txt")

# Serialize public key
pub_data = fhe.Serialize(keyPair.publicKey, fhe.BINARY)
if not pub_data:
    raise Exception("Cannot serialize public key.")
save_file(pub_path, pub_data)

# Serialize private key
priv_data = fhe.Serialize(keyPair.secretKey, fhe.BINARY)
if not priv_data:
    raise Exception("Cannot serialize private key.")
save_file(priv_path, priv_data)

print("\nKey pair generated and saved successfully!")
print(f"Public Key: {pub_path}")
print(f"Private Key: {priv_path}")

```

- File evalMultKey1: Triển khai giai đoạn 1, tích lũy tiền (Forward Accumulation) của quá trình tạo khóa nhân chung

- Đầu vào: Khóa riêng tư (secret key) của chủ thẻ, khóa nhân tích lũy trước đó (nếu có)
- Đầu ra: Một khóa nhân tích lũy, được đóng góp phần tích lũy của chủ thẻ

```

# Tải khóa riêng tư
print(f"Loading your private key from: {prv_key_file}")
privateKey, result = fhe.DeserializePrivateKey(prv_key_file, fhe.BINARY)
if not result:
    raise Exception("Cannot deserialize private key.")

# Xác định xem có phải bên khởi tạo đầu tiên không
is_starter = input("Are you the first party? (y/n): ").strip().lower()

if is_starter == 'y':
    # Nếu là bên đầu tiên, tạo EvalMultKey ban đầu
    print("Generating initial EvalMultKey...")
    evalMulKey = cc.KeySwitchGen(privateKey, privateKey)
else:
    # Nếu không phải bên đầu tiên, tải EvalMultKey trước đó và thêm phần đóng góp
    eval_key_file = input("Input path to previous EvalMultKey: ").strip()
    if not os.path.exists(eval_key_file):
        raise Exception(f"File '{eval_key_file}' does not exist.")

    # Tải và kiểm tra EvalMultKey trước đó
    with open(eval_key_file, 'rb') as f:
        eval_key_str = f.read()
    prev_eval_key = fhe.DeserializeEvalKeyString(eval_key_str, fhe.BINARY)
    if not isinstance(prev_eval_key, fhe.EvalKey):
        raise Exception("Invalid EvalKey type.")

    # Tạo phần khóa mới và tích lũy
    print("Generating EvalMultKey contribution...")
    newKeyPart = cc.MultiKeySwitchGen(privateKey, privateKey, prev_eval_key)

    # Kết hợp các phần khóa
    print("Merging EvalMultKey parts...")
    evalMulKey = cc.MultiAddEvalKeys(prev_eval_key, newKeyPart, privateKey.GetKeyTag())

    # Lưu EvalMultKey vào file
    print("Serializing EvalMultKey...")
    eval_key_str = fhe.SerializeEvalMultKey(fhe.BINARY)

```

- File evalMultKey2.py: Triển khai giai đoạn 2, hoàn thiện ngược (Backward Finalization) của quá trình tạo khóa nhân chung
 - Đầu vào: Khóa riêng tư (secret key) của chủ thẻ, khóa nhân tích lũy của giai đoạn 1, khóa công khai chung (joint public key)
 - Đầu ra:
 - + Một mảnh ghép của khóa nhân chung
 - + Nếu chủ thẻ là bên cuối, có trách nhiệm tổng hợp các mảnh khóa để tạo ra khóa nhân chung cuối cùng

```

# Tải EvalMultKey đã tích lũy
print(f"Loading EvalMultKey from: {eval_key_file}")
with open(eval_key_file, 'rb') as f:
    eval_key_bytes = f.read()
eval_key = fhe.DeserializeEvalKeyString(eval_key_bytes, fhe.BINARY)
if not isinstance(eval_key, fhe.EvalKey):
    raise Exception("Invalid EvalKey type.")

# Tải khóa công khai chung
print(f"Loading joint public key from: {joint_pub_key_file}")
publicKey, result = fhe.DeserializePublicKey(joint_pub_key_file, fhe.BINARY)
if not result:
    raise Exception("Cannot deserialize joint public key.")

# Tải khóa riêng tư cá nhân
print(f"Loading private key from: {prv_key_file}")
privateKey, result = fhe.DeserializePrivateKey(prv_key_file, fhe.BINARY)
if not result:
    raise Exception("Cannot deserialize private key.")

# Tạo phần đóng góp EvalMultKey cuối cùng (ngược)
print("Generating backward EvalMultKey contribution...")
finalKeyPart = cc.MultiMultEvalKey(privateKey, eval_key, publicKey.GetKeyTag())

# Lưu phần đóng góp vào file
eval_path = os.path.join(key_dir, "evalMultKey_final.txt")
print("Serializing your EvalMultKey contribution...")
eval_key_bytes = fhe.Serialize(finalKeyPart, fhe.BINARY)
save_file(eval_path, eval_key_bytes)

print(f"Final EvalMultKey contribution saved to: {eval_path}")

# Xác định xem có phải bên tổng hợp không
is_aggregator = input("Are you the aggregator party? (y/n): ").strip().lower()
if is_aggregator == 'y':
    print("Now merging all final EvalMultKey parts...")

# Nhập số lượng phần khóa cần gộp
num_parts = int(input("How many final EvalMultKey parts to merge?: "))
final_keys = []

# Tải từng phần khóa
for i in range(num_parts):
    path = input(f"Path to final EvalMultKey part #{i + 1}: ").strip()
    if not os.path.exists(path):
        raise Exception(f"File '{path}' does not exist.")

    with open(path, 'rb') as f:
        part_bytes = f.read()
    key_part = fhe.DeserializeEvalKeyString(part_bytes, fhe.BINARY)
    final_keys.append(key_part)

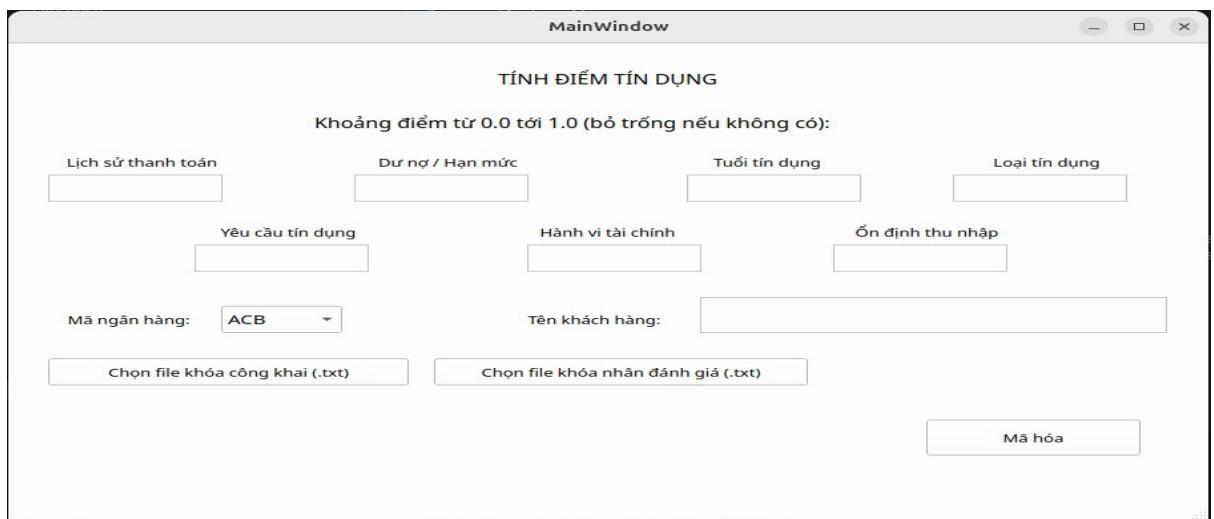
# Gộp tần tự các phần khóa
merged_key = final_keys[0]
for i in range(1, num_parts):
    merged_key = cc.MultiAddEvalMultKeys(merged_key, final_keys[i], merged_key.GetKeyTag())

# Lưu khóa đã gộp
merged_path = os.path.join(key_dir, "evalMultKey_merged.txt")
print("Serializing merged EvalMultKey...")
merged_bytes = fhe.Serialize(merged_key, fhe.BINARY)
save_file(merged_path, merged_bytes)

print(f"Final merged EvalMultKey saved to: {merged_path}")

```

- File interactiveEncrypt.py + MainWindow.ui: Mã hóa trực quan, dùng PyQt6 để tạo GUI.
 - Đầu vào: Các thông tin khách hàng mà ngân hàng sở hữu, khóa công khai chung (Joint Public Key), khóa nhân chung (Eval Mult Key)
 - Đầu ra: Từng thông tin được mã hóa, lưu ở dạng file



```

# Get values from text fields and only include non-empty ones
user_data = {}
fields = {
    's_payment': self.s_payment.toPlainText(),
    's_util': self.s_util.toPlainText(),
    's_length': self.s_length.toPlainText(),
    's_creditmix': self.s_creditmix.toPlainText(),
    's_inquiries': self.s_inquiries.toPlainText(),
    's_behavioral': self.s_behavioral.toPlainText(),
    's_incomestability': self.s_incomestability.toPlainText()
}

for key, value in fields.items():
    if value.strip(): # Only include non-empty values
        try:
            user_data[key] = [float(value)]
        except ValueError:
            QMessageBox.warning(self, "Cảnh báo", f"Giá trị không hợp lệ cho {key}: {value}")
            continue

if not user_data:
    QMessageBox.warning(self, "Cảnh báo", "Không có dữ liệu nào để mã hóa")
    return

customer_name = self.customerName.toPlainText()
bank_name = self.selectBank.currentText()

# Encrypt and serialize data
self.loading = True
try:
    # Lưu metadata
    with open(f'metadata_{bank_name}.txt', 'w') as f:
        f.write(f"Bank: {bank_name}\n")
        f.write(f"Customer: {customer_name}\n")
        f.write(f"Length: {s_length}\n")
        f.write(f"Util: {s_util}\n")
        f.write(f"Creditmix: {s_creditmix}\n")
        f.write(f"Inquiries: {s_inquiries}\n")
        f.write(f"Behavioral: {s_behavioral}\n")
        f.write(f"Incomestability: {s_incomestability}\n")

```

```

bank_name = self.SELECTBANKREVERTEXX()

# Encrypt and serialize data
self.loading = True
try:
    # Lưu metadata
    with open(f'metadata_{bank_name}.txt', 'w') as f:
        f.write(f"Bank: {bank_name}\n")
        f.write(f"Customer: {customer_name}\n")

    # Mã hóa và lưu từng tham số
    for k, v in user_data.items():
        # Mã hóa dữ liệu
        ciphertext = self.encrypt_data(v)
        serialized = self.serialize_ciphertext(ciphertext)
        if serialized:
            # Lưu vào file riêng cho từng tham số
            with open(f'ciphertext_{bank_name}_{k}.txt', 'wb') as f:
                f.write(serialized)

    QMessageBox.information(self, "Kết quả", f"Đã mã hóa và lưu dữ liệu thành công!")
except Exception as e:
    QMessageBox.critical(self, "Lỗi", f"Không thể lưu file: {str(e)}")
finally:
    self.loading = False
    self.close()

except Exception as e:
    QMessageBox.critical(self, "Lỗi", f"Có lỗi xảy ra: {str(e)}")

```

- File multipartyDecrypt.py: Thực hiện giải mã kết quả tính toán liên ngân hàng bằng phương pháp multiparty decryption – mỗi ngân hàng giải mã một phần, sau đó ghép lại để lấy kết quả cuối cùng.
 - Đầu vào: File Ciphertext, khóa riêng tư của chủ thẻ
 - Hỏi vai trò: ngân hàng này là "lead" hay "main" trong quá trình giải mã?
 - + Nếu là "lead": dùng MultipartyDecryptLead
 - + Nếu là "main": dùng MultipartyDecryptMain
 - Đầu ra: Một mảnh giải mã của chủ thẻ (Bản chất vẫn là Ciphertext)

```

# Tải private key
privateKey, result = fhe.DeserializePrivateKey(prv_key_file, fhe.BINARY)
if not result:
    raise Exception("Cannot deserialize private key.")

# === Giải mã kết quả mã hóa liên ngân hàng ===
encrypted_file = input("Path to current encrypted result file: ").strip()
if not os.path.exists(encrypted_file):
    raise Exception(f"Encrypted file '{encrypted_file}' does not exist.")
with open(encrypted_file, 'rb') as f:
    ct_bytes = f.read()
encrypted_result = fhe.DeserializeCiphertextString(ct_bytes, fhe.BINARY)
if not isinstance(encrypted_result, fhe.Ciphertext):
    raise Exception(
        "Error reading serialization of the joint ciphertext"
    )
print("The joint ciphertext has been serialized.")

# Hỏi người dùng Là Lead hay Main
role = input("Are you the 'lead' bank for decryption? (y/n): ").strip().lower()
if role == 'y':
    part_decrypt = cc.MultipartyDecryptLead([encrypted_result], privateKey)[0]
else:
    part_decrypt = cc.MultipartyDecryptMain([encrypted_result], privateKey)[0]

# Lưu phần giải mã cục bộ của bạn
part_dec_path = os.path.join(key_dir, f"{bank_name}_partialDecryption.txt")
with open(part_dec_path, 'wb') as f:
    f.write(fhe.Serialize(part_decrypt, fhe.BINARY))
print(f"Your partial decryption saved to: {part_dec_path}")

```

- Bên cuối cùng có trách nhiệm kết hợp tất cả Ciphertext để lấy ra được Plaintext

```

# Hỏi người dùng có phải bên tập hợp kết quả không
is_aggregator = input("Are you the aggregator bank? (y/n): ").strip().lower()
if is_aggregator == 'y':
    print("Merging all partial decryptions...")
    num_parts = int(input("How many partial decryptions to merge?: "))
    part_decrypts = []

    for i in range(num_parts):
        path = input(f"Path to partial decryption #{i + 1}: ").strip()
        if not os.path.exists(path):
            raise Exception(f"File '{path}' does not exist.")
        part_ct, _ = fhe.DeserializeCiphertext(path, fhe.BINARY)
        part_decrypts.append(part_ct)

    # Ghép các phần giải mã lại
    result_ptxt = cc.MultipartyDecryptFusion(part_decrypts)
    result_ptxt.SetLength(1) # chỉ giải mã một giá trị duy nhất
    # Trả về kết quả thang 300 - 850
    raw_score = result_ptxt.GetRealPackedValue()[0]
    credit_score = 300 + (raw_score * 550)
    print("\n==== Final Decryption Result ===")
    print("Credit score:", credit_score)

```

4. Interbank Module: Mạng lưới giao tiếp giữa các ngân hàng và tổ chức tài chính

- InterbankAPI.py + runAPI.sh: Là API server (FastAPI) nhận dữ liệu liên ngân hàng (file mã hóa, metadata, chữ ký số, certificate) từ các ngân hàng khác.
 - Chỉ cho phép các IP ngân hàng trong whitelist truy cập API:

```
# Danh sách IP cho phép: MSB, ACB, FECREDIT
ALLOWED_IPS = {"192.168.1.11", "192.168.1.12", "192.168.1.14"}
```

```
@app.middleware("http")
async def verify_client_ip(request, call_next):
    client_ip = request.client.host
    if client_ip not in ALLOWED_IPS:
        raise HTTPException(status_code=403, detail="Forbidden: IP not allowed")
    response = await call_next(request)
    return response
```

- Kiểm tra certificate gửi lên có được ký bởi RootCA không:

```
try:
    cert_pem = await certificate.read()
    cert = x509.load_pem_x509_certificate(cert_pem)
    public_key = cert.public_key()

    if not isinstance(public_key, ec.EllipticCurvePublicKey):
        raise HTTPException(status_code=400, detail="Certificate must use EC key.")

    with open(CUSTOM_CA_PATH, "rb") as f:
        root_cert_pem = f.read()
        root_cert = x509.load_pem_x509_certificate(root_cert_pem)

    if not verify_certificate_signed_by_root(cert, root_cert):
        raise HTTPException(status_code=403, detail="Certificate not signed by trusted RootCA.")
except Exception as e:
    raise HTTPException(status_code=400, detail=f"Certificate error: {e}")
```

- Kiểm tra chữ ký số trên dữ liệu file + metadata:

```
try:
    data_to_verify = file_bytes + json.dumps(metadata_dict, sort_keys=True).encode("utf-8")
    decoded_sig = base64.b64decode(signature)

    public_key.verify(
        decoded_sig,
        data_to_verify,
        ec.ECDSA(hashes.SHA256())
    )
except InvalidSignature:
    raise HTTPException(status_code=403, detail="Invalid signature.")
except Exception as e:
    raise HTTPException(status_code=400, detail=f"Error verifying signature: {e}")
```

- Nếu hợp lệ, lưu file và metadata vào thư mục Received/, đồng thời trả về response cho client:

```

try:
    filename_base = f"{file.filename}"
    file_path = UPLOAD_DIR / filename_base
    metadata_path = file_path.with_suffix(".json")

    file_path.write_bytes(file_bytes)
    metadata_path.write_text(json.dumps(metadata_dict, indent=2))

    return JSONResponse(status_code=200, content={
        "message": "File received and verified.",
        "file_path": str(file_path),
        "metadata_path": str(metadata_path),
    })
except Exception as e:
    raise HTTPException(status_code=500, detail=f"Error saving file: {e}")

```

- InterbankClient.py: Là client gửi dữ liệu liên ngân hàng (file mã hóa, metadata, chữ ký số, certificate) sang API của ngân hàng khác.
 - Đọc file cần gửi và metadata; Đọc private key để ký số lên dữ liệu (file + metadata):

```

# === INPUT FILE ===
file_path = input("Input file path: ").strip()
file_path = Path(file_path)
if not file_path.exists():
    print("File not exist.")
    exit(1)

# === OPTIONAL METADATA ===
metadata_input = input("Input Metadata (JSON): ").strip()
try:
    metadata = json.loads(metadata_input) if metadata_input else {}
except json.JSONDecodeError:
    print("Metadata not valid.")
    exit(1)

# === LOAD EC PRIVATE KEY ===
key_path = f"../Certificate/{BANK_CODE}.key"
try:
    with open(key_path, "rb") as f:
        private_key = serialization.load_pem_private_key(
            f.read(),
            password=None,
        )
    if not isinstance(private_key, ec.EllipticCurvePrivateKey):
        raise TypeError("Wrong EC Key")
except Exception as e:
    print(f"Loi khi tai private key: {e}")
    exit(1)

```

- Gửi request HTTPS (POST) đến API /upload của ngân hàng đích, kèm chứng chỉ X509 của chủ thẻ:

```

# === LOAD X.509 CERT ===
cert_path = f"../Certificate/{BANK_CODE}.crt"
try:
    with open(cert_path, "rb") as f:
        cert_pem = f.read()
        cert_obj = x509.load_pem_x509_certificate(cert_pem)
except Exception as e:
    print(f"Lỗi khi đọc certificate: {e}")
    exit(1)

# === GỬI REQUEST ===
files = {
    "file": (file_path.name, file_bytes),
    "certificate": ("cert.pem", cert_pem),
}
data = {
    "metadata": json.dumps(metadata),
    "signature": signature_b64
}

try:
    print("Send request...")
    response = requests.post(SERVER_URL, data=data, files=files, verify="./RootCA.crt", timeout=(10, 300))
    print(f"✓ Server response({response.status_code}): \n{response.text}")
except Exception as e:
    print(f"Lỗi khi gửi HTTPS request: {e}")

```

- sendToFECredit.py: Gửi toàn bộ dữ liệu (các file mã hóa, evalMultKey, metadata, chữ ký số, certificate) sang tổ chức tài chính (FE Credit) để tính điểm tín dụng.
 - Đọc private key để ký số lên tông hợp dữ liệu (nội dung các file + metadata); Đọc certificate để gửi kèm.

```

# === OPTIONAL METADATA ===
metadata_input = input("\nEnter Metadata (JSON): ").strip()
try:
    metadata = json.loads(metadata_input) if metadata_input else {}
except json.JSONDecodeError:
    print("Lỗi: Metadata không phải là JSON hợp lệ.")
    exit(1)

# === LOAD EC PRIVATE KEY CỦA BÊN GỬI ===
key_path = f"../Certificate/{bank_code_sender}.key"
try:
    with open(key_path, "rb") as f:
        private_key = serialization.load_pem_private_key(
            f.read(),
            password=None, # Thêm password nếu key có mã hóa
        )
    if not isinstance(private_key, ec.EllipticCurvePrivateKey):
        raise TypeError("Key không phải là Elliptic Curve Private Key.")
except Exception as e:
    print(f"Lỗi khi tải private key từ '{key_path}': {e}")
    exit(1)

```

```

# === TẠO CHỮ KÝ SỐ ===
try:
    # 1. Đọc nội dung tất cả các file vào bộ nhớ
    file_contents = {}
    for key, path in input_files.items():
        with open(path, "rb") as f:
            file_contents[key] = f.read()

    # 2. Tạo dữ liệu để ký
    # Rất quan trọng: Nối nội dung các file theo thứ tự key đã được sắp xếp
    # để đảm bảo bên nhận có thể tái tạo lại đúng thứ tự để xác minh.
    data_to_sign = b''
    for key in sorted(file_contents.keys()):
        data_to_sign += file_contents[key]

    # 3. Nối metadata đã được chuẩn hóa vào cuối
    data_to_sign += json.dumps(metadata, sort_keys=True).encode('utf-8')

    # 4. Ký lên dữ liệu tổng hợp bằng private key
    signature = private_key.sign(
        data_to_sign,
        ec.ECDSA(hashes.SHA256())
    )
    signature_b64 = base64.b64encode(signature).decode('utf-8')
    print("\nCreate digital signature successful.")

except Exception as e:
    print(f"Loi khi tao chua ky so: {e}")
    exit(1)

```

```

# === LOAD EC PRIVATE KEY CỦA BÊN GỬI ===
key_path = f"../Certificate/{bank_code_sender}.key"
try:
    with open(key_path, "rb") as f:
        private_key = serialization.load_pem_private_key(
            f.read(),
            password=None, # Thêm password nếu key có mã hóa
        )
    if not isinstance(private_key, ec.EllipticCurvePrivateKey):
        raise TypeError("Key không phải là Elliptic Curve Private Key.")
except Exception as e:
    print(f"Loi khi tai private key tu '{key_path}': {e}")
    exit(1)

```

- Gửi request HTTPS (POST) đến endpoint /calculate-credit-score của FE Credit.

```

files_to_send = {
    # Thêm certificate vào danh sách file gửi đi
    "certificate": (f"{bank_code_sender}.crt", cert_pem_bytes, 'application/x-x509-ca-cert'),
}
for key, content in file_contents.items():
    # Sử dụng tên file gốc làm tên trong request
    original_filename = input_files[key].name
    files_to_send[key] = (original_filename, content, 'application/octet-stream')

# Chuẩn bị `data` dictionary cho requests (form data)
data_to_send = {
    "metadata": json.dumps(metadata),
    "signature": signature_b64
}

try:
    print(f"Sending request...")
    response = requests.post(SERVER_URL, data=data_to_send, files=files_to_send, verify="./RootCA.crt", timeout=10)
    print(f"Server response with status code: {response.status_code}")

```

- Nhận response dạng multipart (kết quả mã hóa, chữ ký số, certificate của server), xác thực certificate và chữ ký số của server trước khi lưu kết quả.

```

# 2. LỐP BẢO VỆ 1: Kiểm tra cert của server (logic không đổi)
try:
    print("Step 1: Verifying server's certificate against RootCA...")
    with open(ROOT_CA_PATH, "rb") as f:
        root_cert = x509.load_pem_x509_certificate(f.read())

    server_cert = x509.load_pem_x509_certificate(server_cert_pem_bytes)

    root_cert.public_key().verify(
        server_cert.signature,
        server_cert.tbs_certificate_bytes,
        ec.ECDSA(server_cert.signature_hash_algorithm)
    )
    print("OK: Server's certificate is trusted.")
except Exception as e:
    print(f"CRITICAL: Server's certificate cannot be trusted! Aborting. Reason: {e}")
    exit(1)

# 3. LỐP BẢO VỆ 2: Kiểm tra chữ ký của server (logic không đổi)
try:
    print("Step 2: Verifying server's signature on the result data...")
    server_public_key = server_cert.public_key()

    server_public_key.verify(
        server_signature_bytes, # Dùng trực tiếp bytes
        result_bytes, # Dùng trực tiếp bytes
        ec.ECDSA(hashes.SHA256())
    )
    print("OK: Server's signature is valid. Response is authentic and integral.")
except InvalidSignature:
    print("CRITICAL: Invalid signature from server! Response may have been tampered with. Aborting.")
    exit(1)
except Exception as e:
    print(f"CRITICAL: An error occurred while verifying server signature. Aborting. Reason: {e}")
    exit(1)

```

c. Tô chức tài chính (FE Credit)

File HE Server.py: File này triển khai một hệ thống máy chủ (HE Server) sử dụng mã hóa đồng cấu (Homomorphic Encryption) để tính toán điểm tín dụng một cách bảo mật mà không cần giải mã dữ liệu. Máy chủ được xây dựng với FastAPI và tích hợp các chức năng bảo mật.

1. Khởi tạo và cấu hình máy chủ

Cấu hình cơ bản

```
# --- CONFIGURATION ---
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = FastAPI(title="Secure Homomorphic Credit Score Server")

CUSTOM_CA_PATH = "./Certificate/RootCA.crt"
SERVER_KEY_PATH = "./Certificate/FECREDIT.key"
SERVER_CERT_PATH = "./Certificate/FECREDIT.crt"

if not os.path.exists(CUSTOM_CA_PATH):
    raise FileNotFoundError(f"RootCA file not found at: {CUSTOM_CA_PATH}")
```

- FastAPI: Dùng làm framework để xây dựng REST API.
- CUSTOM_CA_PATH: Đường dẫn đến chứng nhận gốc (Root Certificate Authority).
- SERVER_KEY_PATH, SERVER_CERT_PATH: File chứa khóa riêng và chứng nhận của máy chủ.
- Kiểm tra tồn tại: Kiểm tra chứng nhận gốc khi khởi động.

2. Xác thực bảo mật

a. Xác thực chứng nhận

```
# --- SECURITY VERIFICATION FUNCTIONS ---
def verify_certificate_signed_by_root(cert: x509.Certificate, root_cert: x509.Certificate) -> bool:
    try:
        root_pubkey = root_cert.public_key()
        root_pubkey.verify(
            cert.signature,
            cert.tbs_certificate_bytes,
            ec.ECDSA(cert.signature_hash_algorithm)
        )
        return True
    except Exception:
        return False
```

- Chức năng: Kiểm tra xem chứng nhận có được ký bởi RootCA đáng tin cậy hay không.
- Logic: Dùng khóa công khai của RootCA để xác thực chữ ký số trong chứng nhận.

b. Middleware xác thực IP: Kiểm tra IP của client trước khi xử lý yêu cầu.

```
# Danh sách IP cho phép: MSB, ACB, FECREDIT
ALLOWED_IPS = {"192.168.1.11", "192.168.1.12", "192.168.1.14"}
```

```
@app.middleware("http")
async def verify_client_ip(request: Request, call_next):
    client_ip = request.client.host
    if client_ip not in ALLOWED_IPS:
        raise HTTPException(status_code=403, detail="Forbidden: IP not allowed")
    response = await call_next(request)
    return response
```

3. Các hàm tính toán đồng cấu

a. Hàm tính toán các tham số (Params)

Các hàm này sử dụng thư viện mã hóa đồng cấu ('openfhe') để thực hiện các phép toán trên dữ liệu được mã hóa.

i. Hàm tính toán tham số thứ nhất

```
def get_A(crypto_context, S_util, S_inquiries):
    S_inquiries_sq = crypto_context.EvalMult(S_inquiries, S_inquiries)
    result = crypto_context.EvalAdd(S_util, S_inquiries_sq)
    return result
```

- Chức năng: Tính toán tham số thứ nhất bằng cách cộng 'S_util' với bình phương của 'S_inquiries'.

ii. Hàm tính toán tham số thứ hai

```
def get_second_param(crypto_context, S_util, S_behavioral, w2=0.30, w7=0.02):
    w2_p = crypto_context.MakeCKKSPackedPlaintext([w2])
    w7_p = crypto_context.MakeCKKSPackedPlaintext([w7])
    S_util_scaled = crypto_context.EvalMult(S_util, w2_p)
    S_behavioral_scaled = crypto_context.EvalMult(S_behavioral, w7_p)
    S_behavioral_scaled = crypto_context.EvalMult(S_behavioral_scaled, S_behavioral_scaled)
    S_behavioral_scaled = crypto_context.EvalMult(S_behavioral_scaled, crypto_context.MakeCKKSPackedPlaintext([3.0]))
    result = crypto_context.EvalAdd(S_util_scaled, S_behavioral_scaled)
    result = crypto_context.EvalChebyshevFunction(
        func=lambda x: np.sqrt(x),
        ciphertext=result,
        a=0.0,
        b=0.3012,
        degree=15
    )
    return result
```

- Chức năng: Tính toán tham số thứ hai bằng cách sử dụng hàm Chebyshev để tính căn bậc hai.

iii. Hàm tính toán tham số thứ 3:

```

def get_third_param(crypto_context, S_length, S_creditmix, B, w3=0.20, w4=0.10):
    w3_p = crypto_context.MakeCKKSPackedPlaintext([w3])
    w4_p = crypto_context.MakeCKKSPackedPlaintext([w4])
    S_length_scaled = crypto_context.EvalMult(S_length, w3_p)
    S_creditmix_scaled = crypto_context.EvalMult(S_creditmix, w4_p)
    S_creditmix_scaledsqed = crypto_context.EvalMult(S_creditmix_scaled, S_creditmix_scaled)
    B_plus = crypto_context.EvalAdd(B, crypto_context.MakeCKKSPackedPlaintext([1.0]))
    B_plus_inverse = crypto_context.EvalChebyshevFunction(lambda x: 1/x, B_plus, 1, 3, 7)
    S_total = crypto_context.EvalAdd(S_length_scaled, S_creditmix_scaledsqed)
    result = crypto_context.EvalMult(S_total, B_plus_inverse)
    return result

```

- Chức năng: Tính toán tham số thứ ba dựa trên thời gian sử dụng tín dụng ('S_length') và sự đa dạng tín dụng ('S_creditmix').

iv. Hàm tính toán tham số thứ 4:

```

def get_fourth_param(crypto_context, S_inquiries, S_incomestability, w5=0.05, w6=0.03):
    w5_p = crypto_context.MakeCKKSPackedPlaintext([w5])
    w6_p = crypto_context.MakeCKKSPackedPlaintext([w6])
    S_inquiries_scaled = crypto_context.EvalMult(S_inquiries, w5_p)
    S_incomestability_scaled = crypto_context.EvalMult(S_incomestability, w6_p)
    S_total = crypto_context.EvalAdd(S_inquiries_scaled, S_incomestability_scaled)
    S_totalplus = crypto_context.EvalAdd(S_total, crypto_context.MakeCKKSPackedPlaintext([1.0]))
    result = crypto_context.EvalChebyshevFunction(
        func=lambda x: np.log(x),
        ciphertext=S_totalplus,
        a=1.0,
        b=1.08,
        degree=15
    )
    return result

```

- Chức năng: Tính toán tham số thứ tư dựa trên số lượng yêu cầu tín dụng ('S_inquiries') và sự ổn định thu nhập ('S_incomestability').

b. Hàm tính toán điểm tín dụng đồng cầu

```

def homomorphic_credit_score(crypto_context, weights, encrypted_params):
    weighted_scores = []
    A = get_A(crypto_context, encrypted_params['S_util'], encrypted_params['S_inquiries'])
    B = get_B(crypto_context, encrypted_params['S_creditmix'], encrypted_params['S_incomestability'])
    weighted_scores.append(get_first_param(crypto_context, encrypted_params['S_payment'], weights['w1']))
    weighted_scores.append(get_second_param(crypto_context, encrypted_params['S_util'], encrypted_params['S_behavioral'], weights['w2'], weights['w7']))
    weighted_scores.append(get_third_param(crypto_context, encrypted_params['S_length'], encrypted_params['S_creditmix'], B, weights['w3'], weights['w4']))
    weighted_scores.append(get_fourth_param(crypto_context, encrypted_params['S_inquiries'], encrypted_params['S_incomestability'], weights['w5'], weights['w6']))

    final_score = weighted_scores[0]
    for score in weighted_scores[1:]:
        final_score = crypto_context.EvalAdd(final_score, score)

    A_plus = crypto_context.EvalAdd(A, crypto_context.MakeCKKSPackedPlaintext([1.0]))
    A_plus_inverse = crypto_context.EvalChebyshevFunction(lambda x: 1/x, A_plus, 1, 3, 5)
    final_score = crypto_context.EvalMult(final_score, A_plus_inverse)
    return final_score

```

Chức năng: Tính toán điểm tín dụng cuối cùng bằng cách kết hợp các tham số đã mã hóa với các tham số ở phía trước.

c. Hàm tính toán điểm tín dụng đồng cầu đơn giản (nhóm em thực sự dùng)

```

def homomorphic_credit_score_simplified(crypto_context, weights, encrypted_params):
    weighted_scores = []
    S1_weighted = crypto_context.EvalMult(encrypted_params['$_payment'], crypto_context.MakeCKSPackedPlaintext([weights['w1']]))
    S2_weighted = crypto_context.EvalMult(encrypted_params['$_util'], crypto_context.MakeCKSPackedPlaintext([weights['w2']]))
    S3_weighted = crypto_context.EvalMult(encrypted_params['$_length'], crypto_context.MakeCKSPackedPlaintext([weights['w3']]))
    S4_weighted = crypto_context.EvalMult(encrypted_params['$_creditmix'], crypto_context.MakeCKSPackedPlaintext([weights['w4']]))
    S5_weighted = crypto_context.EvalMult(encrypted_params['$_inquiries'], crypto_context.MakeCKSPackedPlaintext([weights['w5']]))
    S6_weighted = crypto_context.EvalMult(encrypted_params['$_incomestability'], crypto_context.MakeCKSPackedPlaintext([weights['w6']]))
    S7_weighted = crypto_context.EvalMult(encrypted_params['$_behavioral'], crypto_context.MakeCKSPackedPlaintext([weights['w7']]))

    weighted_scores.append(S1_weighted)
    weighted_scores.append(S2_weighted)
    weighted_scores.append(S3_weighted)
    weighted_scores.append(S4_weighted)
    weighted_scores.append(S5_weighted)
    weighted_scores.append(S6_weighted)
    weighted_scores.append(S7_weighted)

    final_score = weighted_scores[0]
    for score in weighted_scores[1:]:
        final_score = crypto_context.EvalAdd(final_score, score)

    return final_score

```

Chức năng: Tính toán điểm tín dụng cuối cùng bằng công thức đơn giản hóa đã đề cập ở chương 3.

4. API chính

```

@app.post("/calculate-credit-score")
async def calculate_credit_score(
    eval_mult_key: UploadFile = File(...),
    S_payment: UploadFile = File(...), S_util: UploadFile = File(...), S_length: UploadFile = File(...),
    S_creditmix: UploadFile = File(...), S_inquiries: UploadFile = File(...),
    S_behavioral: UploadFile = File(...), S_incomestability: UploadFile = File(...),
    certificate: UploadFile = File(...),
    signature: str = Form(...),
    metadata: str = Form("{}")
):

```

Chức năng: Nhận dữ liệu mã hóa từ ngân hàng và tính toán điểm tín dụng.

Các bước xử lý:

- Xác thực chứng nhận: Kiểm tra chứng nhận gửi kèm.
- Xác thực chữ ký số: Kiểm tra chữ ký số của dữ liệu.
- Tính toán điểm tín dụng: Thực hiện tính toán đồng cấu trên dữ liệu.
- Ký và trả về kết quả: Tạo chữ ký số cho kết quả và trả về.

5. Xử lý đầu vào và bảo mật

a. Gom dữ liệu FHE và đọc nội dung file

```

logger.info("Received request for credit score calculation. ")

# Gom tất cả các file dữ liệu FHE vào một dict riêng
fhe_data_files = {
    'eval_mult_key': eval_mult_key,
    'S_payment': S_payment, 'S_util': S_util, 'S_length': S_length,
    'S_creditmix': S_creditmix, 'S_inquiries': S_inquiries,
    'S_behavioral': S_behavioral, 'S_incomestability': S_incomestability
}

# Đọc nội dung file
file_contents: Dict[str, bytes] = {}
try:
    # Đọc các file dữ liệu FHE
    for key, upload_file in fhe_data_files.items():
        file_contents[key] = await upload_file.read()

    # Đọc riêng file certificate và metadata
    cert_pem_bytes = await certificate.read()
    metadata_dict = json.loads(metadata)
except Exception:
    raise HTTPException(status_code=400, detail="Invalid file or metadata format.")

```

- Mục tiêu: Gom tất cả các file dữ liệu mã hóa đồng cấu (FHE) nhận được từ ngân hàng thành một dictionary (`file_contents`) để xử lý.
- Chi tiết:
 - Các file dữ liệu đầu vào bao gồm `eval_mult_key`, `S_payment`, `S_util`, v.v.
 - Từng file được đọc và lưu dưới dạng byte để chuẩn bị cho bước xử lý tiếp theo.
 - Chứng nhận (`certificate`) và metadata được đọc riêng để sử dụng trong xác thực.
 - Xử lý lỗi: Nếu có lỗi trong định dạng file hoặc metadata, máy chủ trả về mã lỗi HTTP 400 với thông báo chi tiết.

b. Xác thực chứng nhận (Certificate)

```

# === LỚP BẢO VỆ 1: XÁC THỰC CERTIFICATE ===
logger.info("Verifying sender's certificate...")
try:
    cert = x509.load_pem_x509_certificate(cert_pem_bytes)
    client_public_key = cert.public_key()

    if not isinstance(client_public_key, ec.EllipticCurvePublicKey):
        raise HTTPException(status_code=400, detail="Certificate must use an Elliptic Curve key.")

    with open(CUSTOM_CA_PATH, "rb") as f:
        root_cert = x509.load_pem_x509_certificate(f.read())

    if not verify_certificate_signed_by_root(cert, root_cert):
        logger.warning("Certificate verification failed: Not signed by trusted RootCA.")
        raise HTTPException(status_code=403, detail="Certificate not signed by the trusted RootCA.")

    logger.info("Certificate is valid and trusted.")
except HTTPException as e:
    raise e
except Exception as e:
    logger.error(f"Error processing certificate: {e}")
    raise HTTPException(status_code=400, detail=f"Certificate processing error: {e}")

```

- Mục tiêu: Xác thực chứng nhận được gửi từ ngân hàng để đảm bảo tính hợp lệ và được ký bởi RootCA đáng tin cậy.
- Quy trình:
 - Load chứng nhận: Đọc chứng nhận PEM từ `cert_pem_bytes`.
 - Kiểm tra khóa công khai: Đảm bảo rằng chứng nhận sử dụng khóa Elliptic Curve.
 - Xác thực RootCA: Kiểm tra chữ ký số trong chứng nhận để đảm bảo nó được ký bởi RootCA đáng tin cậy.
- Xử lý lỗi:
 - Nếu chứng nhận không hợp lệ, máy chủ trả về mã lỗi HTTP 403 hoặc HTTP 400 với thông báo chi tiết.
 - Ghi log lỗi để theo dõi.

c. Xác minh chữ ký số (Digital Signature Verification)

```

logger.info("Verifying digital signature...")
try:
    # SỬA ĐỔI CHÍNH Ở ĐÂY
    # Tái tạo dữ liệu đã ký, chỉ bao gồm các file dữ liệu FHE, KHÔNG BAO GỒM certificate.
    data_to_verify = b''
    # Sắp xếp các key của file dữ liệu để đảm bảo thứ tự nhất quán
    for key in sorted(file_contents.keys()):
        data_to_verify += file_contents[key]

    # Thêm metadata đã được chuẩn hóa vào cuối
    data_to_verify += json.dumps(metadata_dict, sort_keys=True).encode('utf-8')

    decoded_sig = base64.b64decode(signature)

    client_public_key.verify( # Dùng public key từ certificate đã được xác thực
        decoded_sig,
        data_to_verify,
        ec.ECDSA(hashes.SHA256())
    )
    logger.info("Digital signature is valid.")
except InvalidSignature:
    logger.warning("Signature verification failed: Invalid signature.")
    raise HTTPException(status_code=403, detail="Invalid digital signature.")
except Exception as e:
    logger.error(f"Error verifying signature: {e}")
    raise HTTPException(status_code=400, detail=f"Error during signature verification: {e}")

```

- Mục tiêu: Xác minh chữ ký số gửi kèm với dữ liệu mã hóa từ ngân hàng để đảm bảo tính toàn vẹn.
- Quy trình
 - Tái tạo dữ liệu đã ký: Gom tất cả file dữ liệu FHE theo thứ tự key và thêm metadata chuẩn hóa.
 - Giải mã chữ ký: Giải mã chữ ký số từ chuỗi Base64.

- Xác minh: Dùng khóa công khai từ chứng nhận đã được xác thực để kiểm tra chữ ký số.
- Xử lý lỗi:
 - Nếu chữ ký không hợp lệ, máy chủ trả về mã lỗi HTTP 403.
 - Nếu có lỗi khác, máy chủ trả về mã lỗi HTTP 400.

d. Xử lý FHE

```

# === ĐẦU ĐẦU CỦA FHE (SAU KHI ĐÃ AN TOÀN) ===
logger.info("Security checks passed. Starting homomorphic computation.")
try:
    cc = init_crypto_context()

    eval_mult_key = fhe.DeserializeEvalKeyString(file_contents['eval_mult_key'], fhe.BINARY)
    if not isinstance(eval_mult_key, fhe.EvalKey): raise ValueError("Invalid FHE evaluation key")
    cc.InsertEvalMultKey([eval_mult_key])

    encrypted_params: Dict[str, Any] = {}
    for key in [k for k in file_contents.keys() if k.startswith('5_')]:
        param = fhe.DeserializeCiphertextString(file_contents[key], fhe.BINARY)
        if not isinstance(param, fhe.Ciphertext): raise ValueError(f"Invalid ciphertext for {key}")
        encrypted_params[key] = param

    weights = {
        'w1': 0.35, 'w2': 0.30, 'w3': 0.20, 'w4': 0.10,
        'w5': 0.05, 'w6': 0.03, 'w7': 0.02
    }

    logger.info("Calculating final encrypted score...")
    encrypted_result = homomorphic_credit_score_simplified(cc, weights, encrypted_params)

    result_data = fhe.Serialize(encrypted_result, fhe.BINARY)
    if not result_data:
        raise HTTPException(status_code=500, detail="Failed to serialize FHE result.")

except Exception as e:
    logger.error(f"Error during FHE processing: {e}\n{traceback.format_exc()}")
    raise HTTPException(status_code=500, detail=f"An error occurred during homomorphic computation: {e}")

```

- Mục tiêu: Thực hiện tính toán điểm tín dụng trên dữ liệu mã hóa.
- Quy trình:
 - Khởi tạo ngữ cảnh mã hóa: Thiết lập môi trường mã hóa với các thông số phù hợp.
 - Deserialize key và ciphertext: Chuyển các file mã hóa từ byte về đối tượng FHE để xử lý.
 - Áp dụng trọng số: Sử dụng các trọng số đã định nghĩa để tính toán điểm tín dụng.
 - Tính toán điểm tín dụng: Gọi hàm `homomorphic_credit_score_simplified`.
 - Kết quả: Serialize kết quả thành byte để trả về.
- Xử lý lỗi: Trả về mã lỗi HTTP 500 nếu xảy ra lỗi trong quá trình tính toán.

e. Ký và trả về kết quả:

```

# MỤC TIÊU: KÝ KẾT QUẢ FHE VÀ TRẢ VỀ DƯỚI DẠNG MULTIPART RESPONSE
logger.info("Signing the response and preparing multipart package...")
try:
    # 1. Load private key và certificate của SERVER (không đổi)
    with open(SERVER_KEY_PATH, "rb") as f:
        server_private_key = serialization.load_pem_private_key(f.read(), password=None)
    with open(SERVER_CERT_PATH, "rb") as f:
        server_cert_pem_bytes = f.read()

    # 2. Dữ liệu cần ký là kết quả FHE (không đổi)
    data_to_sign = result_data

    # 3. Tạo chữ ký (không đổi)
    server_signature_bytes = server_private_key.sign(
        data_to_sign,
        ec.ECDSA(hashes.SHA256())
    )
    # 1. Tạo boundary
    boundary = f"----Boundary{uuid.uuid4().hex}"

    # 2. Hàm tạo từng phần
    def create_part(name, filename, content_type, content: bytes):
        return (
            f"--{boundary}\r\n"
            f"Content-Disposition: form-data; name=\"{name}\"; filename=\"{filename}\"\\r\\n"
            f"Content-Type: {content_type}\\r\\n"
            f"\r\\n"
        ).encode('utf-8') + content + b"\r\\n"

    # 3. Gộp các phần
    body = b''
    body += create_part("result_data", "encryptedResult.bin", "application/octet-stream", result_data)
    body += create_part("server_signature", "signature.sig", "application/octet-stream", server_signature_bytes)
    body += create_part("server_certificate", "server.crt", "application/x-x509-ca-cert", server_cert_pem_bytes)
    body += f"--{boundary}--\\r\\n".encode('utf-8')

    # 4. Trả về multipart response
    return Response(
        content=body,
        media_type=f"multipart/form-data; boundary={boundary}"
    )

```

- Mục tiêu: Ký kết quả FHE và trả về dưới dạng multipart response.
- Quy trình:
 - Ký kết quả: Sử dụng khóa riêng của máy chủ để ký kết quả FHE.
 - Tạo multipart response: Gồm 3 phần:
 - + Kết quả FHE (`result_data`).
 - + Chữ ký số của máy chủ (`server_signature`).
 - + Chứng nhận của máy chủ (`server_certificate`).

IV. Mục tiêu của quy trình

Quy trình này đảm bảo rằng:

- Các bên tham gia không cần giải mã dữ liệu mà vẫn có thể thực hiện tính toán trên dữ liệu đã mã hóa.
- Tính bảo mật và riêng tư của dữ liệu được đảm bảo trong suốt quy trình.

Chương 4: Kết quả và nhận xét

- Video demo em gửi kèm theo báo cáo.
- Nhận xét:
 - Mô hình Homomorphic Encryption đã chứng minh được tiềm năng trong việc xử lý dữ liệu nhạy cảm mà không cần giải mã.
 - Kết quả cho thấy rằng hệ thống có thể áp dụng để tính toán các phép toán tài chính và điểm tín dụng trong thực tế.
 - Tuy nhiên, những file mã hóa, khóa công khai, khóa riêng tư ... của thuật toán này rất nặng. Phải mất gần 2 phút (trong điều kiện LAN) để gửi dữ liệu từ ngân hàng MSB tới server FE Credit.
 - Thuật toán mã hóa này có độ chịu lỗi (noise budget) nhất định. Độ chịu lỗi được tiêu tốn dần mỗi khi thực hiện các phép tính. Nếu vượt quá độ chịu lỗi, bản mã sẽ nhiễu tới mức không thể giải mã được nữa. Đây là hạn chế rất lớn của HE, và là một lý do lớn khiến HE chưa được áp dụng rộng rãi trong thực tế. Để tăng độ chịu lỗi, phải sửa CryptoContext, nhưng khi đó máy em và máy bạn không thể tải nổi thuật toán này nữa (process bị Ubuntu killed).
 - Việc tài nguyên giới hạn khiến nhóm em không thể triển khai hết những gì đã chứng minh bằng PoC.py, đây là một sự thiếu sót.
- Thủ phân tích tĩnh bằng Safety và Bandit:

```
PS D:\Data\Code\NT219_Cryptography\Project> safety scan
Safety 3.5.2 scanning D:\Data\Code\NT219_Cryptography\Project
2025-06-25 01:20:34 UTC

Account: Tùng Nguyễn Việt, nchinh tung@gmail.com
Environment: Stage.development
Scan policy: None, using Safety CLI default policies

Python detected. Found 2 Python requirements files and 1 Python environment

✓ venv\Lib\site-packages\pbr\tests\testpackage\test-requirements.txt: No issues found.

✓ NT219_Project\requirements.txt: No issues found.

✓ venv\pyvenv.cfg: No issues found.

Tested 113 dependencies for security issues using default Safety CLI policies
1 vulnerability found, 1 ignored due to policy.
0 fixes suggested, resolving 0 vulnerabilities.

PS D:\Data\Code\NT219_Cryptography\Project> 
```

```
Run metrics:
  Total issues (by severity):
    Undefined: 0
    Low: 15
    Medium: 1
    High: 0
  Total issues (by confidence):
    Undefined: 0
    Low: 0
    Medium: 1
    High: 15
Files skipped (0):
```

- ⋮ Lỗi Bandit chủ yếu là lỗi về subprocess, nhưng đầu vào subprocess em đã lọc kỹ (hầu hết là hardcoded, không phải đầu vào động) nên không phải những vấn đề lớn

Tài liệu tham khảo

[1] openfhe-python Docs: [CryptoContext — openfhe-python 0.8.0 documentation](#)

[2] Những ví dụ về triển khai Threshold FHE của team openfhe-python: [openfhe-python/examples at main · openfheorg/openfhe-python](#) [openfhe-python/tests at main · openfheorg/openfhe-python](#)

[3] FastAPI Docs: [Tutorial - User Guide - FastAPI](#)

[4] PyQt6 Docs: [Qt for Python](#)

[5] Stack Overflow: [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)

[6] [“Secure financial application using homomorphic encryption”, Vijaykumar Bidve, Aruna Pavate, Rahul Raut, Shailesh Kediya, 2022.](#)

[7] [“Efficient Threshold FHE for Privacy-Preserving Applications”, Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, 2022.](#)

[8] [“Multi-key Fully Homomorphic Encryption Scheme with Compact Ciphertexts”, Tanping Zhou, Long Chen, Xiaoliang Che, 2021.](#)