

DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO BÀI TẬP LỚN HK251

ADVANCED CRYPTOGRAPHY AND CODING THEORY

Học kỳ: 251
Mã môn học: CO3038
Thực hiện: NHÓM 3
Giảng viên: TS. Nguyễn An Khương
Trợ giảng: Trịnh Cao Thắng, Đặng Dương Minh Nhật

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2025

Danh sách thành viên và Phân công nhiệm vụ

Họ và tên	MSSV	Nhiệm vụ	Hoàn thành
VÕ NGUYỄN ĐỨC PHÁT	2212540	Phần 2	100%
NGUYỄN TRUNG PHÚ	2212592	Phần 1	100%
NGUYỄN DĂNG CƯỜNG	2210432	Phần 3.1	100%
THỊNH TRẦN KHÁNH LINH	2211862	Phần 3.3	100%
TRỊNH ĐÌNH KHẢI	2211561	Phần 3.2	100%



Mục lục

1 Security and Unpredictability of PRGs	4
2 Attacking the Vigenere Cipher	19
2.1 Mật mã Caesar	19
2.2 Mật mã Vigenere	19
2.3 Ý tưởng khai thác	20
2.4 Bài toán thực tế	22
2.5 Chạy thử nghiệm	29
3 SMC Exploitation	31
3.1 Interception & API Analysis (Use Burp Suite)	31
3.1.1 POST /session/create?userId=group-1	31
3.1.2 POST /session/exchange?userId=group-1	33
3.1.3 POST /message/send?userId=group-1	36
3.2 Re-implementation & Protocol Reconstruction	43
3.2.1 Xác thực (Authentication)	43
3.2.2 Trao đổi khóa (Key Exchange)	46
3.2.3 Thiết lập phiên (Session Establishment)	49
3.2.4 Gửi tin nhắn bảo mật (Send Message)	50
3.2.5 Các hàm mật mã cơ bản	53
3.2.6 Implementation Code	55
3.2.7 Kết quả thử nghiệm	60
3.3 Exploitation & Proof-of-Concept	63
3.3.1 Xác định lỗ hổng	63
3.3.2 Khai thác lỗ hổng	64
3.3.3 Khắc phục lỗ hổng	68



1 Security and Unpredictability of PRGs

- 1 point). Read in BS15 and clearly define and explain the following concepts in your report: *pseudo-random generator (PRG)*, *PRG security game*, *secure PRG*, *unpredictable PRG game*, and *unpredictable PRG*. If you use additional definitions or variations from other references, include them as well and provide proper citations.

Pseudo-random Generator (PRG)

Dịnh nghĩa

Một Bộ tạo số giả ngẫu nhiên (Pseudo-Random Generator – viết tắt là PRG) là một thuật toán xác định và hiệu quả (efficient deterministic algorithm) dùng để mở rộng một chuỗi ngắn ngẫu nhiên thành một chuỗi dài hơn trông giống như ngẫu nhiên thật.

Theo BS15, một Bộ tạo số giả ngẫu nhiên (PRG) là một thuật toán G thỏa mãn hai điều kiện:

- Tính hiệu quả (Efficient): G là một thuật toán chạy trong thời gian đa thức, nghĩa là nó phải nhanh và thực tế để tính toán.
- Tính xác định (Deterministic): G không chứa yếu tố ngẫu nhiên. Nếu cung cấp cùng một đầu vào hai lần, nó phải luôn tạo ra chính xác cùng một đầu ra.

Một PRG gồm

- Một thuật toán xác định hiệu quả G ,
- Một không gian hạt giống S ,
- Và một không gian đầu ra R .

Khi nhận $s \in S$, PRG xuất ra $r = G(s) \in R$.

Thông thường $S = \{0, 1\}^\ell$, $R = \{0, 1\}^L$, và $L > \ell$.

Mục đích chính của PRG

Mục đích chính của PRG là tạo ra một chuỗi bit dài có tính ngẫu nhiên không phân biệt được. Nó nhận một chuỗi bit đầu vào ngắn và ngẫu nhiên thật sự, gọi là hạt giống (seed) s , và tạo ra một chuỗi bit đầu ra dài hơn nhiều r , gọi là chuỗi giả ngẫu nhiên.

Ví dụ: Nếu hạt giống s có độ dài ℓ bit (ví dụ: $\ell = 256$ bit) và đầu ra r có độ dài L bit (ví dụ: $L = 1,000,000$ bit), thì ta luôn có $L > \ell$.



Ứng dụng trong Stream Cipher

PRG được dùng trong Stream Cipher (Mật mã dòng):

- Hạt giống s đóng vai trò khóa bí mật.
- PRG sinh ra chuỗi bit dài $r = G(s)$, gọi là keystream.
- Chuỗi keystream này được XOR với thông điệp m để mã hóa $c = m \oplus r$.
- Giải mã bằng cách $m = c \oplus r$.

Nếu PRG đủ mạnh, ciphertext trông hoàn toàn ngẫu nhiên – kẻ tấn công không phân biệt được ciphertext và một chuỗi ngẫu nhiên, từ đó đạt tính bảo mật ngữ nghĩa (Semantic Security).

(Tính bảo mật ngữ nghĩa yêu cầu rằng việc quan sát ciphertext không giúp kẻ tấn công rút ra bất kỳ thông tin hữu ích nào về thông điệp gốc. Nghĩa là ciphertext trông không khác gì một chuỗi ngẫu nhiên đối với đối thủ. Khi PRG đủ mạnh, mã hóa bằng XOR sẽ đạt được tính chất này, vì keystream che giấu hoàn toàn nội dung của thông điệp.)

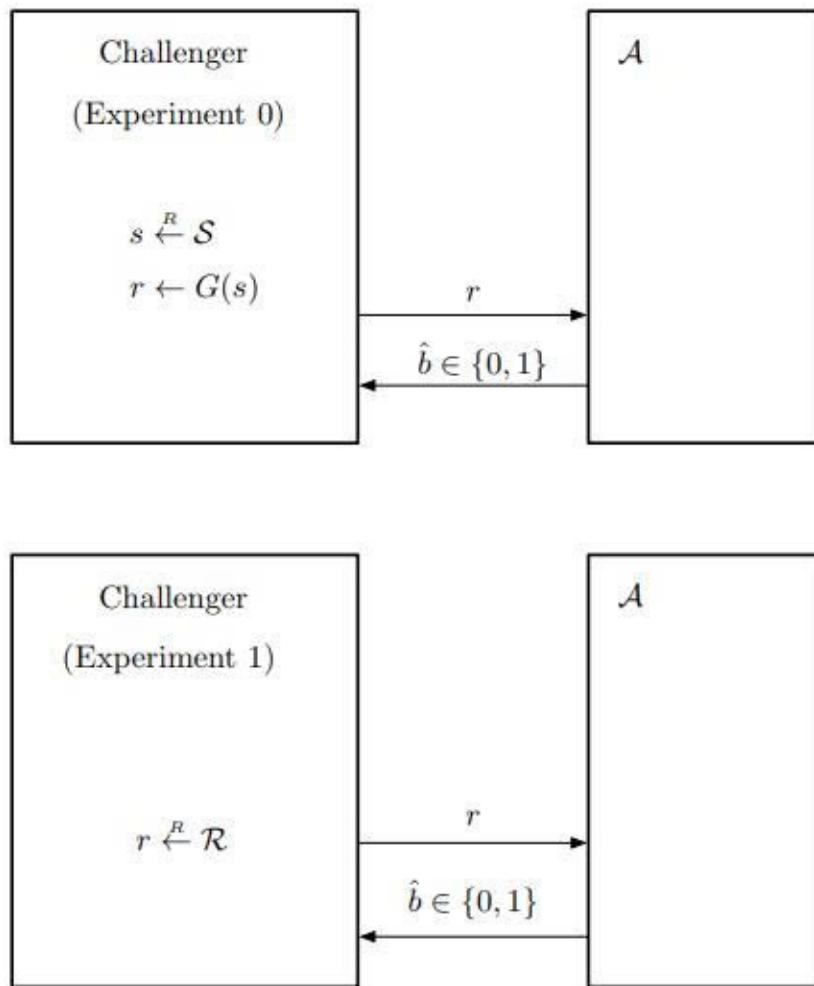
PRG security game

Trò chơi Bảo mật PRG, hay Attack Game 3.1 theo BS15, là một thí nghiệm được xây dựng để cung cấp một định nghĩa toán học chính xác cho tính "an toàn" của một Bộ tạo số giả ngẫu nhiên.

Một PRG G được coi là an toàn nếu đầu ra của nó *trông giống hệt* như một chuỗi ngẫu nhiên thật sự — nghĩa là không một thuật toán hiệu quả nào (kẻ tấn công) có thể phân biệt được hai trường hợp này.

Trò chơi diễn ra giữa hai thực thể:

- Người thách thức (Challenger): Điều hành trò chơi và nắm giữ bí mật (chuỗi là giả hay thật).
- Kẻ tấn công (Adversary) A : Một thuật toán hiệu quả (chạy trong thời gian đa thức) cố gắng xác định bí mật của Người thách thức.



Hình 1: Thí nghiệm 0 và 1 của Attack Game

Quy trình trò chơi được định nghĩa như sau:

- **Bước 1: Thiết lập Bí mật**

Người thách thức bí mật chọn một bit $b \in \{0, 1\}$. Bit b này quyết định thí nghiệm mà Kẻ tấn công sẽ phải đối mặt.

- **Bước 2: Tạo Chuỗi Thử thách r**

Người thách thức tạo ra một chuỗi bit r độ dài L dựa trên b :

- Nếu $b = 0$ (Thí nghiệm 0): Người thách thức chọn một hạt giống (seed) s ngẫu nhiên thật sự từ không gian hạt giống S (ký hiệu $s \leftarrow_R S$). Sau đó, Người thách thức tính toán chuỗi giả ngẫu nhiên bằng cách chạy PRG: $r := G(s)$.



- Nếu $b = 1$ (Thí nghiệm 1): Người thách thức chọn một chuỗi r hoàn toàn ngẫu nhiên và đồng đều từ không gian đầu ra R (ký hiệu $r \leftarrow_R R$).

- Bước 3: Gửi Thủ thách

Người thách thức gửi duy nhất chuỗi r (đã được tạo ở Bước 2) cho Kẻ tấn công A . A không biết giá trị của b .

- Bước 4: Lựa chọn của Kẻ tấn công Kẻ tấn công A nhận r , thực hiện các phân tích, tính toán của mình và xuất ra một bit dự đoán $\hat{b} \in \{0, 1\}$. Mục tiêu của A là làm cho $\hat{b} = b$.

Để định lượng khả năng thành công của A , chúng ta định nghĩa "Ưu thế" (Advantage) của A .

Gọi W_b là sự kiện Kẻ tấn công A xuất ra 1 (tức $\hat{b} = 1$)..

- $\Pr[W_0]$: Xác suất A xuất ra 1 khi nhận được chuỗi Giả ngẫu nhiên ($b = 0$).
- $\Pr[W_1]$: Xác suất A xuất ra 1 khi nhận được chuỗi Ngẫu nhiên thật ($b = 1$).

Ưu thế của A trong trò chơi này, ký hiệu là $\text{PRGadv}[A, G]$, được định nghĩa là sự khác biệt tuyệt đối giữa hai xác suất này:

$$\text{PRGadv}[A, G] := |\Pr[W_0] - \Pr[W_1]|$$

Diễn giải:

- Nếu G là một PRG an toàn: Kẻ tấn công A không thể phân biệt được hai thế giới. Do đó, hành vi của A trong cả hai thí nghiệm sẽ gần như giống hệt nhau. A sẽ xuất ra 1 (hoặc 0) với xác suất gần như tương đương, bất kể nó đang ở Thí nghiệm 0 hay 1.
 - Trong trường hợp này, $\Pr[W_0] \approx \Pr[W_1]$, và do đó ưu thế $\text{PRGadv}[A, G]$ sẽ rất gần 0 (tức là không đáng kể).
- Nếu G là một PRG không an toàn (bị lỗi): Giả sử G có một khuyết điểm khuyếch đại không được tốt (ví dụ: Các bit trong chuỗi đầu ra của G không độc lập với nhau mà có sự tương quan).

 - Ví dụ, nửa đầu của chuỗi r chứa thông tin cho phép suy luận về nửa sau của nó. Kẻ tấn công A có thể sử dụng nửa đầu chuỗi để kiểm tra tính hợp lệ của nửa sau. Trong khi một chuỗi ngẫu nhiên thật sự không có bất kỳ mối liên kết nào giữa các phần, chuỗi từ G sẽ bộc lộ cấu trúc nội tại này. Sự khác biệt trong cấu trúc đó chính là "dấu hiệu nhận biết" giúp A phân biệt dễ dàng hai thế giới, phá vỡ tính bảo mật của PRG.

Secure PRG

Theo định nghĩa trong BS15: "A PRG G is secure if the value $\text{PRGadv}[A, G]$ is negligible for all efficient adversaries A ."



Nghĩa là: Một bộ tạo số giả ngẫu nhiên G được xem là an toàn nếu giá trị lợi thế $\text{PRGadv}[A, G]$ là không đáng kể, đối với mọi kẻ tấn công hiệu quả A .

“efficient adversary” có nghĩa là kẻ tấn công có khả năng tính toán trong giới hạn thực tế.

Ta chỉ xét những kẻ tấn công có thời gian hoạt động đa thức theo độ dài đầu vào, tức là các thuật toán có thể chạy được trong thời gian hợp lý.

- Thuật toán được gọi là hiệu quả nếu số bước xử lý tăng không quá nhanh khi kích thước đầu vào tăng (ví dụ tỉ lệ n^2, n^3 , hoặc n^5).
- Ngược lại, các thuật toán có độ phức tạp hàm mũ (như 2^n) hoặc giai thừa (như $n!$) là không hiệu quả vì thời gian thực hiện vượt quá khả năng của mọi máy tính hiện nay.

Vì vậy, khi nói “mọi kẻ tấn công hiệu quả”, ta chỉ xét đến những thuật toán có thể thực thi trong thực tế, chứ không xét các phương pháp tưởng tượng hoặc có độ phức tạp quá lớn.

Cụm “the value $\text{PRGadv}[A, G]$ is negligible” có nghĩa là lợi thế của kẻ tấn công trong việc phân biệt PRG với chuỗi ngẫu nhiên thật là cực kỳ nhỏ – đến mức có thể xem như bằng 0 trong thực tế.

Một hàm $\varepsilon(n)$ được gọi là *không đáng kể* nếu nó giảm nhanh hơn mọi hàm đa thức. Nói cách khác, với mọi số $c > 0$, tồn tại giá trị N sao cho nếu $n > N$ thì $\varepsilon(n) < \frac{1}{n^c}$.

Ví dụ:

- 2^{-n} là hàm không đáng kể.
- $\frac{1}{n}$ hoặc $\frac{1}{n^2}$ thì không, vì chúng giảm quá chậm.

Điều đó có nghĩa là, nếu ta tăng độ dài khóa n , xác suất tấn công thành công giảm nhanh đến mức vô nghĩa trong thực tế.

Giả sử PRG có độ dài khóa 128 bit. Kẻ tấn công có thể cố gắng phân biệt PRG với chuỗi ngẫu nhiên bằng cách thử mọi hạt giống có thể. Số lượng trường hợp cần thử là 2^{128} — quá lớn để thực hiện, ngay cả với các siêu máy tính hiện nay.

Vì vậy:

- Kẻ tấn công vẫn là hiệu quả nếu chỉ chạy trong thời gian đa thức, ví dụ n^3
- Nhưng việc thử toàn bộ khóa là không hiệu quả, nên không được tính là “adversary” trong định nghĩa trên.

Khi đó, xác suất để một kẻ tấn công hiệu quả thành công là rất nhỏ, có thể nhỏ hơn 2^{-100} , và được xem là không đáng kể.

Định nghĩa này cho phép thay thế yêu cầu “bảo mật tuyệt đối” (perfect security) — vốn không thể đạt được nếu khóa ngắn hơn thông điệp — bằng yêu cầu “bảo mật tính toán” (computational security), tức là an toàn trước mọi kẻ tấn công có khả năng tính toán hợp lý. Một hệ thống được coi là an toàn không phải vì không thể bị phá, mà vì không thể bị phá trong thực tế.



Unpredictable PRG Game

Trò chơi PRG Không thể đoán được, hay Attack Game 3.2 theo BS15, cung cấp một định nghĩa bảo mật thay thế cho PRG. Thay vì kiểm tra khả năng phân biệt toàn bộ chuỗi (như Attack Game 3.1), trò chơi này kiểm tra xem liệu một bit cụ thể trong chuỗi có thể bị dự đoán hay không.

Ý tưởng cốt lõi là "Bài kiểm tra bit tiếp theo" (Next-Bit Test): Một PRG được coi là tốt nếu, ngay cả khi biết tất cả các bit trước đó, không ai có thể dự đoán bit tiếp theo với xác suất thành công tốt hơn đáng kể so với việc đoán mò (50/50).

Các bên tham gia vào:

- Người thách thức (Challenger): Thực thể tạo ra chuỗi giả ngẫu nhiên.
- Kẻ tấn công (Adversary, A): Một thuật toán hiệu quả cố gắng dự đoán một bit của chuỗi.

Quy trình trò chơi được định nghĩa như sau:

- Bước 1: Chọn Vị trí

Kẻ tấn công A quyết định vị trí i (với $0 \leq i < L$, trong đó L là độ dài chuỗi đầu ra) mà nó muốn dự đoán. A gửi chỉ số i này cho Người thách thức.

- Bước 2: Tạo Chuỗi

Người thách thức chọn một hạt giống (seed) ngẫu nhiên $s \leftarrow_R S$ và tính toán toàn bộ chuỗi giả ngẫu nhiên $r := G(s)$.

- Bước 3: Cung cấp Tiền tố (Prefix)

Người thách thức gửi cho A tiền tố của r , tức là tất cả các bit đúng trước vị trí i . Chuỗi được gửi là: $r[0 \dots i - 1]$.

- Bước 4: Dự đoán

Kẻ tấn công A phân tích tiền tố đã nhận và xuất ra một bit dự đoán $g \in \{0, 1\}$. Mục tiêu của A là dự đoán đúng giá trị của bit $r[i]$.

Kẻ tấn công A được coi là "thắng" nếu dự đoán của nó chính xác, tức là $g = r[i]$.

Nếu một chuỗi là ngẫu nhiên thật sự, việc biết các bit trước đó không cung cấp thông tin gì về bit tiếp theo, vì vậy xác suất đoán đúng luôn là 1/2. Ưu thế của A đo lường mức độ nó có thể làm tốt hơn việc đoán mò ngẫu nhiên.

Ưu thế dự đoán (Prediction Advantage) của A , ký hiệu là $\text{Predadv}[A, G]$, được định nghĩa là:

$$\text{Predadv}[A, G] := |\Pr[A \text{ thắng}] - \frac{1}{2}|$$

Điễn giải:



- Nếu G là một PRG tốt (không thể đoán được), thì tiền tố $r[0 \dots i - 1]$ không tiết lộ thông tin hữu ích nào về $r[i]$. Do đó, A không thể làm gì tốt hơn là đoán mò, $\Pr[A \text{ thắng}] \approx 1/2$, và ưu thế Predadv sẽ là không đáng kể.
- Nếu G bị lỗi (ví dụ: G có một quy luật: "bit thứ i luôn bằng bit thứ $i - 1$ "), A có thể lợi dụng điều này. A sẽ chọn vị trí i đó, nhận tiền tố, và chỉ cần đoán $g = r[i - 1]$. Trong trường hợp này, $\Pr[A \text{ thắng}] = 1$, và ưu thế Predadv sẽ là $|1 - 1/2| = 1/2$, là một giá trị đáng kể, cho thấy G đã bị phá vỡ.

Unpredictable PRG

Theo định nghĩa trong BS15: "A PRG G is unpredictable if the value $\text{Predadv}[A, G]$ is negligible for all efficient adversaries A ."

Nghĩa là: Một Bộ tạo số giả ngẫu nhiên (PRG) G được gọi là không thể đoán định (unpredictable) nếu giá trị ưu thế dự đoán $\text{Predadv}[A, G]$ là không đáng kể (negligible) đối với tất cả các kẻ tấn công hiệu quả (efficient adversaries) A .

Để thỏa mãn định nghĩa này, ba yếu tố sau đây phải đồng thời được đáp ứng:

- Đối tượng tấn công (A - Efficient Adversary): Định nghĩa giới hạn phạm vi đe dọa ở các "kẻ tấn công hiệu quả". Ta chỉ xét những kẻ tấn công có thời gian hoạt động đa thức. Điều này loại trừ các mô hình tấn công lý thuyết có sức mạnh tính toán vô hạn (như tấn công vét cạn không gian khóa lớn), tập trung vào tính an toàn trước các máy tính thực tế.
- Thước đo lợi thế ($\text{Predadv}[A, G]$): Thước đo khả năng chiến thắng của A trong Trò chơi 3.2 (Next-Bit Test). Giá trị này được tính bằng độ lệch tuyệt đối giữa xác suất đoán đúng của A và xác suất ngẫu nhiên

$$\text{Predadv}[A, G] = |\Pr[A \text{ đoán đúng}] - 1/2|$$

- Ngưỡng an toàn (Negligible): Ưu thế của kẻ tấn công phải là một hàm "không đáng kể". Về mặt toán học, hàm này giảm nhanh hơn nghịch đảo của bất kỳ đa thức nào khi tham số bảo mật tăng lên.
 - Xác suất A đoán đúng bit tiếp theo chỉ được phép xấp xỉ $1/2$ (ví dụ: $0.5 + 2^{-128}$). Phần dư (2^{-128}) là quá nhỏ để có thể khai thác trong thực tế.

Ý nghĩa thực tế:

- Mặc dù G là một thuật toán xác định, định nghĩa này yêu cầu rằng việc nắm giữ toàn bộ lịch sử đầu ra (các bit từ 0 đến $i - 1$) không cung cấp bất kỳ lợi thế tính toán nào để dự báo bit thứ i .



- Một PRG thỏa mãn định nghĩa này đảm bảo rằng mỗi bit mới được sinh ra đều mang tính "bất ngờ" đối với kẻ tấn công, tương đương với việc tung một đồng xu công bằng.
2. (1 point). Let G be a pseudo-random generator. Prove that if G is **secure**, then G is **unpredictable**. Formally, show that if there exists an adversary A that wins the *Unpredictable PRG Game* with non-negligible advantage, then there exists a reduction (adversary) B that wins the *PRG Security Game* with the same advantage as A . Provide a clear construction of B and explain how the advantage is preserved.

Giả định

Trong phần này, ta sẽ đi chứng minh một định lý nền tảng của bộ tạo số giả ngẫu nhiên (PRG): Nếu một PRG G là an toàn (secure), thì G cũng không thể dự đoán được (unpredictable).

Ta sẽ sử dụng phương pháp chứng minh mệnh đề phản đảo : "Nếu G không "không thể dự đoán được", thì G cũng không "an toàn". Có nghĩa là ta sẽ chứng minh nếu G có thể dự đoán được thì G không an toàn.

Việc G không "không thể dự đoán được" có nghĩa là tồn tại một đối thủ (bộ dự đoán) A có thể thắng Trò chơi Unpredictable PRG (Game 3.2) với một lợi thế $\epsilon > 0$ (không tầm thường).

Để đơn giản, ta giả định A có lợi thế ϵ trong việc dự đoán một bit cụ thể i (với $1 \leq i \leq L$), khi biết $i - 1$ bit đúng trước. Xác suất thành công của A được định nghĩa là:

$$\Pr_{s \leftarrow \{0,1\}^s} [A(G(s)[0\dots i-1]) = G(s)[i]] = \frac{1}{2} + \epsilon$$

Lợi thế của A là $\text{Predadv}[A, G] = \epsilon$.

Xây dựng Đối thủ B

Từ giả định về sự tồn tại của A , chúng ta sẽ xây dựng một đối thủ mới B . Mục tiêu của B là phá vỡ tính an toàn của G bằng cách thắng PRG Security Game.

B sẽ sử dụng A như một chương trình con

Logic của B : B tham gia Trò chơi An toàn và nhận một chuỗi r . B không biết r là giả ngẫu nhiên (Exp 0) hay ngẫu nhiên thật (Exp 1). B sẽ sử dụng A để kiểm tra:

- Nếu r là giả ngẫu nhiên, A (theo giả định) có khả năng dự đoán bit $r[i]$ rất tốt (xác suất $1/2 + \epsilon$).
- Nếu r là ngẫu nhiên thật, A không có lợi thế gì, vì $r[i]$ hoàn toàn độc lập với các bit trước. A chỉ đoán đúng với xác suất $1/2$.



Do đó, B sẽ đoán rằng r là giả ngẫu nhiên nếu A dự đoán đúng, và đoán r là ngẫu nhiên thật nếu A dự đoán sai.

Thuật toán của $B(r)$: B nhận một chuỗi đầu vào r dài L bit từ bộ thách thức của mình.

1. Nhận chuỗi: B nhận $r = r_0r_1\dots r_{L-1}$.
2. Trích xuất tiền tố: B lấy $i - 1$ bit đầu tiên của r và đặt $x \leftarrow r[0\dots i - 1]$.
3. Gọi A : B cung cấp x làm đầu vào cho A . A trả về một bit dự đoán $g \leftarrow A(x)$.
4. Quyết định: B so sánh dự đoán g của A với bit $r[i]$ thực tế trong chuỗi r :
 - Nếu $g == r[i]$ (tức là A đoán đúng): B output 1 (nghĩ rằng r là chuỗi giả ngẫu nhiên từ Exp 0).
 - Nếu $g \neq r[i]$ (tức là A đoán sai): B output 0 (nghĩ rằng r là chuỗi ngẫu nhiên thật từ Exp 1).

Phân tích Lợi thế của B

Bây giờ, chúng ta sẽ phân tích lợi thế của B (ký hiệu là $\text{PRGadv}[B, G]$). Lợi thế này được định nghĩa là:

$$\text{PRGadv}[B, G] = |\Pr[B(r) = 1|\text{Exp 1}] - \Pr[B(r) = 1|\text{Exp 0}]|$$

Lợi thế của đối thủ B trong việc phá vỡ bộ tạo số giả ngẫu nhiên G chính là trị tuyệt đối của sự chênh lệch giữa xác suất B xuất ra kết quả là 1 khi đối mặt với chuỗi ngẫu nhiên thật (trong Thí nghiệm 1) và xác suất B xuất ra kết quả là 1 khi đối mặt với chuỗi giả ngẫu nhiên (trong Thí nghiệm 0).

- Phân tích Thí nghiệm 0 (Exp 0: r là chuỗi PRG)

Trong thí nghiệm này, r là một chuỗi giả ngẫu nhiên, $r = G(s)$ với s được chọn ngẫu nhiên.

- Chúng ta cần tính $\Pr[B(r) = 1|\text{Exp 0}]$. Theo thuật toán của B , B output 1 khi và chỉ khi $g == r[i]$.
- Thay thế các biến từ thuật toán, B output 1 khi $A(r[0\dots i - 1]) == r[i]$. Vì $r = G(s)$, sự kiện này tương đương với: $A(G(s)[0\dots i - 1]) == G(s)[i]$.
- Đây chính xác là sự kiện A thắng Trò chơi Unpredictable PRG. Theo giả định ở Mục 1, xác suất của sự kiện này là $\frac{1}{2} + \epsilon$.

Do đó, ta có: $\Pr[B(r) = 1|\text{Exp 0}] = \frac{1}{2} + \epsilon$

- Phân tích Thí nghiệm 1 (Exp 1: r là chuỗi Ngẫu nhiên thật)

Trong thí nghiệm này, r là một chuỗi ngẫu nhiên thật, được chọn đồng nhất từ $\{0, 1\}^L$.

- Chúng ta cần tính $\Pr[B(r) = 1|\text{Exp 1}]$. B output 1 khi và chỉ khi $A(r[0\dots i - 1]) == r[i]$.



- Trong một chuỗi ngẫu nhiên thật, bit $r[i]$ là một biến ngẫu nhiên độc lập, được chọn đồng nhất (xác suất $1/2$ là 0 , $1/2$ là 1). Quan trọng nhất, $r[i]$ hoàn toàn độc lập về mặt thống kê với tất cả các bit trước nó ($r[0...i-1]$).
- Khi A nhận đầu vào $x = r[0...i-1]$, nó sẽ tính toán và đưa ra một dự đoán g (là 0 hoặc 1). Giá trị g này hoàn toàn phụ thuộc vào x .
- Vì $r[i]$ độc lập với x , nó cũng độc lập với g (là hàm của x).
- Do đó, chúng ta đang so sánh g (một giá trị đã được xác định) với $r[i]$ (một biến ngẫu nhiên $50/50$). Xác suất chúng khớp nhau luôn là $1/2$, bất kể g là 0 hay 1 .

Do đó, ta có: $\Pr[B(r) = 1 | \text{Exp } 1] = \frac{1}{2}$

Chúng ta thay hai xác suất vừa tính được vào công thức lợi thế của B :

$$\text{PRGadv}[B, G] = |\Pr[B(r) = 1 | \text{Exp } 1] - \Pr[B(r) = 1 | \text{Exp } 0]|$$

$$\text{PRGadv}[B, G] = \left| \frac{1}{2} - \left(\frac{1}{2} + \epsilon \right) \right|$$

$$\text{PRGadv}[B, G] = |- \epsilon| = \epsilon$$

Kết quả: Chúng ta đã chỉ ra rằng $\text{PRGadv}[B, G] = \text{Predadv}[A, G] = \epsilon$.

Điều này có nghĩa là nếu tồn tại một bộ dự đoán A có lợi thế ϵ (không tầm thường) trong việc phá vỡ tính không thể dự đoán được của G , thì cũng tồn tại một bộ phân biệt B có cùng lợi thế ϵ (không tầm thường) trong việc phân biệt đầu ra của PRG với chuỗi ngẫu nhiên thật.

Nói cách khác, việc G có thể dự đoán được kéo theo rằng G không an toàn, vì tồn tại một adversary B phá vỡ định nghĩa PRG-security (một PRG an toàn yêu cầu rằng mọi đối thủ phân biệt chỉ được có lợi thế không tầm thường).

Do đó, ta đã chứng minh được mệnh đề phản đảo:

$$\neg \text{Unpredictable}(G) \Rightarrow \neg \text{Secure}(G).$$

Và vì mệnh đề phản đảo tương đương logic với mệnh đề gốc, ta kết luận rằng bất kỳ PRG G nào an toàn thì phải không thể dự đoán được.

3. (1 point). Prove the **converse direction**: if G is **unpredictable**, then G is **secure**. Formally, construct an adversary B that wins the *PRG Security Game* given any adversary A that wins the *Unpredictable PRG Game*, and describe the relationship between their advantages. Conclude that the two notions, *security* and *unpredictability*, are equivalent for PRGs.

Mục tiêu của chúng ta là chứng minh theo chiều ngược lại: Nếu một PRG G là không thể dự đoán (unpredictable), thì G là an toàn (secure).

Việc chứng minh trực tiếp rất khó khăn vì "An toàn" là một tính chất tổng quát về phân bố xác suất của toàn bộ chuỗi, trong khi "Không thể dự đoán" là tính chất cụ thể của từng bit. Do



đó, chúng ta sử dụng phương pháp chứng minh mệnh đề phản đảo: Nếu G KHÔNG an toàn thì F CÓ THỂ dự đoán

Chúng ta bắt đầu với giả định rằng G không an toàn. Điều này nghĩa là tồn tại một đối thủ A (Distinguisher) có thể phân biệt chuỗi giả ngẫu nhiên $G(s)$ với chuỗi ngẫu nhiên thật r với lợi thế đáng kể ϵ :

$$|\Pr[A(G(s)) = 1] - \Pr[A(r) = 1]| = \epsilon$$

(Với ϵ là không không đáng kể).

Vấn đề nằm ở chỗ: Đối thủ A nhìn vào toàn bộ chuỗi (ví dụ 128 bit) và nói "Đây là giả" hoặc "Đây là thật". A không chỉ ra bit nào bị lỗi, hay bit nào làm lộ tính giả ngẫu nhiên.

- Để chứng minh G "Có thể dự đoán", ta cần xây dựng một đối thủ B chỉ nhìn vào một phần chuỗi và đoán chính xác bit kế tiếp.
- Nút thắt: Làm thế nào để chuyển đổi khả năng "phân biệt toàn bộ chuỗi" của A thành khả năng "dự đoán 1 bit cụ thể" của B?

Để giải quyết vấn đề "nút thắt" đã nêu ở trên (chuyển đổi khả năng phân biệt toàn bộ chuỗi thành khả năng dự đoán một bit), chúng ta sử dụng kỹ thuật Đối số Lai (Hybrid Argument). Thay vì so sánh trực tiếp chuỗi giả ngẫu nhiên và chuỗi ngẫu nhiên thật, ta sẽ so sánh các chuỗi trung gian biến đổi dần dần.

Để làm rõ cơ chế này, ta sẽ xem xét một trường hợp cụ thể với đầu ra có độ dài 3 bit.

Giả sử bộ tạo số G tạo ra chuỗi 3 bit: $G(s) = (g_1, g_2, g_3)$. Đối thủ A phân biệt được $G(s)$ với chuỗi ngẫu nhiên $R = (r_1, r_2, r_3)$ với lợi thế ϵ .

Chúng ta xây dựng 4 phân bô lai (H_0 đến H_3) tạo thành một câu nói từ "Hoàn toàn thật" đến "Hoàn toàn giả":

- H_0 (Toàn bộ là thật): (r_1, r_2, r_3)
- H_1 (Bit 1 giả): (\mathbf{g}_1, r_2, r_3)
- H_2 (Bit 1, 2 giả): $(\mathbf{g}_1, \mathbf{g}_2, r_3)$
- H_3 (Toàn bộ là giả): $(\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3)$

Lợi thế tổng thể của A là sự khác biệt giữa hai đầu mút:

$$\epsilon = |\Pr[A(H_3) = 1] - \Pr[A(H_0) = 1]|$$

Biến đổi khoảng cách tổng này bằng tổng các khoảng cách nhỏ liền kề:

$$\epsilon = |(\Pr[H_3] - \Pr[H_2]) + (\Pr[H_2] - \Pr[H_1]) + (\Pr[H_1] - \Pr[H_0])|$$



(Viết tắt $\Pr[H_i]$ thay cho $\Pr[A(H_i) = 1]$ để gọn hơn).

Sử dụng bất đẳng thức tam giác, ta có:

$$\epsilon \leq |\Pr[H_3] - \Pr[H_2]| + |\Pr[H_2] - \Pr[H_1]| + |\Pr[H_1] - \Pr[H_0]|$$

$$(\forall a, b, c \in \mathbb{R}, |a + b + c| \leq |a| + |b| + |c|.)$$

Tổng của 3 số hạng trên lớn hơn hoặc bằng ϵ . Do đó, theo nguyên lý trung bình, phải tồn tại ít nhất một cặp liền kề (H_i, H_{i-1}) mà sự khác biệt giữa chúng là đáng kể (ít nhất là $\epsilon/3$).

Giả sử cặp đó là H_2 và H_1 :

- $H_1 = (g_1, \mathbf{r}_2, r_3)$
- $H_2 = (g_1, \mathbf{g}_2, r_3)$

Hai chuỗi này giống hệt nhau ở bit đầu (g_1) và bit cuối (r_3). Sự khác biệt duy nhất nằm ở vị trí thứ 2: một bên là ngẫu nhiên thật (r_2), một bên là giả ngẫu nhiên (g_2). → Nếu A phân biệt được H_2 và H_1 , thì A đang phân biệt được bit thứ 2. Đây chính là cơ sở để ta dự đoán bit thứ 2.

Dựa trên trực giác từ mô hình 3-bit, ta mở rộng chứng minh cho độ dài $L(n)$ bất kỳ.

Gọi $L = L(n)$ là độ dài đầu ra của G . Ta định nghĩa một dãy $L + 1$ phân bố lai H_k (với $k \in \{0, \dots, L\}$):

- H_k : Là phân bố trong đó k bit đầu tiên được lấy từ $G(s)$ (giả ngẫu nhiên), và $L - k$ bit còn lại được lấy từ phân bố thống nhất U (ngẫu nhiên thật).

$$H_k = G(s)_1 \dots G(s)_k \| r_{k+1} \dots r_L$$

Trong đó:

- $H_0 = r_1 r_2 \dots r_L$ ($k = 0 \iff r_1 r_2 \dots r_L$)
(Phân bố Ngẫu nhiên thật — tất cả L bit đều là ngẫu nhiên.)
- $H_L = G(s)_1 G(s)_2 \dots G(s)_L$ ($k = L \iff G(s)_1 G(s)_2 \dots G(s)_L$)
(Phân bố Giả ngẫu nhiên của G — toàn bộ L bit đều là đầu ra của PRG.)

Giả sử G không an toàn, tức là tồn tại đối thủ A sao cho:

$$|\Pr[A(H_L) = 1] - \Pr[A(H_0) = 1]| = \epsilon$$



Ta khai triển hiệu số này thành tổng của các bước nhảy đơn lẻ (tương tự ví dụ 3-bit):

$$\epsilon = \left| \sum_{k=1}^L (\Pr[A(H_k) = 1] - \Pr[A(H_{k-1}) = 1]) \right|$$

Áp dụng bất đẳng thức tam giác:

$$\epsilon \leq \sum_{k=1}^L |\Pr[A(H_k) = 1] - \Pr[A(H_{k-1}) = 1]|$$

Vì tổng của L số hạng không âm $\geq \epsilon$, tồn tại ít nhất một chỉ số $k^* \in \{1, \dots, L\}$ sao cho sự khác biệt là lớn nhất và thỏa mãn:

$$|\Pr[A(H_{k^*}) = 1] - \Pr[A(H_{k^*-1}) = 1]| \geq \frac{\epsilon}{L}$$

(*)

Tại đây, ta thấy cấu trúc của hai phân bố liền kề này:

- $H_{k^*-1} = \text{Prefix} \parallel \mathbf{r}_{k^*} \parallel \text{Suffix}$
- $H_{k^*} = \text{Prefix} \parallel \mathbf{G(s)}_{k^*} \parallel \text{Suffix}$

(Với Prefix là $k-1$ bit giả ngẫu nhiên đầu, Suffix là các bit ngẫu nhiên thật phía sau).*

Ta xây dựng thuật toán B để thắng Trò chơi Không thể dự đoán tại vị trí bit thứ k^* , sử dụng A làm công cụ phân biệt.

Thuật toán $B(y)$:

- Đầu vào: Tiết tố $y = G(s)_1 \dots G(s)_{k^*-1}$.
- Mục tiêu: Dự đoán $G(s)_{k^*}$.

1. B chọn một bit dự đoán ngẫu nhiên $b' \leftarrow \{0, 1\}$.
2. B chọn chuỗi hậu tố ngẫu nhiên $r' \leftarrow \{0, 1\}^{L-k^*}$.
3. B ghép thành chuỗi thử thách: $w = y \parallel b' \parallel r'$.
4. B chạy $A(w)$ và nhận kết quả $out \in \{0, 1\}$.
5. Quyết định
 - Nếu $out = 1$: B dự đoán bit tiếp theo là b' .
 - Nếu $out = 0$: B dự đoán bit tiếp theo là $1 - b'$. (Giả sử A là gán 1 cho chuỗi giả ngẫu nhiên).

Ta xét xác suất B đoán đúng bit thật $b_{true} = G(s)_{k^*}$. Chuỗi w mà B tạo ra sẽ rơi vào một trong hai phân bố dựa trên giá trị của b' :



- Nếu $b' = b_{true}$ (xác suất 0.5):
 - w có cấu trúc giống hệt H_{k^*} (vì vị trí k^* là bit giả đúng).
 - B đoán đúng khi $A(w) = 1$.
 - Xác suất thành phần: $\Pr[A(H_{k^*}) = 1]$.
- Nếu $b' \neq b_{true}$ (xác suất 0.5):
 - Vì b' là ngẫu nhiên và khác bit giả, b' phân bố như một bit ngẫu nhiên thật. w có cấu trúc giống hệt H_{k^*-1} .
 - B đoán đúng (đảo ngược $1 - b'$) khi $A(w) = 0$.
 - Xác suất thành phần: $\Pr[A(H_{k^*-1}) = 0] = 1 - \Pr[A(H_{k^*-1}) = 1]$.

Tổng hợp lại, xác suất thắng của B là:

$$\Pr[B \text{ thắng}] = \frac{1}{2} \Pr[A(H_{k^*}) = 1] + \frac{1}{2} (1 - \Pr[A(H_{k^*-1}) = 1])$$

Lợi thế của B (độ lệch so với 0.5):

$$\text{Adv}(B) = \left| \Pr[B \text{ thắng}] - \frac{1}{2} \right| = \frac{1}{2} |\Pr[A(H_{k^*}) = 1] - \Pr[A(H_{k^*-1}) = 1]|$$

Sử dụng kết quả từ mục (*) ta đã biết $|\Pr[A(H_{k^*}) = 1] - \Pr[A(H_{k^*-1}) = 1]| \geq \frac{\epsilon}{L}$. Thay vào phương trình trên:

$$\text{Adv}_{\text{Pred}}[B, G] \geq \frac{\epsilon}{2L} = \frac{\text{Adv}_{\text{PRG}}[A, G]}{2L}$$

Từ bất đẳng thức đã chứng minh ở trên, Chúng ta xem xét các tính chất sau:

1. Theo giả định ban đầu, G là không an toàn, nghĩa là tồn tại đối thủ A có lợi thế $\epsilon = \text{Adv}_{\text{PRG}}[A, G]$ là không đáng kể (non-negligible).
2. L là độ dài đầu ra của PRG, là một hàm đa thức (polynomial) theo tham số an toàn n .
3. Một hàm không không đáng kể chia cho một hàm đa thức thì kết quả vẫn là một hàm không không đáng kể.

⇒ Do đó, lợi thế của B ($\text{Adv}_{\text{Pred}}[B, G]$) cũng là không không đáng kể.

Điều này có nghĩa là: Nếu tồn tại một đối thủ phá vỡ tính an toàn của G , thì chắc chắn tồn tại một đối thủ B có thể dự đoán bit tiếp theo của G với xác suất tốt hơn đáng kể so với đoán mò.

Chúng ta đã chứng minh được mệnh đề phản đảo: "Nếu G không an toàn \Rightarrow thì G có thể dự đoán (không 'không thể dự đoán')."

Do đó, ta có thể khẳng định: "Nếu G là không thể dự đoán (unpredictable) \Rightarrow thì G là an toàn (secure)." (Điều phải chứng minh)



Kết luận

Dựa trên các phân tích và chứng minh toán học đã thực hiện trong báo cáo, chúng ta có các kết quả sau:

- Chiều thuận (Câu b): Chúng ta đã chứng minh rằng nếu một PRG thỏa mãn định nghĩa an toàn mật mã (cryptographically secure), thì nó không thể bị dự đoán bit tiếp theo (next-bit unpredictable).

$$G \text{ is Secure} \implies G \text{ is Unpredictable}$$

- Chiều ngược (Câu c): Thông qua kỹ thuật đối số lai (hybrid argument) và phương pháp chứng minh phản đảo (contrapositive), chúng ta đã chứng minh rằng nếu tồn tại một đối thủ phá vỡ tính an toàn (phân biệt được chuỗi giả ngẫu nhiên), thì tồn tại một đối thủ dự đoán được bit tiếp theo. Điều này tương đương logic với việc: Nếu G là không thể dự đoán, thì G là an toàn.

$$G \text{ is Unpredictable} \implies G \text{ is Secure}$$

Vì mối quan hệ kéo theo đúng ở cả hai chiều, chúng ta đi đến kết luận cuối cùng:

Hai khái niệm "An toàn giả ngẫu nhiên" (Indistinguishability) và "Không thể dự đoán bit tiếp theo" (Next-Bit Unpredictability) là TƯƠNG ĐƯƠNG (Equivalent) đối với các bộ tạo số giả ngẫu nhiên (PRGs).



2 Attacking the Vigenere Cipher

Mật mã Vigenere Cipher là một loại mật mã được cải thiện từ một loại mật mã rất quen thuộc, đó là Caesar Cipher. Trong quá khứ, vì sự kém bảo mật của Caesar Cipher mà Vigenere Cipher đã được ra đời để bù đắp cho khoảng bảo mật đó.

2.1 Mật mã Caesar

Nói về Mật mã Caesar... thực sự nó là một trong những kỹ thuật mật mã cổ điển và đơn giản nhất, gần như là... rất đơn giản

Về cơ bản, nó là một mật mã thay thế đơn ký tự (monoalphabetic substitution). Ý tưởng cốt lõi là ta chỉ cần chọn một khóa bí mật duy nhất, đây là một con số cố định, ví dụ như 3. Sau đó... bùm... ta chỉ việc dịch chuyển (**shift**) toàn bộ bảng chữ cái đi đúng bấy nhiêu nấc. Ví dụ, với khóa **3** thì chữ A sẽ thành chữ D. Tuy nhiên, nó có 1 điểm yếu chí mạng, đó chính là tuy thay đổi được mặt chữ nhưng không thể thay đổi tần suất xuất hiện thông thường của từ tiếng Anh. Trong tiếng Anh, chữ E có **tần suất xuất hiện** nhiều hơn so với các từ khác, nên nó hoàn toàn bị nhận ra được. Ví dụ là từ **HELLO** nếu bị dời với độ dài là 3 thì sẽ thành từ **KHOOR**, rõ ràng thấy 2 chữ O xuất hiện liền kề và ta hoàn toàn có thể sử dụng thuật toán phân tích tần suất để đoán ra được chữ O này chính là chữ E. Từ đó ta đoán ra KEY là 3

2.2 Mật mã Vigenere

Nhận thức rõ được điểm yếu chí mạng của mật mã Caesar, Mật mã Vigenere được phát triển như một giải pháp tinh vi hơn, đại diện cho một bước nhảy vọt từ thay thế đơn ký tự sang **thay thế đa ký tự**.

Cơ chế hoạt động của Vigenere không còn dựa vào một phép dịch chuyển *cố định* duy nhất. Thay vào đó, nó sử dụng một **từ khóa (keyword)**, ví dụ "KEY". Độ dài của từ khóa này (ví dụ: 3) quyết định số lượng bảng chữ cái Caesar khác nhau sẽ được sử dụng một cách luân phiên. Quá trình mã hóa được thực hiện như sau:

- **Mở rộng khóa (Key Expansion):** Từ khóa được lặp lại để có độ dài tương ứng với bản rõ (plaintext).
- **Mã hóa (Encryption):** Mỗi ký tự của bản rõ được dịch chuyển (shift) bằng một số khác nhau, số này được xác định bởi ký tự tương ứng của khóa đã mở rộng (thường A=0, B=1, ..., K=10, ...).

Hãy xem xét ví dụ mã hóa bản rõ **ATTACKATDAWN** với từ khóa **KEY**:

Plaintext: T H E Q U I C K B R O W N F O X



Keyword: K E Y K E Y K E Y K E Y K E Y K

Ciphertext: D L G Z Y M K K K V S A D R X

Trong ví dụ này, cùng một chữ cái bản rõ (ví dụ: chữ **O** trong "BROWN" và "FOX") sẽ được mã hóa bằng các chữ cái khóa khác nhau (**E** và **Y**), do đó tạo ra các chữ cái bản mã khác nhau.

Chính cơ chế "đa dịch chuyển" (multiple shifts) này đã **phá vỡ** hoàn toàn phương pháp phân tích tần suất đơn giản. Tần suất xuất hiện của ngôn ngữ gốc (hay được gọi là tần suất xuất hiện ngôn ngữ) bị làm phẳng (flattened), nên ta không thể xác định được đồi núi từ vựng, không thể xác định tần suất từ vựng một cách hiệu quả. Kẻ tấn công không thể nào nhìn vào bản mã và nói "chữ K xuất hiện nhiều nhất, nó chắc chắn là E" được nữa, vì chữ E trong bản rõ đã bị mã hóa thành nhiều ký tự khác nhau (D, G, M, V...).

Đây chính là sự cải tiến bảo mật mang tính cách mạng của Vigenere so với Caesar.

2.3 Ý tưởng khai thác

Dè bài yêu cầu ta phải khai thác, tấn công được loại mật mã học này, nên trước hết ta nên nghiên cứu về cách khả thi để khai thác.

Mặc dù Vigenere là một cải tiến vượt bậc, nó không phải là không thể bị bẻ gãy. Cuộc tấn công vào Vigenere là một trong những thành tựu vĩ đại của mật mã học thế kỷ 19. Nghịch lý nằm ở chỗ: Vigenere *che giấu* tần suất xuất hiện tần suất khi ta phân tích *toàn bộ* bản mã, nhưng nó lại *lộ* chính tần suất xuất hiện đó nếu ta có thể phân tách bản mã một cách chính xác.

Cuộc tấn công bao gồm hai bước chính:

- Xác định Độ dài Khóa (L):** Tìm ra "nhịp điệu" lặp lại của từ khóa.
- Xác định Khóa:** Khi đã biết độ dài L , chia bản mã thành L nhóm và tấn công từng nhóm như một mật mã Caesar đơn lẻ.

Bước 1: Xác định Độ dài Khóa (L)

Đây là bước đột phá quan trọng nhất. Chúng ta phải tìm ra độ dài L của từ khóa (ví dụ: "KEY" có $L = 3$) mà không hề biết bản thân từ khóa đó. Có hai phương pháp chính:

Phép thử Kasiski (Kasiski Examination)

Phương pháp này dựa trên một quan sát thông minh về **sự trùng hợp ngẫu nhiên**. Ta có thể tưởng tượng từ khóa là một từ 3 ký tự KEY. Người mã hóa sẽ đóng dấu K-E-Y-K-E-Y-K-E-Y... lặp lại liên tục trên bản rõ.

Bây giờ, hãy giả sử trong bản rõ có hai từ THE xuất hiện:



- Từ THE đầu tiên (ở vị trí 10) tình cờ được đóng dấu bởi K-E-Y. Kết quả mã hóa là DLG.
- Từ THE thứ hai (ở vị trí 70) cũng tình cờ được đóng dấu bởi đúng K-E-Y. Kết quả mã hóa cũng là DLG.

Kẻ tấn công không thấy bản rõ, nhưng sẽ thấy chuỗi DLG xuất hiện lặp lại ở vị trí 10 và 70. Họ sẽ suy luận: "Khoảng cách giữa chúng là $70 - 10 = 60$. Rất có thể 60 là một **bội số** của độ dài con dấu (độ dài khóa)."

Bằng cách tìm *nhiều* chuỗi lặp lại như vậy (ví dụ: ABC lặp lại cách nhau 45, XYZ lặp lại cách nhau 75), kẻ tấn công sẽ ghi lại tất cả các khoảng cách (60, 45, 75) và tìm **Ước chung Lớn nhất (GCD)** của chúng.

$$GCD(60, 45, 75) = 15$$

Điều này cho ta một "ứng cử viên" rất mạnh: độ dài khóa L có thể là 3, 5, hoặc 15.

Chỉ số Trùng hợp (Index of Coincidence - IC)

Đây là phương pháp thuần toán học hơn, chính là việc ứng dụng xác suất và "tần suất xuất hiện (fingerprint). Nó đo lường độ gồ ghề của tần suất văn bản. Thông thường, có quy luật như sau:

- **Tiếng Anh chuẩn:** Có tần suất xuất hiện rất gồ ghề (E, T, A nhiều). $IC \approx 0.067$.
- **Văn bản ngẫu nhiên:** Có tần suất xuất hiện phẳng (mọi chữ bằng nhau). $IC \approx 0.038$ (hay $1/26$).

Bản mã Vigenere (nếu nhìn tổng thể) sẽ có IC thấp ≈ 0.038 vì tần suất xuất hiện đã bị san phẳng.

Tuy nhiên, đây là mấu chốt:

- Nếu ta **đoán sai** độ dài khóa (ví dụ: $L = 5$, trong khi khóa thật là 3) và chia bản mã thành 5 nhóm, mỗi nhóm vẫn là một mớ lộn xộn. IC trung bình của chúng sẽ ≈ 0.038 .
- Nếu ta **đoán đúng** độ dài khóa (ví dụ: $L = 3$) và chia bản mã thành 3 nhóm (nhóm 1 gồm các chữ 1, 4, 7...; nhóm 2 gồm các chữ 2, 5, 8...; v.v.), thì **mỗi nhóm** này là một mật mã Caesar! tần suất xuất hiện của nó vẫn gồ ghề y hệt tiếng Anh, chỉ bị dịch chuyển đi.

Do đó, khi ta đo IC của từng nhóm này, chúng sẽ cho kết quả **cao** ≈ 0.067 .

Thuật toán: Ta chạy một vòng lặp, thử đoán L từ 2 đến 20, tức là giả sử độ dài khóa từ khoảng 2 đến 20. Với mỗi L , ta chia bản mã thành L nhóm, tính IC trung bình của các nhóm đó. Giá trị L nào cho ra IC trung bình **cao nhất** (gần 0.067 nhất) chính là độ dài khóa L mà chúng ta cần tìm.



Lý do lựa chọn Chỉ số Trùng hợp (IC) cho bài tập này: Trong khi Phép thử Kasiski rất thông minh, nó phụ thuộc rất nhiều vào *sự may mắn* – cụ thể là sự trùng hợp ngẫu nhiên của (1) từ lặp lại trong bản rõ và (2) từ đó phải thảng hàng với cùng một đoạn khóa. Nhóm chúng em cũng đã thử sử dụng Phép thử Kasiski trong bài toán này nhưng không đoán ra được giá trị *KEY*. Với một bản mã dài và phức tạp như trong bài tập thử thách, phương pháp IC trở nên **vượt trội** vì nó đáng tin cậy hơn, ít bị ảnh hưởng bởi sự ngẫu nhiên, và cho ta một kết quả rõ ràng dựa trên cơ sở toán học vững chắc, thay vì dựa vào sự trùng hợp may rủi.

Bước 2: Tìm Khóa (Áp dụng tần suất xuất hiện)

Khi đã có L , ta hoàn toàn tìm được ra khóa. Ta biết rằng bản mã được chia thành L nhóm, và mỗi nhóm là một mật mã Caesar độc lập.

- (a) **Ta tiến hành tách nhóm (Splitting):** Chia bản mã thành L nhóm con (subgroups) một cách rõ ràng.
 - Nhóm 1: Gồm các ký tự ở vị trí $1, 1 + L, 1 + 2L, \dots$
 - Nhóm 2: Gồm các ký tự ở vị trí $2, 2 + L, 2 + 2L, \dots$
 - ...
 - Nhóm L : Gồm các ký tự ở vị trí $L, L + L, L + 2L, \dots$
- (b) **Tấn công từng nhóm (Frequency Attack):** Ta tấn công Nhóm 1 như một mật mã Caesar. Ta có thể tìm "ngọn núi E" như đã nói, nhưng một phương pháp thống kê hiệu quả hơn là **Phép thử Chi-squared (χ^2)**.
- (c) **Phép thử χ^2 :** Ta thử "dịch lùi" (decrypt) Nhóm 1 với tất cả 26 khả năng (shift = 0 đến 25). Với mỗi lần thử, ta so sánh tần suất xuất hiện (phân bố tần suất) của kết quả thu được với tần suất xuất hiện chuẩn của tiếng Anh. Phép dịch lùi nào cho ra kết quả *khớp nhất* (có "độ vênh" hay χ^2 thấp nhất) sẽ tiết lộ chữ cái khóa.
- (d) **Ví dụ:** Nếu thử dịch lùi Nhóm 1 bằng 'G' (shift=6) cho kết quả giống tiếng Anh nhất, thì ký tự khóa đầu tiên là **G**.
- (e) **Tổng hợp (Combine):** Lặp lại quá trình này cho L nhóm, ta sẽ thu được L ký tự khóa, và đó chính là từ khóa Vigenere hoàn chỉnh.

2.4 Bài toán thực tế

Đề bài cho ta 1 file sample_encrypt.py để ta thử mã hóa thử, và 1 file ciphertext2.txt để ta decrypt nó, ta tiến hành phân tích file sample_encrypt.py

```
1 def expand_key(self, text_length: int) -> str:  
2     key = self.key  
3     idx = 0  
4     while True:
```



```
5     if text_length > len(key):
6         key += key[idx]
7         idx += 1
8     else:
9         break
10    return key
```

Phân tích hàm cụ thể:

- Hàm này nhận độ dài của bản rõ (`text_length`) làm đầu vào.
- Nó khởi tạo một biến `key` bằng với từ khóa gốc (`self.key`).
- Vòng lặp `while` sẽ chạy miễn là `key` còn ngắn hơn `text_length`.
- Bên trong vòng lặp, hai dòng code `key += key[idx]` và `idx += 1` là quan trọng để cho độ dài khóa có độ dài bằng từ vựng từ plaintext.
- **Hoạt động:** Giả sử khóa là "KEY" (dài 3).
 - Lần 1: `key = "KEYK"` (thêm `key[0]`), `idx = 1`.
 - Lần 2: `key = "KEYKE"` (thêm `key[1]`), `idx = 2`.
 - Lần 3: `key = "KEYKEY"` (thêm `key[2]`), `idx = 3`.
 - Lần 4: `key = "KEYKEYK"` (thêm `key[3]`, là 'K'), `idx = 4`.
- **Kết luận:** Ta thấy rõ độ dài khóa sẽ được mở rộng sao cho ngang với độ dài plaintext.

Phân tích hàm `encrypt` Đây là hàm thực hiện logic mã hóa.

```
1 def encrypt(self, plaintext: str) -> str:
2     key = self.expand_key(len(plaintext))
3     ciphertext = ""
4     idx = 0
5     for char in plaintext:
6         if char in self.alphabet:
7             shift = self.alphabet.index(key[idx])
8             ciphertext += self.alphabet[(self.alphabet.index(char) + shift) %
26]
9             idx += 1
10        else:
11            if char in ALPHABET_UPPER:
12                shift = self.alphabet.index(key[idx])
13                ciphertext += self.alphabet[
14                    (self.alphabet.index(char.lower()) + shift) % 26
15                ].upper()
16                idx += 1
17            else:
18                ciphertext += char
19    return ciphertext
```



Phân tích:

- Hàm nhận bản rõ (plaintext) và lấy về khóa đã mở rộng (key).
- Nó sử dụng một biến idx riêng biệt để theo dõi vị trí trên key.
- **Logic chính (dòng 10-12):** Nếu ký tự là chữ thường, nó thực hiện chính xác phép toán Vigenere chuẩn: $C = (P + K) \pmod{26}$. Quan trọng nhất, nó idx += 1 để di chuyển đến ký tự khóa tiếp theo.
- **Xử lý chữ hoa (dòng 15-18):** Logic tương tự được áp dụng, chỉ khác là nó bảo toàn việc viết hoa. Nó cũng idx += 1.
- **Chi tiết của dòng else (dòng 20):** Nếu ký tự là bất cứ thứ gì khác (khoảng trống, dấu chấm, số...), nó được thêm thẳng vào bản mã.
- **Kết luận:** Sau khi đưa input và key vào hàm này, đầu ra ta sẽ có được bản ciphertext.

Cùng nhau chạy thử đối với một câu đơn giản, với khóa là "undertale" và ta có ciphertext là "his theme will stay here forever.>".

Hình 2: Kết quả ciphertext khi mã hóa thử

Được rồi, sau khi đã chạy thử và hiểu được cách thức hoạt động của Vigenere Cipher, cùng nhau viết script để tấn công dựa trên những gì đã nghiên cứu trên.

Khởi tạo và Cơ sở dữ liệu Trước tiên, ta cần import các thư viện và định nghĩa tần suất xuất hiện tiếng Anh. Dựa trên tài liệu trên mạng, ta có bảng tần suất như sau:

```
1 import math
2 from collections import Counter
3
4
5 ENGLISH_FREQUENCIES = { ... } # 'a': 0.08167, ...
6 ALPHABET = "abcdefghijklmnopqrstuvwxyz"
7 IC_ENGLISH = 0.067
8 IC_RANDOM = 0.038
```

- `from collections import Counter`: Thư viện dùng để giúp ta đếm các ký tự nhanh hơn
- `ENGLISH_FREQUENCIES`: Đây là tần suất tiếng Anh được lấy từ bảng trên. Toàn bộ Bước 2 (Phép thử χ^2) đều dựa trên việc so sánh với cơ sở dữ liệu này.



Table 1.1 Relative letter frequencies of the English language

Letter	Frequency	Letter	Frequency
A	0.0817	N	0.0675
B	0.0150	O	0.0751
C	0.0278	P	0.0193
D	0.0425	Q	0.0010
E	0.1270	R	0.0599
F	0.0223	S	0.0633
G	0.0202	T	0.0906
H	0.0609	U	0.0276
I	0.0697	V	0.0098
J	0.0015	W	0.0236
K	0.0077	X	0.0015
L	0.0403	Y	0.0197
M	0.0241	Z	0.0007

Hình 3: Bảng tần suất xuất hiện của ký tự tiếng Anh

- IC_ENGLISH: Đây là điểm chuẩn cho Bước 1. Chúng ta tìm độ dài khóa L nào cho ra IC trung bình gần với con số này nhất.

Hàm clean_text Dựa trên phân tích encrypt (bỏ qua dấu câu), ta cần một hàm "dọn dẹp" để chuẩn bị dữ liệu cho phân tích thống kê.

```
1 def clean_text(text: str) -> str:  
2     return "".join(filter(str.isalpha, text)).lower()
```

- **Tại sao:** Hàm encrypt chỉ tăng idx của khóa khi gặp chữ cái. Điều này có nghĩa là ciphertext[0] được mã hóa bởi key[0], nhưng ciphertext[5] (nếu có 2 dấu cách) có thể mới được mã hóa bởi key[3].
- Nếu chúng ta chia nhóm (splitting) mà không dọn dẹp, các nhóm sẽ bị *lệch* (misaligned) so với khóa.
- Bằng cách loại bỏ mọi thứ không phải chữ cái, ta đảm bảo cleaned_text[0] khớp key[0], cleaned_text[1] khớp key[1], v.v.

Hàm calculate_ic Đây là hàm để tính IC

```
1 def calculate_ic(text: str) -> float:  
2     N = len(text)  
3     if N <= 1: return 0.0  
4     counts = Counter(text)  
5     numerator = 0.0
```



```
6     for char in ALPHABET:
7         count = counts.get(char, 0)
8         numerator += count * (count - 1)
9     denominator = N * (N - 1)
10    return numerator / denominator if denominator > 0 else 0.0
```

- **Phân tích:** Hàm này nô đang triển khai định nghĩa xác suất của Chỉ số Trùng hợp (IC), tức là: "Xác suất để chọn ngẫu nhiên hai ký tự trong văn bản và chúng giống hệt nhau."
 - **Mẫu số (Denominator):** $\text{denominator} = N * (N - 1)$. Với một văn bản dài N , tổng số cách để chọn 2 ký tự (có thứ tự, ví dụ: (ký tự 1, ký tự 5) và (ký tự 5, ký tự 1)) là $P(N, 2) = N \times (N - 1)$.
 - **Tử số (Numerator):** $\text{numerator} += \text{count} * (\text{count} - 1)$. Đây là tổng số cách để chọn 2 ký tự *giống hệt nhau*. Ví dụ: Nếu văn bản có 5 chữ 'A' ($\text{count} = 5$), số cách chọn 2 chữ 'A' (có thứ tự) là $P(5, 2) = 5 \times 4 = 20$. Vòng lặp `for char in ALPHABET` sẽ tính tổng tất cả các khả năng này (số cách chọn 2 chữ 'A', cộng số cách chọn 2 chữ 'B', v.v.).
 - **Kết luận:** Phép chia $\text{numerator} / \text{denominator}$ chính xác là $\frac{\sum_{i=1}^Z P(f_i, 2)}{P(N, 2)}$ (với f_i là tần suất của chữ i), đây chính là công thức chuẩn để tính xác suất trùng hợp (IC). Hàm này trả về một con số duy nhất để cho hàm `find_key_length_ic` biết được "phép đoán" của nó đúng hay sai (gần 0.067 hay 0.038).

Hàm `find_key_length_ic` (Bước 1) Đây là "Nhà khoa học" tự động, thực hiện lần chạy tìm độ dài khóa L .

```
1 def find_key_length_ic(ciphertext: str, max_key_len=20) -> int:
2     cleaned_text = clean_text(ciphertext)
3     key_lens_sorted = []
4     for key_length in range(2, max_key_len + 1):
5         total_ic = 0.0
6         for i in range(key_length):
7             subgroup = cleaned_text[i::key_length]
8             total_ic += calculate_ic(subgroup)
9
10        average_ic = total_ic / key_length
11        key_lens_sorted.append((key_length, average_ic, abs(average_ic -
12 IC_ENGLISH)))
13
14    key_lens_sorted.sort(key=lambda x: x[2])
15    best_key_len = key_lens_sorted[0][0]
16    return best_key_len
```

- **Tại sao:** Đây là sự tự động hóa của Bước 1. Thay vì làm thủ công, nó lặp qua tất cả các độ dài L khả thi.



- **Cơ chế:** Dòng `subgroup = cleaned_text[i::key_length]` chính là hành động "chia xô" (sàng lọc) mà ta đã thảo luận.
- **Phân tích average_ic:** Đây là logic quan trọng nhất. Ta phải trả lời: "Tại sao lại lấy $\text{total_ic} / \text{key_length}$?"
 - `total_ic` là *tổng điểm* "gồ ghè" của tất cả các "xô" cộng lại.
 - **Ví dụ:** Giả sử ta đoán $L = 3$ (đoán đúng). Ta sẽ có 3 "xô", mỗi xô đều "gồ ghè" ($IC \approx 0.067$). Vậy, $\text{total_ic} \approx 3 \times 0.067 = 0.201$.
 - **Ví dụ:** Giả sử ta đoán $L = 5$ (đoán sai). Ta sẽ có 5 "xô", mỗi xô đều "phẳng" ($IC \approx 0.038$). Vậy, $\text{total_ic} \approx 5 \times 0.038 = 0.190$.
 - Vấn đề là: chỉ so sánh hai giá trị `total_ic` (0.201 và 0.190) là rất rủi ro, không rõ ràng và bị sai lệch bởi số lượng "xô".
 - Phép chia `average_ic = total_ic / key_length` là một bước **chuẩn hóa (normalization)**. Nó thay đổi câu hỏi từ "Tổng điểm là bao nhiêu?" (vốn phụ thuộc L) thành "Điểm trung bình *cho mỗi xô* là bao nhiêu?" (độc lập với L).
 - Với phép chia này, kết quả trở nên cực kỳ rõ ràng:
 - * Đoán đúng ($L = 3$): $\text{average_ic} \approx (3 \times 0.067)/3 = 0.067$ (gần chuẩn).
 - * Đoán sai ($L = 5$): $\text{average_ic} \approx (5 \times 0.038)/5 = 0.038$ (gần rác).
- **Quyết định:** Dòng `key_lens_sorted.sort(key=lambda x: x[2])` là bộ não. Nó so sánh tất cả các "điểm trung bình" (các giá trị `average_ic` đã được chuẩn hóa) và chọn ra Lần chạy nào (giá trị L nào) có kết quả gần với `IC_ENGLISH` (0.067) nhất.

Hàm find_best_caesar_shift (Bước 2) Đây là Hàm phá mã Caesar, công cụ cốt lõi của Bước 2.

```
1 def find_best_caesar_shift(subgroup_text: str) -> str:
2     best_shift = 0
3     min_chi_squared = float('inf')
4     text_len = len(subgroup_text)
5
6     for shift in range(26):
7         chi_squared_sum = 0.0
8
9         shifted_counts = Counter()
10        for char in subgroup_text:
11            shifted_index = (ALPHABET.index(char) - shift) % 26
12            shifted_counts[ALPHABET[shifted_index]] += 1
13
14        for i in range(26):
15            char = ALPHABET[i]
16            expected_count = ENGLISH_FREQUENCIES[char] * text_len
17            chi_squared_sum += ((shifted_counts[char] - expected_count) ** 2) / expected_count
18
19    if chi_squared_sum < min_chi_squared:
20        min_chi_squared = chi_squared_sum
21        best_shift = shift
22
23    return best_shift
```



```
19         observed_count = shifted_counts.get(char, 0)
20         if expected_count == 0: continue
21         difference = observed_count - expected_count
22         # Calculate (Chi-squared)
23         chi_squared_sum += (difference * difference) / expected_count
24
25
26         if chi_squared_sum < min_chi_squared:
27             min_chi_squared = chi_squared_sum
28             best_shift = shift
29
30     return ALPHABET[best_shift]
```

- **Tại sao:** Hàm này được gọi L lần (một lần cho mỗi "xô"). Nó phải trả lời: "Chìa khóa Caesar (ký tự) nào đã mã hóa cái xô này?"
- **Cơ chế & Quyết định (Chi-squared):** Hàm này tiến hành bruteforce 26 shifts (từ 'shift = 0' đến 'shift = 25') và so sánh kết quả.

Hãy tưởng tượng nó nhận được một "xô" (subgroup) là "XGXA" (đây là Nhóm 2 từ ví dụ ATTACKATDAWN của chúng ta, với khóa thật là 'E' (shift=4)).

(a) **Lần chạy 1 (shift=0, 'A'):**

- *"Giả thuyết: Khóa là 'A'."*
- Dịch lùi "XGXA" với shift 0 → "XGXA".
- **Đo "độ vênh" χ^2 :** So sánh "dấu vân tay" của "XGXA" (Tần suất: X=50%, G=25%, A=25%) với tiếng Anh chuẩn (E=12.7%, T=9.1%...).
- **Kết quả:** "Độ vênh" RẤT CAO (ví dụ: $\chi^2 = 500.0$).

(b) **Lần chạy 2 (shift=1, 'B'):**

- *"Giả thuyết: Khóa là 'B'."*
- Dịch lùi "XGXA" với shift 1 → "WFWA".
- **Đo "độ vênh" χ^2 :** So sánh "dấu vân tay" của "WFWA" (W=50%, F=25%, A=25%) với tiếng Anh chuẩn.
- **Kết quả:** "Độ vênh" RẤT CAO (ví dụ: $\chi^2 = 480.0$).

(c) ... (Thử các shift 2, 3) ...

(d) **Lần chạy 5 (shift=4, 'E'):**

- *"Giả thuyết: Khóa là 'E'."*
- Dịch lùi "XGXA" với shift 4 → "TCTW".
- **Đo "độ vênh" χ^2 :** So sánh "dấu vân tay" của "TCTW" (T=50%, C=25%, W=25%) với tiếng Anh chuẩn.
- **Phân tích:** Trong tiếng Anh, T là chữ phổ biến thứ 2 (9.1%), C và W cũng khá phổ biến. "Dấu vân tay" này, mặc dù không hoàn hảo (vì mẫu quá ngắn), nhưng trông hợp lý hơn nhiều so với "XGXA" hay "WFWA".



- **Kết quả:** Phép toán (thực tế - kỳ vọng) / kỳ vọng cho ra "độ vênh" RẤT THẤP (ví dụ: $\chi^2 = 12.5$).
- (e) ... (Tiếp tục thử các shift 5 đến 25) ...
- (f) **Lần chạy 21 (shift=20, 'U'):**
 - "Giả thuyết: Khóa là 'U'."
 - Dịch lùi "XGXA" với shift 20 → "DQDP".
 - **Đo "độ vênh" χ^2 :** So sánh "DQDP" (D=50%, Q=25%, P=25%) với tiếng Anh chuẩn (trong đó 'Q' siêu hiếm 0.1%).
 - **Kết quả:** "Độ vênh" CỰC KỲ CAO (ví dụ: $\chi^2 = 1000.0$).
- **Quyết định cuối cùng:** Sau khi chạy 26 Lần chạy, hàm so sánh 26 giá trị "độ vênh" (500.0, 480.0, ..., 12.5, ..., 1000.0). Nó thấy rằng 12.5 là giá trị *nhỏ nhất* (`min_chi_squared`). Giá trị này tương ứng với `best_shift = 4`.
- **Trả về:** Hàm trả về `ALPHABET[4]`, chính là ký tự khóa 'e'.

Sau khi tìm được khóa, việc decrypt là vô cùng đơn giản, ta thay nhìn logic encrypt là + shifts, decrypt ta sẽ tiến hành - shifts, hoàn tất. Giờ hãy tiến hành chạy thử nghiệm nào

2.5 Chạy thử nghiệm

```
PS D:\Study\Cryptography\BTL> & C:/Users/Phat/AppData/Local/Programs/Python/Python313/python.exe d:/Study/Cryptography/BTL/attack_tools.py
--- Thử độ dài khóa (Phương pháp IC) ---
Độ dài 2: IC trung bình = 0.04517
Độ dài 3: IC trung bình = 0.04292
Độ dài 4: IC trung bình = 0.04512
Độ dài 5: IC trung bình = 0.05190
Độ dài 6: IC trung bình = 0.04511
Độ dài 7: IC trung bình = 0.04275
Độ dài 8: IC trung bình = 0.04488
Độ dài 9: IC trung bình = 0.04255
Độ dài 10: IC trung bình = 0.06756
Độ dài 11: IC trung bình = 0.04268
Độ dài 12: IC trung bình = 0.04491
Độ dài 13: IC trung bình = 0.04236
Độ dài 14: IC trung bình = 0.04474
Độ dài 15: IC trung bình = 0.05212
Độ dài 16: IC trung bình = 0.04419
Độ dài 17: IC trung bình = 0.04253
Độ dài 18: IC trung bình = 0.04425
Độ dài 19: IC trung bình = 0.04281
Độ dài 20: IC trung bình = 0.06797
Độ dài 21: IC trung bình = 0.04218
Độ dài 22: IC trung bình = 0.04467
Độ dài 23: IC trung bình = 0.04386
Độ dài 24: IC trung bình = 0.04453
Độ dài 25: IC trung bình = 0.05128

--- Xếp hạng độ dài khóa tiềm năng ---
Hạng 24: Độ dài = 10 (IC: 0.06756, cách IC chuẩn: 0.00056)
Hạng 23: Độ dài = 20 (IC: 0.06797, cách IC chuẩn: 0.00097)
Hạng 22: Độ dài = 15 (IC: 0.05212, cách IC chuẩn: 0.01488)
Hạng 21: Độ dài = 5 (IC: 0.05190, cách IC chuẩn: 0.01510)
Hạng 20: Độ dài = 25 (IC: 0.05128, cách IC chuẩn: 0.01572)

==> Độ dài khóa tiềm năng nhất là: 10
```

Hình 4: Kết quả việc tìm độ dài khóa



```
--- Bước 2: Tìm Khóa (Phân tích Tần suất) ---
Nhóm 1/10: Chữ cái khóa tìm được = 'h'
Nhóm 2/10: Chữ cái khóa tìm được = 'e'
Nhóm 3/10: Chữ cái khóa tìm được = 'a'
Nhóm 4/10: Chữ cái khóa tìm được = 'v'
Nhóm 5/10: Chữ cái khóa tìm được = 'e'
Nhóm 6/10: Chữ cái khóa tìm được = 'n'
Nhóm 7/10: Chữ cái khóa tìm được = 'h'
Nhóm 8/10: Chữ cái khóa tìm được = 'e'
Nhóm 9/10: Chữ cái khóa tìm được = 'l'
Nhóm 10/10: Chữ cái khóa tìm được = 'l'

==> Khóa bí mật tìm được (dự đoán): 'heavenhell'
```

Hình 5: Kết quả việc tìm khóa

Sau khi thấy độ dài khóa là 10, tiến hành chạy hàm để chia nhỏ ra thành 10 nhóm, tiến hành brute-force từng ký tự để tìm ra từ giống tiếng Anh nhất, và sau đó, ta dùng hàm decrypt kết hợp với khóa tìm được để giải lại mã như hình dưới đây

The concept of the afterlife has fascinated humanity for millennia, shaping cultures, religions, and philosophies across civilizations. At its core, the afterlife refers to the belief that some form of existence continues after physical death—a realm or state where the soul, spirit, or consciousness transcends the limitations of the mortal body. Throughout history, human beings have sought answers to the profound questions of what happens after death, and these reflections have given rise to a rich diversity of interpretations. From ancient mythologies to modern spiritual movements, the afterlife remains a central theme in understanding the nature of existence, morality, and the ultimate destiny of the human soul. In many ancient civilizations, the afterlife was viewed as a continuation of earthly life, governed by divine or cosmic justice. The ancient Egyptians, for example, believed in the eternal journey of the soul through the underworld, where the heart of the deceased was weighed against the feather of Ma'at—the goddess of truth and justice. A pure heart would enter the blissful Field of Reeds, while a corrupt one would face annihilation. Similarly, in Greek mythology, souls were judged by the gods of the underworld and sent to one of three realms: the Elysian Fields for the virtuous, the Asphodel Meadows for ordinary souls, and Tartarus for the wicked. These beliefs reflected not only an attempt to explain death but also to promote moral conduct in life, reinforcing the idea that actions had consequences beyond the grave. In contrast, Eastern philosophies such as Hinduism and Buddhism view the afterlife through the lens of reincarnation—a cyclical process where the soul is reborn in

Hình 6: Kết quả đáp án

Và khóa ở đây chính là **heavenhell**, tiến hành decrypt ta ra được 1 đoạn văn liên quan gì đó đến thế giới bên kia (isekai?).

Sau khi xong, nhóm bạn em có thử đưa đoạn văn trên vào lại hàm Encrypt sau đó đối chiếu với ciphertext2.txt được gửi và nhận thấy kết quả trùng khớp. Kết luận **key** là chính xác.



3 SMC Exploitation

3.1 Interception & API Analysis (Use Burp Suite)

Bảng tổng hợp các API endpoints

API Endpoint	Method	Chức năng	Code Location
/session/create	POST	Tạo phiên ECDH	LoginActivity.java
/session/exchange	POST	Trao đổi khóa	LoginActivity.java
/message/send	POST	Gửi tin nhắn mã hóa	ChatActivity.java

Bảng 1: API Endpoints

3.1.1 POST /session/create?userId=group-1

Phân Request

```
Request
Pretty Raw Hex
1 POST /session/create?userId=group-1 HTTP/1.1
2 Host: crypto-assignment.dangduongminhnhat2003.workers.dev
3 X-User-Id: group-1
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 380
6 Accept-Encoding: gzip, deflate, br
7 User-Agent: okhttp/4.11.0
8 Connection: keep-alive
9
10 {
    "algorithm": "ecdh",
    "curveParameters": {
        "p": "6277101735386680763835789423207666416083908700390324961279",
        "a": "-3",
        "b": "245515546008943817740293915197451784769108058161191238065",
        "Gx": "3289624317623424368845348028842487418520868978772050262753",
        "Gy": "5673242899673324591834582889556471730778853907191064256384",
        "order": "6277101735386680763835789423176059013767194773182842284081"
    }
}
```

Hình 7: Request create session

userId=group-1: Định danh người dùng/nhóm

X-User-Id: group-1: Header tùy chỉnh chứa ID người dùng

Content-Type: application/json: Định dạng dữ liệu gửi lên

Content-Length: 380: Kích thước body



Body Parameters:

```
{
  "algorithm": "ecdh",           // Thuật toán trao đổi khóa (Elliptic Curve Diffie-Hellman)
  "curveParameters": {           // Tham số đường cong elliptic
    "p": "62771017353866...",   // Số nguyên tố định nghĩa trường hữu hạn
    "a": "-3",                  // Hệ số a trong phương trình đường cong
    "b": "24551555460089...",   // Hệ số b trong phương trình đường cong
    "Gx": "32896243176234...",   // Tọa độ x của điểm cơ sở (generator)
    "Gy": "56732428996733...",   // Tọa độ y của điểm cơ sở
    "order": "62771017353866..." // Bậc của điểm cơ sở (số điểm trên đường cong)
  }
}
```

Phản響 (Response)

Response

Pretty	Raw	Hex	Render
HTTP/2 200 OK			
2 Date: Fri, 07 Nov 2025 11:41:27 GMT			
3 Content-Type: application/json			
4 Access-Control-Allow-Origin: *			
5 Access-Control-Allow-Headers: Content-Type, X-Session-Token			
6 Access-Control-Allow-Methods: GET, POST, OPTIONS			
7 Vary: accept-encoding			
8 Report-To:			
9 {"group": "cf-neel", "max_age": 604800, "endpoints": [{"url": "https://a.neel.cloudflare.com/report/v4?s=1JPmGUxFtCBiLui3dFtFaajsvrAkyKUvvVFf9nKlNrmukYhHgJuPWWSeJskpMs09bG6Tb23gnbYqiuJ549EUBvCn2D01XtTwCFerI8NLPlxqzb8WBpSMGHdsqGsJru4eDVIXim0QbdBzH0est3D"}]}			
10 Neel: {"report_to": "cf-neel", "success_fraction": 0.0, "max_age": 604800}			
11 Server: cloudflare			
12 Cf-Ray: 99ac8fc709cb9c15-SIN			
13 Alt-Svc: h3="443"; ma=86400			
14 {			
"success": true,			
"sessionToken":			
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 eyJpc3MiOiJtZWNlcwDaGF0iioxNzYvNTER1njg3LCJ1eHAiOiJk3NjI1MTU5DcsInN1YiI6Imdyb3VwLTER1LCJzaWQ1oiI4OD1ONDQ4YCMODMsNDcw0TgC0TEzqzgS1NDU2CYEcYsE3MDU2OTBzMDQCNtL1NDVhMsYx0TBh1iYWhnb3JpdGhtIjoiZWNaCaIsInBlIaixpY0tlesI6eyi4IjoiMjYzNjQsNDYjNjMINTA1ndg1NzscMsI3NjMSMsIwNTh0Tg3Nj1lOTAOmsYZNTMwNjM40TA1NtVSh1sInhi0i1wNyUONDUsGNTQz0Dgkjh4NTU5HTcaMsIxNsypMjQCOceMDE2EMT1wNT140DA0NsQ40TQCNQ5NTc4IndsImVuY3j5cHL1ZERhdGk1o1jNzNf22ExNx1pMjQWJC0XFEVlyWnM020dr1lnFeuoxbhp1VWVpWKL5MhHuUghtfH0Rde1pPX0FQfUvN1SyamRq2E5HHDuNWWfCrTjAxOFiTFFrM3shEvNHDmstQjVFNFpBxLYdRhTUHY2aW1Z2HA2SGc23pD5FNVSVAnchQwW5yjWENNTXA3ZE1UHEhoYmNS10NjU1dhblRwpsQ2NOMwpxp0E1WVGNRbTRNybRpRdnZicQOHMh51Up2bbjkJZ0twW1ZJSXVvUHHVY25EbbWnC5jg3YWwvWnZxdFd00UhewNbjnBzL18xY1NQQUVfebxZw94VzJtbEMC50bhUeh1Zh53ZGZTHFQ1VFM5WVpVwkh8Q18sTGJyV7td4Whb3VoTndSm1dIngsQmHhEyY04yS1FCbUlaZHR02E5TxmNHchPheEVcdGZkdlI4ajhscaFBaFPmV3JUV18yYhRleEZIT0wYmEzcmlQZnBXUEUREtCUGJGUatFMaxSamhaOZ0el1H8Um9acdrh1BpJd0f1ZwHMPZBnCwvHVRdM01ekh1WWZTphLVEsZ1nZo2ZdhZVfQ10Swfmat55npvUzUSUERFc1JZbXQalh0U1BUnnZxWZtd0inMip40V2QNWx1M3pVS3J1RU5c0ShMsN4T31HeHbhUhpNjWZn4NthaeVJ9c01LZWIteF13N0hfQmf1mpRsgb1BqUClpHOFYNW6EV1NNUWEeGRxS1uNDAMM0hCaFBVcmidIVRGX0xhC1cZB5TT0d2ZB5fcHRAZB1zKCF0T1F4VrJWWphmPieGHBMVTe1s1mY2WFO2BdC16HtCmjh0wNTY4NsUiMywibGFzdEFjUgi2AxR51joxNzYyNTE1Njg3NTUzfQ.vhliUgidFP3ARWqlxR1qQAJpdiBSm2j8EVt6BvXx-ShQ",			
"algorithm": "ecdh",			
"serverPublicKey": {			
"x": "26644146c635054857763276393201319872259043665306389055696",			
"y": "16544595468432985591733c17822468730161c052880474894449578"			
},			
"signatureSupported": true,			
"serverSignaturePublicKey": {			
"x": "31097569923C0604139868590090034485069056515831683235942831",			
"y": "C919938537555021218181577327501328320147366910146265339904"			
},			
"sessionSignature": {			
"x": "556844490783082015223660050040392216183429906148687912855",			
"y": "478192603416360242038970775287756793497358492118705124055",			
"messageHash": "5047781211627034291771882024276935663529859710654155076191",			
"algorithm": "ECDSA-P192"			

Hình 8: Response create session

```
{
  "success": true,           // Trạng thái thành công
  "sessionToken": "eyJhbGciOiJIUzI... ", // JWT token cho phiên làm việc
  "algorithm": "ecdh",       // Thuật toán được sử dụng
}
```



```
"serverPublicKey": {                                // Public key của server
    "x": "26664146263550...",                      // Tọa độ x của public key
    "y": "16544595468432..."                         // Tọa độ y của public key
},
"signatureSupported": true,                        // Hỗ trợ chữ ký số
"serverSignaturePublicKey": {                     // Public key dùng cho chữ ký
    "x": "31097569923206...",                      // Thành phần r của chữ ký ECDSA
    "y": "29199395375550..."                         // Thành phần s của chữ ký ECDSA
},
"sessionSignature": {                             // Chữ ký số cho phiên
    "r": "55684449078308...",                      // Hash của thông điệp được ký
    "s": "47919268354163...",                      // Thuật toán chữ ký
    "messageHash": "50477812116278...",           // Thuật toán chữ ký được sử dụng
    "algorithm": "ECDSA-P192"                       // Thuật toán chia sẻ
},
"signatureAlgorithm": "ECDSA-P192"               // Thuật toán chia sẻ
}
```

Luồng giao thức: Client gửi tham số ECDH → Server trả về public key + session token + chữ ký → Hai bên tính toán shared secret → Dùng shared secret để mã hóa tin nhắn.

Mapping to Source Code

```
1 // File: LoginActivity.java
2 Request request = new Request.Builder()
3     .url("https://crypto-assignment.../session/create?userId=" + this.userId)
4     .addHeader("x-user-id", this.userId)
5     .addHeader("Content-Type", "application/json")
6     .post(RequestBody.create(requestBody.toString(),
7         MediaType.parse("application/json")))
     .build();
```

3.1.2 POST /session/exchange?userId=group-1

Phần Request



Request

Pretty Raw Hex

```

1 POST /session/exchange?userId=group-1 HTTP/2
2 Host: crypto-assignment.dangduongminhnhat2003.workers.dev
3 X-User-Id: group-1
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 2034
6 Accept-Encoding: gzip, deflate, br
7 User-Agent: okhttp/4.11.0
8 Connection: keep-alive
9
10 {
    "sessionToken": "eyJhbGciOiIzIiINiIisInR5cIi6IkpxWCGJ9_eyJpc3Mi0iJTZWN1cmVdAGF0IjoxNmYyNTE1Njg3LCJleHaiOjE3MjI1MTU5DcsInNiYiI6I[redacted]dyb3VwLTEiLCJzaWQiOii4ODI0NDQ4Y2M0MDMsNdcwOTcgOTxMwgSNdUcYWWcYzE3MDU20TBjMDQNT1lMDt3MDI0OWIzMDvhsYzOTBhIixiYwxb3JpdGhtIjoizW0hcaCisInElYmxpY0tlesI6eyJ4IjjoimjYCMj0wHDYyNjMlNTA1NDq1Nmc2MsI3NjM5MzJwMTMx0Tg3MjI10TA0hSYzNTMwMjM40TA1NTY5M1IaInk10iIxNjU0HdUsNTzQCDQzHj44NTU5M7cfsIxNzgMyq20DcsmHDE2MT1wNT14ODAOHsQ4C7Q2NDQ5NTc4In0sImVdY35cHR12BPhGEi0iJxNmF2ZEENiIpHQwJC0XFVhyWnHsZ0drdl1ncP0xFQ0QH15yamrqZESHNUUWFWcTjaXOF17FrM3heYhNHds5CQjVHFEPDpxLYdahfUHYzaw1ZZHACSGcC3pdSFNVSAzchQHqWWSyWENWxM3ZB10hboYnNjSPEL0HjU1dch1RWp+LWYzqZN0HampcELWKCNRbTNybRpRdn2icUg0HHSUfp1bhJkZ0etW12JSxNyUHNYT25NbWhCSjg3YWywNz2xfd0UhuUWj4tRh21sY1NOQUVfekxewZw9fVzdtbENGSOBhUsh1ZK3ZG2TNPQ1VFN5WVpvXnhQ18zTGjYt4dWNE3V0YndSM1dINCxQHHEyY04s1FCb01AZHb02ES5TxmHckFhevEVCdgZKd14ajhscmfRaFpm3JUV1ByYRleEZ1T0wYmEzc1c10ZnExURkrtCUGJGUmFhxwSamilaDZee1Um5acFdBHU45Szhd4DF1c052T3c0WRY10WZKMFZNB-CwzHWEdm01ek1hWWZYTUpb1VBS21m0e2ZdhZVF0T5mhs5npvzuUSUEFK1J2bXqalhOU1UEun2bWZtd0lmM1p40V2QNw1M3pVs3J1RU50c09M3N473lHeHbhNhpumPwN4VThaeV3Pe0ll2W1t#F13NhQfahfYnpjkxgshb1NedRhg1IBqfClphFYNWJev1LNKUWEceGR8sjluUDACMUhcAFBvcmdiYVRGNx0Ne1czB97T0dZ2Z95fcHRA2EZK2POT1F4VrJTTWphbfleGNBMRVfei1sImY2WF02WBaC16HTc2MjUxNTY4N0U1MwyibGFm+dFjdg12aXB51joxMjYNT1Njg3NTUsfq_vhluUg1dFP3AWq1xR1qAJpd1BSm2j8VtcbVx-Shu",
    "clientPublicKey": {
        "x": "3219326529410874192541262463488014560114582110770535426672",
        "y": "37732060849512822640852206417224907842480338805455018091482"
    },
    "clientPublicKeySignature": {
        "r": "3586189277906405261025888273954569256011301129333822167946",
        "s": "38413137610650095036578814524084424521871673798495618077",
        "messageHash": "209311550939029653468419550850415808154317619766564535379",
        "algorithm": "ECDSA-P192"
    },
    "clientSignaturePublicKey": {
        "x": "340858144219692264061599604345640022097270188501791678621",
        "y": "51555258243950165748754167342171038387169304242438507539"
    }
}

```

Hình 9: Request exchange session

```
{
    "sessionToken": "...", // Token phiên từ bước trước, dùng để nhận diện phiên
    "clientPublicKey": { // Khóa công khai của client cho ECDH
        "x": "...", "y": "..." // Tọa độ điểm trên đường cong elliptic
    },
    "clientPublicKeySignature": { // Chữ ký số xác thực clientPublicKey
        "r": "...", "s": "...", // Thành phần chữ ký ECDSA
        "messageHash": "...", // Hash của clientPublicKey
        "algorithm": "ECDSA-P192" // Thuật toán ký
    },
    "clientSignaturePublicKey": { // Khóa công khai dùng để verify chữ ký
        "x": "...", "y": "..." // Tọa độ điểm
    }
}
```

Phản Respone



The screenshot shows a network request from a browser. The response body is a JSON object containing session information:

```
Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Date: Fri, 07 Nov 2025 11:41:27 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Access-Control-Allow-Headers: Content-Type, X-Session-Token
6 Access-Control-Allow-Methods: GET, POST, OPTIONS
7 Vary: accept-encoding
8 Report-To:
  {"group": "cf-nel", "max_age": 604800, "endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4?s=MHZqfNA8SFq01wK5cgRnHztqvyyzXIAiWwGxclrC&qvRVGiiCwAiic27esMYlvlUSgc3B3%CBQh6Kt%CF0yL0en3z7BbmzB1Sy%7juwN9gCZCotskhbTUNYNU1jBcnmYTNTcYz6Wo285wqRLoHlw%3D"}]}
9 Nel: {"report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800}
10 Server: cloudflare
11 Cf-Ray: 99ac8fc8cb0b5c15-SIN
12 Alt-Svc: h3=":443"; ma=6400
13
14 {
  "success": true,
  "message": "Key exchange completed",
  "algorithm": "ecdh",
  "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJTZWNIcmVDAGF0IiwiwAF0IjoxNsYyNTE1Njg3LCJ1eHaiOjE3NjI1MTU5ODcsInI1YiI6Imdyb3VwLTReIJCzaWQ10iI4ODI0NDQ4YCMONDmMzNDew0TgCOTBmHsGNDUCYWEYsE3MDU2OTBjMDQ2NT11NDk3MD100W1zHDVhMsYxOTBhIiwiYWnbh3p4ghIjoiZWNaCisInB1YmpxY0tlesS1eeyd4joiMjY2NjQxNDYyNjMlNTA1NDg1NsccMzI3NjM5MsiIwMTMxIjg3MjI10TA0MhYzNTWjM40TA1NNTYSN1iisInki0iixjNyU0NDUSNTQz0DQzjk4HTUSMHTcMsIxNzgyMjQz0DczHDEZMTIwNTI4DA0Mz40TQzNDQzNTc4In0sImVuY3J5chH1ZERhdGEi0i3j6V1U5UFNjNCheGgOSXbieUS1aTZYQmVEWTzQnFPMWRcQcm7zbG56eG9iNFhRcWs4SDN1SGd30RdcSs20TlpDTX0oHdCMxJOn1iuvNU01YXWzTNFjMjQ1SE95Rc1KKnUyRzYcoJhQ19CSGr4UGR0UnhOs1NzRvdmaMahsQmFCWVg3allyQTNQTh1c0uxZWH1UDWFsUk1WT3jNMHS12U2FV2zKNTFCYUlpCbVcsbUlpoVO2kzTFNNCN1VbFVVWw530U10sEdmWgk1dgd3MzEB0lpvZzQ0MWpTV9hU1WTUhWSDPMActMjZzeFPFSk5XeFRCVv0N1VMM3UDUVN1lyaySWfD0N5WKC5V8VgveHR31dwahChb01NQUJiVHVv2ckhYbEl6YsBrNU4F89z2khL0tMmfroyZypVwscbVYxMh1dWNPsvBFcGxdRrhjMU45Tm1sRE8wLchbQmZ0DAD340w5EndsNu91SeRuVGZRBkpuTNUvV0MtdchzTz20FpZW2n0kF2ccER0Nxw0pMuVFBgWhchEPVd8QTxzCNOBHzsdhdtVVsTng4VFRDdHFWNUd0dnJhCxVlaQdrVbd0akbyZzNmMz1bn2shFr4UWJ6TghMUUF1X0s2T1JK0F2LRw9malhNZ12N2ENsTFVatG1FZmMoRsVERnml4VGlnbTFNZE9xNuBc2FTGZEc3NjNGFEZlh3M0xChmnxbEpPc2EDNmSCM3UxSW4lb1QtMHDuVEMtTFpGu1dDX01s2W00SDNGVdhyQk1RERrlc284Rlp6ZnF1bHN3bDNiSVRUavVv0XFHZcd1SkJxeFlpUlFmRDg0c0tLSjR3hHk5Wx1dF9LUnd5N1NGV21yV1RSVjd1M1pkMONBS1dfZ2c5cWgwY1c0WjBFNGgtawdUbDR1NshHQXQ3SDhWUUt3tVYxTOh4M3U4OHVPOUFChjFROC1s1mNyZWF0ZWBAC1gHt1cHjyUaNTY4NzU1MywibGfzdEFjdg12aXRs1joxNsTyNTE1Njg3ODByfq.QMaCFeUzhhCl_Hf5amKL20hRCxdqqCw0AmKj7XsRABY",
  "clientSignatureVerified": true
}
```

Hình 10: Response exchange session

```
{
  "success": true, // Trao đổi khóa thành công
  "message": "Key exchange completed", // Thông báo trạng thái
  "algorithm": "ecdh", // Thuật toán đang sử dụng
  "sessionToken": "...", // Token phiên (có thể được cập nhật)
  "clientSignatureVerified": true // Xác nhận chữ ký client hợp lệ
}
```

Mapping to Source Code



```
1 // File: LoginActivity.java
2 this.client.newCall(new Request.Builder()
3     .url("https://crypto-assignment.../session/exchange?userId=" + this.userId)
4     .addHeader("x-user-id", this.userId)
5     .post(RequestBody.create(requestBody.toString(),
6         → MediaType.parse("application/json")))
7     .build()).enqueue(new Callback() { ... });
8
9 // Crypto operations
10 CryptoManager.SignatureWithPublicKey signResult =
11     this.cryptoManager.signMessageEphemeral(publicKeyString);
12 requestBody.put("clientPublicKeySignature", signResult.signature.toJSONString());
```

3.1.3 POST /message/send?userId=group-1

Message: name

Request

Pretty Raw Hex

```
1 POST /message/send?userId=group-1 HTTP/2
2 Host: crypto-assignment.dangduongmainhnhat2003.workers.dev
3 X-User-Id: group-1
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 2042
6 Accept-Encoding: gzip, deflate, br
7 User-Agent: okhttp/4.11.0
8 Connection: keep-alive
9
10 {
    "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJtZWN1cmVdAeFOliwiAWFOIjoxMzYtNTIOMzUwLCJleHAiOjE3NjI1MjQ2NTAsInNjYiI6Imdyb3WvLT8LJCJzaWQ1o1xMQ50DQmNs9g3ZDdjMjBkYTU4YTJmMDZ1NTd1mJ1CM71hNDf1MmNmZGV3NjcyNDVjOWMxMmNzZWM4Y2UxYzY1iwiwYxnb3JpdGhtIjoizWWhaCisInB1TwxpY0t1eS16eyJ4IjoimjYCNjQzNDYyMjM1NTIA1NDg1Nzc2Mz13NjMsMz1wTHx0tgSMj110TA0MzTNTMwNjM4GTA1NTyN1isInhri0i1ixNjQ0NDUsNTQCCD0zMejk4NTU5MtcMs1xNzggyMjQ2ODczMDECM1iWNT140DADNzQ40TQ2NDQ5NTc4In0sImw3J5cHb1ZERhdGE1013bGF6b31SehU3aHwUG03eWcBwW1xbGeTINrjAadDrkEwBzGp1TTd2WDVrcGpLSDdob3d2Y2FJzE1I2ZJfSlpWTRNPjBKCzUVcHdQZGJHjF1ZodjZWRIwFwHelineFBCc3HtNmJBV1jEWjZLWTd2ZHe1eUmQ1BNNxFZX1yWjFZ2W1tcGpKaUwWGx0OWxh11DcPdN3VLSd1eLWg52nNUUNcylWNRd2tST1Zza0w4YsEDd2UwAlhudzg1NGQyaLN1NWpJzRQVUNPwlpf1nICRXZ1NGRsSD4cEVTU20WVQYUxt3BeoWx5eSF1WGAzZ2ZFCV1jRTVZxcGcvhmc1dH1jRzjXWTRLchcCVkV1TkRaBd2tChgzUzVrb1lIdmPmeHJLzjhurRBWVVBDnhnHTe1vUno2d1RuSk1WhxwYvao01xcsZRVpVdhpw0UteBWrF1WxRYonZ2ZedSSV1FOwpQWP10VW1WHU1oNjdhNF5KUVUdlTVhWREJ3VmQ4Mkg2VGk3UVBDNH1Uan1NVyBiZzJc0G9oUVd1hmU7QmfEM112SndKbCp6ZGJvbU43Tv04dzFqTmlsaFEmMW85ThnzdFZUAglqTjBkewQWQvndldeY0V3FW0UF1UWdjel1wNkh0SHVpVmVxY2NGV1BCbkJQeHB1RVNwY29EqmZ6YUNnWkSDAhpoct1MdChxR3NHHmdEdmRWbX25cHrvOHnsZDNJdjyVzdyh3JJKX2dPUEpfv2RrTj1zen1CTWNg2WFQbmFXQvp0RHJaYmPmBjZDNMM31ESEdnMVZzaG4QU5MON4VjJnQ1BjOHJFYlhxaGhsdnHUUNLd3BxUF1TWW15Th3e11gdUUUzERbnZ1dEHMvjFLMHE1WVEShkKTzB40FNEd1rc1VCAhpkKV021WhPRKrcxvTaIzh80QnZrUTVwZGRWn3JYcyIsImNyZWF0ZWEbdC16Mtc2MjUyNDM1MDALMwibGfzdBfjdG12axR51joxNzYNTI0mUwMygafQ.UkfS3gTchcj_3wWxB35CfWhge7L_y2VsU020azB5Uo",
    "encryptedMessage": "/xeafq21faB0dzGKIAQqTyHCeo2FdIo4mE9vu3FbnZ0=",
    "messageSignature": {
        "r": "218816676298316657654876228128845235806953469231039220938",
        "s": "8834650781799140550678389378768900667824188992612638787108",
        "messageHash": "4729872575924978881334107727117314030315322545688821775079",
        "algorithm": "ECDSA-P192"
    },
    "clientSignaturePublicKey": {
        "x": "1008366956292591305325928042610209349405253184479361695265",
        "y": "234617650452058772836667588474985074719377559299385885792"
    }
}
```

Hình 11: Request Message "name"

```
Response
Pretty Raw Hex Render

1 HTTP/2 200 OK
2 Date: Fri, 07 Nov 2025 14:05:59 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Access-Control-Allow-Headers: Content-Type, X-Session-Token
6 Access-Control-Allow-Methods: GET, POST, OPTIONS
7 Vary: accept-encoding
8 Report-To:
  ("group": "cf-nel", "max_age": 604800, "endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4?s=ZRUqlJLTofY5InsMwB84LEDYqx+2BSuUH4agHzurhcpKrSqa8MTPgdkPF8wdvLhAGSVeDcKaI2C8QfTIE2kay7t2FjZER5Y#Wps1lCaahSSbxRIdbt2FSN0TVG50hSMzoiYhTs1+26EA1dKxNeb#CFvLYLR#43D#3D"}])
9 Nel: {"report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800}
10 Server: cloudflare
11 Cf-Ray: 99ad63744c3fcde-SIN
12 Alt-Svc: h3=":443"; ma=86400
13
14 {
  "success": true,
  "encrypted_response": "ysecpQD9cBEmfT16Jg3cx/GAc0mfWgpqOk+YoCstTHoBir2aoqkgrAYg+3YH0h6UlVva4qd5/LByWQ7CnVB+2Q1hRN3h1NUHSHfB+5hCx/37Fz81ldU0ILKgCrYtiQ0w8g==",
  "sessionToken":
    "eyJhbGciOiJIUzI1NiIsInR5cIiKpXCVJS.eyJpc3MiOiJtZWNnVmUdAfp0IjoxNzYyNTI0MsUSLCJleHAiOjE3NjI1MjQ2NTksInNlYiI6Iadyb3VsLTetIiCzawQiOjIxMzQ50DQmNsge3ZDdjMjBzEzQmZzI0TA0MycTNTWjhM40TAINTYNS1isInkiO1iXNjYUNODUsNTQ2DQmHj4kANTUSTcmXaiXnsyHjQzODczmDfEMTfWnt14DA0NsQ40TCQNDQ5NTc4InosIuVxJ5JcSR1ZERhdGE1iY5JnUhYImsHfzD3B3NjE2ZGZERDWFASzFzds1pfdVRDap3V1lhNWWCQmibEZHMUrxZEN3eUtOUkrRsU1DVFrTBucmERT1iWtENETndJkRGfqtWcUfVpTzOfKF2FLHMVWHTH9nDUVFWxtT05Frm14UTDQ001EdWnHeftfNmp7URBwsEJRSwsR2h1dUcEri0RC0fGzRGC2STPM2EhdQ2xBVVFeN051ITUrzcdEaBnZn1lUUVNAHZDZ1i2ZD7GbnRaTWwva0hnZJCM9CbFRR0051TfFsbnH0V2F0z105CULsaWB1RmlRgPhAWFlY2lk1Q09HaJnUVEtWhoeGRMVWUSeZWUQySmZqdlNKUg5DWH1UG1LESEZsenFcndlWjgB5TnkzdZ23SchOb1EDQGtFq1VzQphbEVj2mzR3U1ExSU8y0d0c4TfVHgjtTmPmTzR0GV0a0ZRN3FISj3Z6VGNtRtCJuJmTwUtrR50WmJc0FMtkolb1zWjJC0UDVjRqWt4dGJNTNrt3FFQzMeEpCUHdVt1R0c1dHNwlnUwdfNmpENVXKOGtTR1JH0JXVuaxNfHhAhrHndVh1leUSXZmpDSTJlcU$yranFYaZQ7WJkcfFGVYhESThjS$11dF83a0oOS1fxSkdSd2s1UXctVnko$15aqWRFc3FSVut4ekHHMjh1Opnc3dv0XW4Q1VNZ0kwtFmR1h0eXi1uInFlalIwWHRE105GsaWfN1npkcfZXFtd0Y5T3pjTUctQbWpZenRyVlhCOWE3TTFnMcGdPtaYtCOVB1YnV2Qvc1RwthBx12QTR6RzNOXp6Y1FSSHJvXKtpt1h6LWtUHptEX2qTUFjzSVWdnHmFStc4ZwvnVnfZUE2y1sImhyZF0ZWBdC1GENTcCHjMyNDHMDAlMCwibGfzdfF7dg1zCaMR51joxNzYt10Mu5NTq0f.jYxszJ-EfQZedBdpVua3eZmZpb1bHGQZxenMyb5pmk",
  "messageSignatureVerified": true,
  "responseSignature": {
    "r": "1558826231456105024506923239058239548395244759825056785525",
    "s": "223473087525802465172311238705109551305028467503533540517",
    "messageHash": "5475819209312642364361028879769950930448495784620793987",
    "algorithm": "ECDSA-P192"
  },
  "serverSignaturePublicKey": {
    "x": "45116430885794711383432756117378844512036634066090868365",
    "y": "51920017900503152297717703651755183920554460880142952798"
  },
  "signatureAlgorithm": "ECDSA-P192"
}
```

Hình 12: Response Message "name"

Message: age



Request

```

Pretty Raw Hex
1 POST /message/send?userId=group-1 HTTP/2
2 Host: crypto-assignment.dangduongminhhat2003.workers.dev
3 X-User-Id: group-1
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 2042
6 Accept-Encoding: gzip, deflate, br
7 User-Agent: okhttp/4.11.0
8 Connection: keep-alive
9
10 {
    "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpWCXJS.eyJpc3Mi0iJTZWNIcmVdAeGF0IjoxNsYNTI1MDk0LCJlreHAI0jE3NjI1MjUsOTQsInN1Yi16Imdyb3VwLT8LJCzaWQi0i0y0TnInjkzYTfjM3MxYUzUzMeJkYwvYzcILNedkYtClNzHdkwNWFly24YTFlODk1MTICNqEw0TF10G13MT0iwiyWxnb3JpdGhIjjoizWNkaCIsInB1YaxpY0teSl6eyJ4IjoiMjY2Nj0xNDYvNjMINTA1ndg1nMz2MzI3NjM5MzIwHTNx0Tg3Mj11OTAMyZCMTWd1jM40TA1NTY5N1i1mkl0i1xNjju0UDU5NTQCD0Mj4k4NTU5MTczM1xNsgyMjQ2ODcmcDEMTiwiT14DA0MwQ40TQ2NDQ5NTc4In0s1mVuY3J5jchR1ZEhdG10i1jXb1267UpYtmFrZ1s+TENnetTPkUVFUM1sC12NGcsRa32j3pRZHF6G51LYT1ZUbWtchb15SxW4jk4jN0D1mJUUD1dkSyCgdqndSSHVL51bH01s1NkRPY1xNsdwREFNMjhEXK25CVV2zTETLNUFkEYap3Njd2vZFSJQ2LXcF05V7JGc0s3Y7TA20Ew1clp1SFk5X0wTbkrwvYThK3W3aShpB0QJHudvNhcJc21jIMm2aeDVYJ12CamsXc2PBR11QH1rbD1gWBUDUmShxa0xQk11VnE3S24zTTBWRE2zUhph0WfsdGBETHV1mD14PndMlhaT0c1sWVJNWVCAvZzZcc0ghGh1lwUTFp0khXa7ApWacs1PhRukyaxdPaWWNNj4003RkemdhLW1cReBwc2z1PFewHT1pEcZH1Rmd1cAx0v0HVVmMxkzLx3h0dFD2S1qjJ0NjcsQmSh3NHTTZKSNi3QViuUd0WkSVV80Zmg1Uwh4VocHTVZFc3VqUvgxNmFuV1NHdGh0cEE4014Tdm13c09hT1LFRhneHFPanhdT2hBPUxM0VGZEHThUFY0GHOuU2JQn1taf1g5jBCdixaf1g01g6UUmteVh3daiWmJtKcNeEkybV10qfMK23wv0KFYUVTvTnJ1w1hRcj2UTm2SwMs1P0FyU1nsb0U00V1QbQp1t1F1FaYmpBaGR0VFNmc0dn2GxJ2T2S5Pp1c1JXK19PvNb1VMMNF5R5+BWVH0wvEINONX1q2dySmc3vBNhAudnMC4EUUVJCR1V5S05wEVWfp3aVYV51uBFYQlo1shGjaACIxU3N3WhprcFrAbEJYVFLC1JvJvRDUHnvhwLRfaGSamrSGcyd1g0Vl0TmP3R11T2ZGn0HFFTJ2T1VLUWPNWVUpybV5TQ1WnREHNSc2c2OUgUU09sa1hWkSPbyis1mNyZWF0ZWBdC1EMTc2Hj9UyNTASNDYyNiwibGFzdFpjG12axRS1joxNsyYNT1MDk0OD1xf0.cWmqUoqPlfXegix1aZ_kUbFLW0XzbzuZk8tmazuQs",
    "encryptedMessage": "2UDvGf94LGgyo1ixM+4hASuefSQABMh0tAzahQk7w=",
    "messageSignature": {
        "r": "5546376952702695307653730758020201308122512749005063000861",
        "s": "5732387022597404545C8025126915329880485767525969867644682462",
        "messageHash": "34125459121068906605310029330236805171731038685533071064280",
        "algorithm": "ECDSA-P192"
    },
    "clientSignaturePublicKey": {
        "x": "3632245240392326001517341543016162475405542302344263475550",
        "y": "4501559415404781774164680221704569304354809569831110372051"
    }
}

```

(0 highlights)

Hình 13: Request Message "age"

Response

```

Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Date: Fri, 07 Nov 2025 14:18:30 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Access-Control-Allow-Headers: Content-Type, X-Session-Token
6 Access-Control-Allow-Methods: GET, POST, OPTIONS
7 Vary: accept-encoding
8 Report-To:
9 {"group": "cf-nel", "max_age": 604800, "endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4?s=d4YsMZEiCDSJDdkF8BC6XceBkyi+CBfsPn+CF5TjQj0JoVwXicu3506nyHeXw051iWV76k7G+BYh0SLs8kEdSAFYOLJ4bMSRaMaSzLlorgIYHmVjSMVX2BtBrESw%2FrvtxHnk8jezckhnK7yHNHSR+CBjt9T6+2Ftw+3D+3D"}]}
10 Nel: {"report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800}
11 Server: cloudflare
12 Cf-Ray: 95ad75d6a5c4fdC8-SIN
13 Alt-Svc: h3=/:443"; ma=8400
14 {
    "success": true,
    "encryptedResponse": "vPvv4Eri+b0g7VwvEcXm6g8YarHUJyca0Cr0Ngf0tUAYQ0tTFSnRBrR17s+d7j512y2C1PT7YAC2XgWYF/QWQc1s4wFTdTevoE5hFn8f7g+4sZs0uNf39W9vdshjzYTWWUJ1JPQ==",
    "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpWCXJS.eyJpc3Mi0iJTZWNIcmVdAeGF0IjoxNsYNTI1MDk0LCJlreHAI0jE3NjI1MjUsOTQsInN1Yi16Imdyb3VwLT8LJCzaWQi0i0y0TnInjkzYTfjM3MxYUzUzMeJkYwvYzcILNedkYtClNzHdkwNWFly24YTFlODk1MTICNqEw0TF10G13MT0iwiyWxnb3JpdGhIjjoizWNkaCIsInB1YaxpY0teSl6eyJ4IjoiMjY2Nj0xNDYvNjMINTA1ndg1nMz2MzI3NjM5MzIwHTNx0Tg3Mj11OTAMyZCMTWd1jM40TA1NTY5N1i1mkl0i1xNjju0UDU5NTQCD0Mj4k4NTU5MTczM1xNsgyMjQ2ODcmcDEMTiwiT14DA0MwQ40TQ2NDQ5NTc4In0s1mVuY3J5jchR1ZEhdG10i1jXb1267UpYtmFrZ1s+TENnetTPkUVFUM1sC12NGcsRa32j3pRZHF6G51LYT1ZUbWtchb15SxW4jk4jN0D1mJUUD1dkSyCgdqndSSHVL51bH01s1NkRPY1xNsdwREFNMjhEXK25CVV2zTETLNUFkEYap3Njd2vZFSJQ2LXcF05V7JGc0s3Y7TA20Ew1clp1SFk5X0wTbkrwvYThK3W3aShpB0QJHudvNhcJc21jIMm2aeDVYJ12CamsXc2PBR11QH1rbD1gWBUDUmShxa0xQk11VnE3S24zTTBWRE2zUhph0WfsdGBETHV1mD14PndMlhaT0c1sWVJNWVCAvZzZcc0ghGh1lwUTFp0khXa7ApWacs1PhRukyaxdPaWWNNj4003RkemdhLW1cReBwc2z1PFewHT1pEcZH1Rmd1cAx0v0HVVmMxkzLx3h0dFD2S1qjJ0NjcsQmSh3NHTTZKSNi3QViuUd0WkSVV80Zmg1Uwh4VocHTVZFc3VqUvgxNmFuV1NHdGh0cEE4014Tdm13c09hT1LFRhneHFPanhdT2hBPUxM0VGZEHThUFY0GHOuU2JQn1taf1g5jBCdixaf1g01g6UUmteVh3daiWmJtKcNeEkybV10qfMK23wv0KFYUVTvTnJ1w1hRcj2UTm2SwMs1P0FyU1nsb0U00V1QbQp1t1F1FaYmpBaGR0VFNmc0dn2GxJ2T2S5Pp1c1JXK19PvNb1VMMNF5R5+BWVH0wvEINONX1q2dySmc3vBNhAudnMC4EUUVJCR1V5S05wEVWfp3aVYV51uBFYQlo1shGjaACIxU3N3WhprcFrAbEJYVFLC1JvJvRDUHnvhwLRfaGSamrSGcyd1g0Vl0TmP3R11T2ZGn0HFFTJ2T1VLUWPNWVUpybV5TQ1WnREHNSc2c2OUgUU09sa1hWkSPbyis1mNyZWF0ZWBdC1EMTc2Hj9UyNTASNDYyNiwibGFzdFpjG12axRS1joxNsyYNT1MDk0OD1xf0.cWmqUoqPlfXegix1aZ_kUbFLW0XzbzuZk8tmazuQs",
    "messageSignature": {
        "r": "5184603560030394745263847692250111198412913466336972596251",
        "s": "550391781021241263116400056802147088603952608405897291",
        "messageHash": "4395756437924055684400520514778689765988571384464431605701",
        "algorithm": "ECDSA-P192"
    },
    "serverSignaturePublicKey": {
        "x": "41770357463567617179840893866066576778831751653107261911011",
        "y": "49759773261794447538716579719162391239101598c5938615361507"
    },
    "signatureAlgorithm": "ECDSA-P192"
}

```

Hình 14: Response Message "age"



Message: location

Phản Request

```
Request
Pretty Raw Hex
1 POST /message/send?userId=group-1 HTTP/2
2 Host: crypto-assignment.dangduongsinhhhat.2003.workers.dev
3 X-User-Id: group-1
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 2044
6 Accept-Encoding: gzip, deflate, br
7 User-Agent: okhttp/4.11.0
8 Connection: keep-alive
9
10 {
    "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcGVJ9.eyJpc3MiOiJtZWnlcmVDAgF0IiwiWF0IjoxNzYNT1lMTk1eHaiOjE3NjI1MjU0MTAsInI1Yi16Imdyb3VwLTExLCJzaWQ10iIy0TNIjNjkeYTFjMChMycUoMsJkYmaIwYsc1NsdkYTe1NzZhMDMwMWF1YU4YTf1ODk1MTICjBwot1OG13MT10iixiYWrbn3JpdGhIjjoizZWNaCisInBlYmxpY0tLeSI6eyJ4IjoiMjYCNjQxNDYyNjM1NTA1NDg1Nzc2MzI3jHSmsIwMTMs0Tg3Mj110TAOMsYcNTMwMjM40TAINTY5HiisInki0iixiNjU0NDU5NTQ20DqMj4NTU5MTc0MsInNgzyMjQ2ODczMDcEMT1wNT140DA0NzQ40TQ2NDQ5NTc4In0sImVuY3j5cHRI2EhdGEi0ijs1s15aXRN0HBFTFQRhAbDBwNpWNilc1UUVRLUFNpRlpY1ZxNjydnYZ2TDJdWhJTUF1eUDwemsEUJJC3dwf1QhERvS1MNC173mlolWH12zv5VVfUm5zCNhWbHc2a3QxTHpWH3BsaJNwVsTEhAOGB0VnRcREOFu2hVvYvQ2Ghcvdjh0RZ2x1cVV1VnVvYmVRUVdUnhETUjsODRwc3g5bEU1QWtEb0410VRrUskyVmYxdjdn3aVVTThf2dgCUPRjNvEBT11q5VYOLVpQWVsbhNAx1AyadWUS2NUU01z2Ez2NCzJMGvUms3Q7hmd2x4dyByanFVN1sc1nBV11uUEFVN02pd244exWQSjdm1B8Tgdt195b0xsSj4RhnoYjZueFrccjRjHWZub2p5ZV1w02zTj14au0sREcyV1LYta1zdjFnTGJWCNPmRz1MWz2UvppYjBBH1BjQkxhEZxai1vVUVnQXmURBaieV8V2VnUHb1JAWjATUGEMhxQxNTU21TUdx0R1YTWvZD11amV1T1ROYm1UmwRWNnd1Wm91ajBSblMKfcxdsDsl1vTVn1Q2ZM24RpVhXoMEmzpmOnsrhSO/pJ7089jU8zaChveUrseWjLQ4dJ2d1d1VGFNnhshVv1yAd4TMOpnHVnjQGUFRhU1ybkQ3ZUUm3JQjzsUCTLc0573ARVVPbUcQTDrhyG0tRUn1lWRAK0NYChFVFFcOpq219PbfclQ4s80TDh1TXJGb5ASHPBuT0JEMD1hWcnR19tSy05AV15Wb5sAoNxq1lmTwFck1p212sV05LWV1khVhCAgWnADNjmh3VFlN3Bc0GzalBqg5c5WmpkS385MwxtYUJfHjZKZVFBvahNmVhMmcMTH4QzUuThdUvnNwZjNGW1RvdWmheWEZ2F1NSVhPdHUStV0C1s1mNy2W02WBAC1cH7rcMGuYTA5D0YyN1xibGFsdEFjdg12aC85joxNzYNT1lMTk1DQwF0.DAaqoLBQijhrBZU2BtQ17j6mdD35j9ATONQz29Kqn1",
    "encryptedMessage": "tGm4YjNw4lgcUHSTcAKMRBqt7pJchHc/vwRgloocVpJZkQyLdn",
    "messageSignature": {
        "r": "272B3C68173036581533508c36776176540450537516320076586800c",
        "s": "5490504164109417670612015257600932074683177537798963632",
        "messageHash": "647619986621063837730092445350442384679529105880335416105",
        "algorithm": "ECDSA_P12"
    },
    "clientSignaturePublicKey": {
        "x": "-1205781753405392840229876880117756006201766575552020831154",
        "y": "83513508497510884194593460385297846795258524670840071385"
    }
}
```

Hình 15: Request Message "location"

{

```
"sessionToken": "...", // Token phiên
"encryptedMessage": "tGm4YjNw4...", // "location" đã mã hóa
"messageSignature": {...}, // Chữ ký số cho "location"
"clientSignaturePublicKey": {...} // Khóa verify chữ ký
}
```

Phản Respone

```
Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Date: Fri, 07 Nov 2025 14:21:06 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Access-Control-Allow-Headers: Content-Type, X-Session-Token
6 Access-Control-Allow-Methods: GET, POST, OPTIONS
7 Vary: accept-encoding
8 Report-To:
  ("group": "cf-nel", "max_age": 604800, "endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4?s=lxSDxEwFTi0DRTBKehFNTLtwoUlrxLdsEDKg5Vpzqem5chIJKcSAhHyLLAs1zBqCKnf08Apj=s2tY8Z8SwAmVpc0z565UqwfH0DlwvzCkXmvwpuDltputpUaq73aprlJpnEgqPSDkmsaA13Dv3D" }])
9 Nel: {"report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800}
10 Server: cloudflare
11 Cf-Ray: 95ad795a6d9f59e-SIN
12 Alt-Svc: h3=::443; ma=60400
13
14 {
  "success": true,
  "encryptedResponse": "nJ0f08uSw3QBF/HQ2WQXerVi fuLJadBW53uJ3dFMXCMcS1xAJ2SjQKHaPyfqztSdvnISFOsQxs=6HdABaDSViAqr7gecJbIPTsdxKEACgt/acoc=",
  "sessionToken": "eyJhbGciOiAiJUslInN1iLnsEc16IkpXVCJ9.eyJpc3Ni0iJzTzWnlcmVdaGFlIjoxNzYyNTI1MjY2LCJleHAiOjE3Nj1MjU1NjYsInN1Yi16Imdyb3VwLT1LJCjzaWQ10i1yGTHmJkxYTfjHCMxYCUsMsJkY1alYzclNsdkYtC1n2zHNdKwHfM0DlwvzCkXmvwpuDltputpUaq73aprlJpnEgqPSDkmsaA13Dv3D" })
15 Nel: {"report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800}
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
427
428
429
429
430
431
432
433
434
435
436
436
437
438
439
439
440
441
442
443
444
444
445
446
447
448
448
449
449
450
451
452
453
454
455
455
456
457
458
458
459
459
460
461
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
15
```

Hình 16: Response Message "location"

```
{  
    "success": true, // Thành công  
    "encryptedResponse": "nJ0f89Uu2BQF/HQZWQXsrV...", // Phản hồi đã mã hóa  
    "sessionToken": "...", // Token mới  
    "messageSignatureVerified": true, // Xác nhận chữ ký "location" hợp lệ  
    "responseSignature": {...}, // Chữ ký số phản hồi  
    "serverSignaturePublicKey": {...}  
}
```

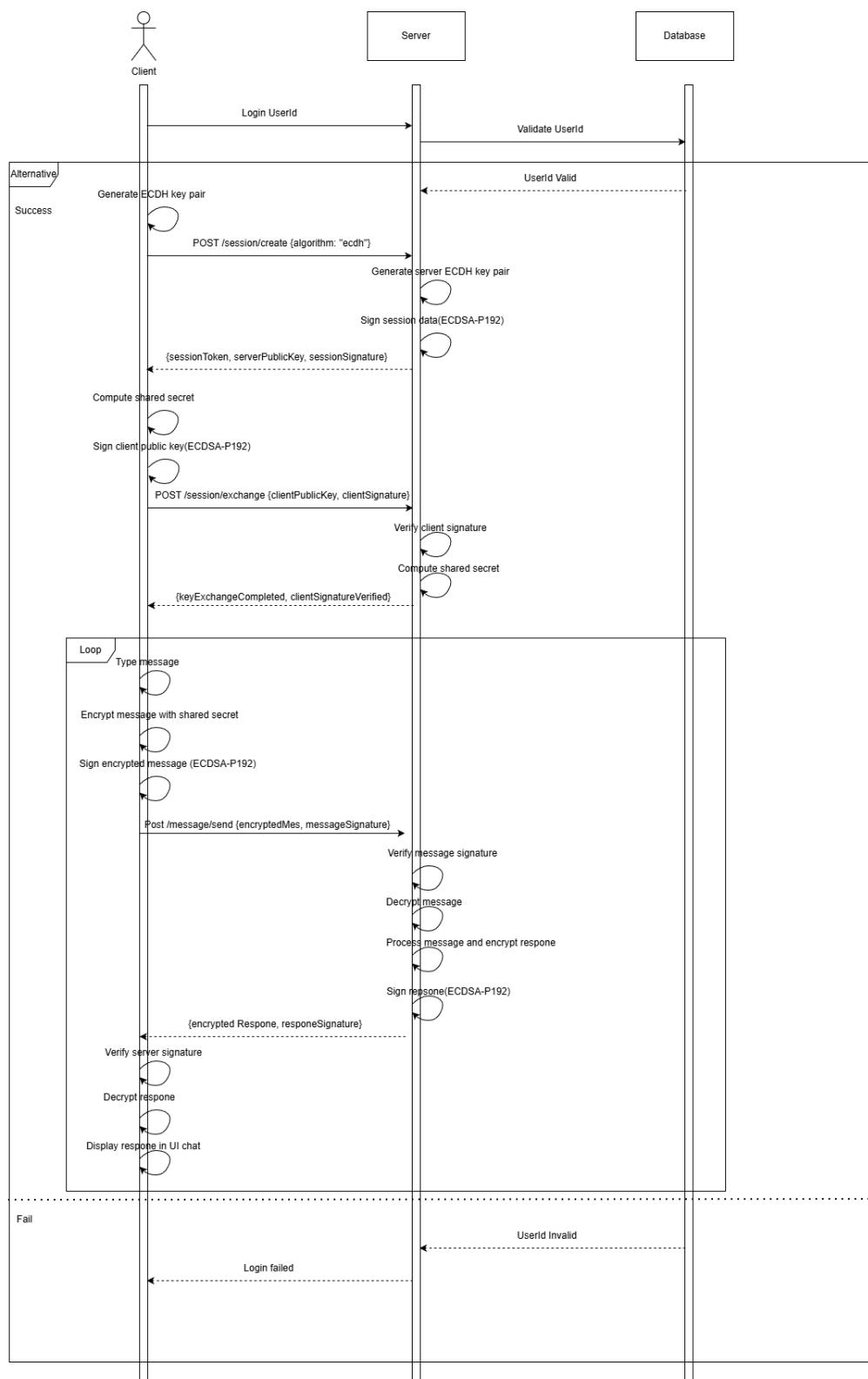
Mapping to Source Code



```
1 // File: ChatActivity.java
2 Request request = new Request.Builder()
3     .url("https://crypto-assignment.../message/send?userId=" + this.userId)
4     .addHeader("x-user-id", this.userId)
5     .post(RequestBody.create(requestBody.toString(),
6         MediaType.parse("application/json")))
7     .build();
8
9 // Crypto operations
10 CryptoManager.SignatureWithPublicKey signResult =
11     this.cryptoManager.signMessageEphemeral(encryptedMessage);
12 requestBody.put("messageSignature", signResult.signature.toJSONString());
```

Sơ đồ hóa luồng giao thức

Diagram: <https://drive.google.com/file/d/1TtPYCVfeUdlvLpeXV0XQ0btbYFC8L7RY/view?usp=sharing>



Hình 17: Secure Chat Protocol Sequence Diagram

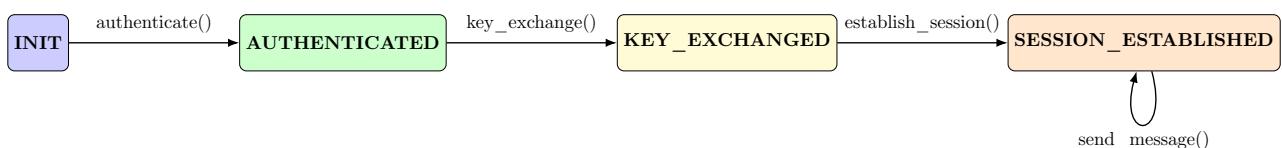


3.2 Re-implementation & Protocol Reconstruction

Giao thức Secure Messaging Component (SMC) được thiết kế để tạo ra một kênh truyền thông an toàn giữa client và server thông qua mạng không tin cậy. Giao thức này sử dụng các công nghệ mật mã hiện đại bao gồm:

- **ECDH** (Elliptic Curve Diffie-Hellman) cho việc trao đổi khóa
- **ECDSA** (Elliptic Curve Digital Signature Algorithm) để xác thực
- **AES-256-GCM** để mã hóa tin nhắn
- **PBKDF2-HMAC-SHA256** để dẫn xuất khóa phiên

Dựa trên phân tích Burp Suite ở phần 3.1, nhóm đã tái tạo giao thức thành bốn giai đoạn chính như mô tả trong Hình 18.



Hình 18: Sơ đồ chuyển trạng thái của giao thức SMC

3.2.1 Xác thực (Authentication)

Giai đoạn xác thực là bước đầu tiên và rất quan trọng trong giao thức. Mục đích chính của giai đoạn này là để client xác nhận danh tính với server và tạo một phiên làm việc mới. Ngoài ra, đây cũng là lúc server cung cấp tham số của đường cong elliptic P-192 mà cả hai bên sẽ sử dụng cho tất cả các phép tính mật mã trong phiên này.

Khi giai đoạn xác thực bắt đầu, client phải kiểm tra xem trạng thái hiện tại có phải là INIT không. Nếu không, điều này có nghĩa là client đã gửi yêu cầu xác thực trước đó và không nên gửi lại. Tiếp theo, client khởi tạo đường cong elliptic P-192. Đây là một đường cong được NIST chứng nhận, có modulus nguyên tố p dài 192 bit (tương đương 24 bytes), một điểm sinh G, và một hạng mục order xác định số điểm trên đường cong.

Sau khi tạo đường cong, client tạo một cặp khóa ECDSA dài hạn. Khóa riêng (sk_sign) sẽ được giữ bí mật trên client, trong khi khóa công khai (pk_sign) có thể được gửi cho server. Cặp khóa này sẽ được sử dụng để ký tất cả các tin nhắn từ client, cho phép server xác minh rằng tin nhắn thực sự đến từ client này.

Tiếp theo, client chuẩn bị một payload JSON chứa các tham số đường cong P-192. Payload này bao gồm modulus nguyên tố p, các hệ số a và b của phương trình đường cong, tọa độ của điểm sinh G (G_x và G_y), và hạng mục order của điểm sinh. Tất cả những thông tin này đều được chuyển đổi thành dạng chuỗi để gửi dưới dạng JSON.



Client sau đó gửi một yêu cầu HTTP POST tới điểm cuối /session/create trên server, kèm theo user ID trong tham số truy vấn. Server sẽ nhận yêu cầu này, xác minh các tham số đường cong, và tạo một phiên mới. Server cũng tạo một cặp khóa ECDH cho nó và gửi công khai khóa này cùng với một session token duy nhất trả lại client.

Khi client nhận được phản hồi, trước tiên nó kiểm tra mã trạng thái HTTP. Nếu không phải 200, có nghĩa là server gặp lỗi. Tiếp theo, client phân tích JSON từ phản hồi. Nếu trường success không phải true, có nghĩa là server từ chối yêu cầu vì một lý do nào đó. Nếu tất cả đều ổn, client lưu session token và khóa công khai ECDH của server để sử dụng sau này.

Cuối cùng, client cập nhật trạng thái thành AUTHENTICATED, cho biết rằng nó đã xác thực thành công và sẵn sàng để giai đoạn tiếp theo.



Input: user_id (String), server_url (String)

Output: success (Boolean)

```
1 PROCEDURE
2   if state ≠ INIT then
3     OUTPUT "Invalid state: " + state;
4     RETURN FALSE;
5   end
6   curve ← EllipticCurve(P_192_PARAMETERS);
7   OUTPUT "Initialized P-192 elliptic curve";
8   generate_ECDSA_keypaircurve, sk_sign, pk_sign;
9   OUTPUT "Generated long-term identity keys";
10  payload ← CREATE_OBJECT();
11  payload["algorithm"] ← "ecdh";
12  payload["curveParameters"] ← CREATE_OBJECT();
13  payload["curveParameters"]["p"] ← STRING(P_192_P);
14  payload["curveParameters"]["a"] ← STRING(P_192_A);
15  payload["curveParameters"]["b"] ← STRING(P_192_B);
16  payload["curveParameters"]["Gx"] ← STRING(P_192_GX);
17  payload["curveParameters"]["Gy"] ← STRING(P_192_GY);
18  payload["curveParameters"]["order"] ← STRING(P_192_ORDER);
19  endpoint ← server_url + "/session/create?userId=" + user_id;
20  headers ← CREATE_HEADERS("Content-Type": "application/json", "x-user-id":
21    user_id);
22  response ← HTTP_POST(endpoint, payload, headers);
23  if response.status_code ≠ 200 then
24    OUTPUT "HTTP error: " + STRING(response.status_code);
25    RETURN FALSE;
end
```



```
1 data ← JSON_PARSE(response.body);
2 if data.success ≠ TRUE then
3     OUTPUT "Server rejected authentication";
4     return FALSE;
5 end
6 token ← data.sessionToken;
7 if data.serverPublicKey ≠ NULL then
8     server_pk_json ← data.serverPublicKey;
9     pk_server ← RECONSTRUCT_POINT(server_pk_json.x, server_pk_json.y, curve);
10    OUTPUT "Stored server ECDH public key";
11 end
12 state ← AUTHENTICATED;
13 OUTPUT "Authentication successful";
14 return TRUE;
```

Algorithm 1: authenticate()

3.2.2 Trao đổi khóa (Key Exchange)

Giai đoạn trao đổi khóa là phần quan trọng nhất của giao thức vì nó cho phép client và server thiết lập một shared secret mà chỉ cả hai bên biết. Shared secret này sẽ được sử dụng để dẫn xuất tất cả các khóa mã hóa cho phiên làm việc. Ngoài ra, giai đoạn này cũng bao gồm xác thực kỹ thuật số bằng ECDSA để đảm bảo rằng khóa công khai ECDH thực sự đến từ client đã xác thực.

Giai đoạn trao đổi khóa bắt đầu bằng việc tạo một cặp khóa ECDH tạm thời (ephemeral). Khác với cặp khóa ECDSA dài hạn mà client tạo ở giai đoạn xác thực, cặp khóa ECDH này chỉ tồn tại cho phiên hiện tại. Nếu phiên kết thúc hoặc client khởi tạo trao đổi khóa mới, khóa ECDH này sẽ bị hủy và thay thế bằng cặp khóa mới. Điều này cung cấp tính chất forward secrecy: ngay cả khi kẻ tấn công chiếm được khóa riêng ECDH hoặc khóa ký dài hạn trong tương lai, chúng cũng không thể giải mã các tin nhắn từ phiên quá khứ vì shared secret phụ thuộc vào khóa ECDH ephemeral đã bị hủy.

Tiếp theo, client phải serialize khóa công khai ECDH thành dạng bytes theo tiêu chuẩn X962. Định dạng này bắt đầu với byte 0x04 (chỉ ra điểm không nén), tiếp theo là 24 bytes cho tọa độ x, rồi 24 bytes cho tọa độ y. Client sau đó trích xuất các tọa độ này và chuyển đổi thành một đối tượng JSON chứa hai trường "x" và "y", mỗi trường là một chuỗi biểu diễn số nguyên ở dạng thập phân.

Client tính toán hash SHA256 của chuỗi JSON này (được serialize không có khoảng trắng với separators=(',', ':')). Hash được tính để thay vì ký toàn bộ cấu trúc JSON, chỉ cần ký một giá



trị cố định 32 bytes. Sau đó, hash này được chuyển đổi thành số nguyên và rút gọn modulo với hạng mục order của đường cong P-192, đảm bảo rằng giá trị nằm trong phạm vi hợp lệ cho thuật toán ECDSA.

Tiếp theo là bước ký số. Client sử dụng khóa riêng ECDSA dài hạn (sk_{sign}) đã được tạo ở giai đoạn xác thực để ký hash của khóa công khai ECDH. Việc sử dụng khóa ký dài hạn thay vì tạo khóa tạm thời mới đảm bảo tính nhất quán trong việc xác thực danh tính client trong suốt phiên làm việc. Chữ ký ECDSA tạo ra hai giá trị r và s , trong đó r là tọa độ x của điểm $R = k \times G$ (với k là số ngẫu nhiên) rút gọn modulo order, và s được tính theo công thức $s = k^{-1} \times (hash + r \times sk_{sign}) \pmod{o}$.

Client chuẩn bị payload JSON chứa session token hiện tại để xác định phiên làm việc, khóa công khai ECDH (`clientPublicKey`), chữ ký bao gồm các thành phần r và s cùng với hash của thông điệp và thuật toán sử dụng (`clientPublicKeySignature`), và khóa công khai ECDSA dài hạn (`clientSignaturePublicKey`) để server có thể xác minh chữ ký. Payload này được gửi tới server thông qua HTTP POST tới endpoint `/session/exchange`. Server nhận payload và thực hiện hai bước xác minh quan trọng. Đầu tiên, server kiểm tra xem khóa công khai ECDH có nằm trên đường cong P-192 không bằng cách thay tọa độ vào phương trình đường cong $y^2 = x^3 + ax + b \pmod{p}$. Thứ hai, server xác minh chữ ký ECDSA bằng cách sử dụng khóa công khai được cung cấp trong payload. Nếu cả hai kiểm tra đều thành công, server trả về một phản hồi với trường `clientSignatureVerified` được đặt thành true. Server có thể cấp một session token mới để tăng cường bảo mật, vì vậy client phải cập nhật token của nó nếu có.

Cuối cùng, khi tất cả điều kiện được thỏa mãn, client chuyển sang trạng thái KEY_EXCHANGED, cho phép tiến hành giai đoạn thiết lập phiên.



Input: None

Output: success (Boolean)

```
1 PROCEDURE
2   if state ≠ AUTHENTICATED then
3     | RETURN FALSE;
4   end
5   generate_ECDH_keypaircurve, sk_client, pk_client;
6   OUTPUT "Generated ephemeral ECDH keypair";
7   pk_bytes ← serialize_ECDH_public_key(pk_client, curve);
8   coord_length ← GET_COORDINATE_LENGTH(curve);
9   x_coord ← EXTRACT_BYTES(pk_bytes, 1, coord_length);
10  y_coord ← EXTRACT_BYTES(pk_bytes, 1 + coord_length, coord_length);
11  client_public_key ← CREATE_OBJECT();
12  client_public_key["x"] ← STRING(BYTES_TO_INT(x_coord));
13  client_public_key["y"] ← STRING(BYTES_TO_INT(y_coord));
14  message_str ← JSON_STRINGIFY(client_public_key);
15  message_bytes ← ENCODE_UTF8(message_str);
16  hash_digest ← SHA256(message_bytes);
17  hash_int ← BYTES_TO_INT(hash_digest);
18  hash_int_mod ← hash_int mod curve.order;
19  generate_ECDH_keypaircurve, sk_sign_eph, pk_sign_eph;
20  ECDSA_signcurve, sk_sign_eph, hash_int_mod, r_int, s_int;
21  payload ← CREATE_OBJECT();
22  payload["sessionToken"] ← token;
23  payload["clientPublicKey"] ← client_public_key;
24  payload["clientPublicKeySignature"] ← CREATE_OBJECT();
25  payload["clientPublicKeySignature"]["r"] ← STRING(r_int);
26  payload["clientPublicKeySignature"]["s"] ← STRING(s_int);
27  payload["clientPublicKeySignature"]["messageHash"] ← STRING(hash_int_mod);
28  payload["clientPublicKeySignature"]["algorithm"] ← "ECDSA-P192";
29  payload["clientSignaturePublicKey"] ← CREATE_OBJECT();
30  payload["clientSignaturePublicKey"]["x"] ← STRING(pk_sign_eph.x);
31  payload["clientSignaturePublicKey"]["y"] ← STRING(pk_sign_eph.y);
32  endpoint ← server_url + "/session/exchange?userId=" + user_id;
33  response ← HTTP_POST(endpoint, payload, headers);
```



```
1 if response.status_code ≠ 200 then
2     OUTPUT "Key exchange failed";
3     return FALSE;
4 end
5 data ← JSON_PARSE(response.body);
6 if data.success ≠ TRUE or data.clientSignatureVerified ≠ TRUE then
7     OUTPUT "Server rejected key exchange";
8     return FALSE;
9 end
10 if data.sessionToken ≠ NULL then
11     token ← data.sessionToken;
12     OUTPUT "Session token updated";
13 end
14 state ← KEY_EXCHANGED;
15 OUTPUT "Key exchange successful";
16 return TRUE;
```

Algorithm 2: key_exchange()

3.2.3 Thiết lập phiên (Session Establishment)

Giai đoạn thiết lập phiên là nơi client tính toán shared secret từ trao đổi khóa ECDH và dẫn xuất các khóa mã hóa mà sẽ được sử dụng để bảo vệ tất cả các tin nhắn. Đây là bước chuyển đổi từ các hoạt động mật mã công khai (như ECDH và ECDSA) sang mật mã đối xứng (AES-256).

Khi giai đoạn thiết lập phiên bắt đầu, client phải kiểm tra xem trạng thái có phải KEY_EXCHANGED không. Nếu không, có nghĩa là giai đoạn trao đổi khóa chưa hoàn thành.

Tiếp theo, client tính toán shared secret bằng cách nhân khóa riêng ECDH của nó (sk_client) với khóa công khai ECDH của server (pk_server). Phép nhân này được thực hiện trên đường cong elliptic, kết quả là một điểm trên đường cong. Để lấy shared secret, client chỉ cần lấy tọa độ x của điểm này. Điều quan trọng cần lưu ý là server có thể tính toán cùng một shared secret bằng cách nhân khóa riêng của nó với khóa công khai ECDH của client. Đây là bản chất của ECDH - cả hai bên có thể tính toán cùng một secret mà không cần chia sẻ khóa riêng của chúng với nhau.

Tiếp theo, client sử dụng PBKDF2 (Password-Based Key Derivation Function 2) để dẫn xuất session key từ shared secret. PBKDF2 là một hàm dẫn xuất khóa được thiết kế để có thể chịu đựng các cuộc tấn công brute-force. Nó lặp lại một hàm MAC (HMAC-SHA256 trong trường hợp này) nhiều lần để làm cho việc tính toán khóa trở nên chậm. Cụ thể, client sử dụng salt là 16 bytes của zeros, 1000 lần lặp lại, và chiều dài đầu ra là 32 bytes. Sau khi hoàn thành, client



có được session key sẽ được sử dụng để mã hóa và giải mã tin nhắn.

Cuối cùng, client chuyển sang trạng thái SESSION_ESTABLISHED, cho biết rằng nó đã sẵn sàng để gửi và nhận tin nhắn bảo mật.

```
Input: None
Output: success (Boolean)

1 PROCEDURE
2   if state ≠ KEY_EXCHANGED then
3     | RETURN FALSE;
4   end

5   shared_secret ← ECDH_compute_shared_secret(curve, sk_client, pk_server);
6   OUTPUT “Computed ECDH shared secret”;

7   salt ← ZERO_BYTES(16);
8   session_key ← PBKDF2_HMAC_SHA256(shared_secret, salt, 1000, 32);
9   OUTPUT “Derived session key using PBKDF2”;

10  state ← SESSION_ESTABLISHED;
11  OUTPUT “Session established successfully”;
12  RETURN TRUE;
```

Algorithm 3: establish_session()

3.2.4 Gửi tin nhắn bảo mật (Send Message)

Giai đoạn gửi tin nhắn bảo mật cho phép client gửi tin nhắn tới server với sự đảm bảo rằng tin nhắn được mã hóa để chỉ server có thể đọc được, đồng thời server có thể xác minh rằng tin nhắn thực sự đến từ client. Điều này được thực hiện bằng cách kết hợp AES-256-GCM để mã hóa có xác thực và ECDSA để ký số.

Khi gửi tin nhắn, client trước tiên kiểm tra xem trạng thái hiện tại có phải là SESSION_ESTABLISHED hay không. Nếu không phải, quá trình gửi tin nhắn sẽ bị từ chối vì session key chưa được thiết lập.

Tiếp theo, client tạo một initialization vector (IV) ngẫu nhiên dài 12 bytes. IV này được sử dụng bởi AES-GCM để đảm bảo rằng ngay cả khi cùng một tin nhắn được gửi hai lần với cùng session key, ciphertext sẽ hoàn toàn khác nhau. IV không cần phải bí mật và sẽ được gửi kèm với ciphertext để server có thể giải mã.

Client sau đó mã hóa plaintext bằng AES-256-GCM sử dụng session key và IV. AES-GCM là một chế độ mã hóa có xác thực tích hợp (Authenticated Encryption with Associated Data - AEAD), không chỉ mã hóa tin nhắn mà còn tự động tạo ra một authentication tag dài 16 bytes. Authentication tag này được sử dụng để xác minh rằng ciphertext không bị chỉnh sửa trong quá trình truyền tải. Đầu ra của AES-GCM là ciphertext ghép liền với authentication tag. Client ghép thêm IV vào phía trước để tạo thành encrypted blob hoàn chỉnh, sau đó mã hóa toàn bộ



bằng Base64.

Để đảm bảo tính xác thực nguồn gốc, client ký chuỗi Base64 của encrypted blob bằng khóa ECDSA dài hạn được tạo từ giai đoạn xác thực. Quy trình ký bao gồm: tính hash SHA256 của chuỗi Base64, chuyển đổi hash thành số nguyên, rút gọn modulo order của đường cong, và cuối cùng tính toán giá trị r và s của chữ ký ECDSA. Chữ ký này cho phép server xác minh rằng tin nhắn thực sự được gửi từ client sở hữu khóa riêng tương ứng.

Client chuẩn bị payload JSON chứa session token để xác định phiên làm việc, IV được mã hóa Base64, encrypted message được mã hóa Base64, chữ ký bao gồm các thành phần r và s cùng với hash của thông điệp và thuật toán sử dụng, và khóa công khai ECDSA để server có thể xác minh chữ ký. Payload này được gửi tới server thông qua HTTP POST tới endpoint /message/send.

Khi server nhận được tin nhắn, nó thực hiện xác minh chữ ký ECDSA bằng khóa công khai được cung cấp, sau đó giải mã ciphertext bằng session key mà server cũng đã dẫn xuất được từ shared secret. Do AES-GCM có tích hợp xác thực, quá trình giải mã sẽ tự động kiểm tra authentication tag và thất bại nếu ciphertext bị chỉnh sửa. Nếu server có dữ liệu để phản hồi, nó sẽ mã hóa dữ liệu đó bằng cùng session key theo cấu trúc tương tự (IV 12 bytes ghép với ciphertext và tag) và gửi trong trường encryptedResponse. Client sau đó tách IV từ 12 bytes đầu tiên và giải mã phần còn lại bằng session key để nhận được phản hồi từ server.



Input: plaintext (String)

Output: (success, message_id, decrypted)

1 PROCEDURE

```
2   if state ≠ SESSION_ESTABLISHED then
3       OUTPUT "Invalid state for sending message";
4       RETURN (FALSE, NULL, FALSE);
5   end
6   iv ← RANDOM_BYTES(12);
7   plaintext_bytes ← ENCODE_UTF8(plaintext);
8   ciphertext_with_tag ← AES_GCM_ENCRYPT(session_key, plaintext_bytes, iv);
9   final_encrypted_blob ← iv + ciphertext_with_tag;
10  encrypted_msg_b64 ← BASE64_ENCODE(final_encrypted_blob);
11  msg_to_sign ← encrypted_msg_b64;
12  message_bytes ← ENCODE_UTF8(msg_to_sign);
13  hash_digest ← SHA256(message_bytes);
14  hash_int ← BYTES_TO_INT(hash_digest);
15  hash_int_mod ← hash_int mod curve.order;
16  ECDSA_signcurve, sk_sign, hash_int_mod, r, s;
17  payload ← CREATE_OBJECT();
18  payload[“sessionToken”] ← token;
19  payload[“iv”] ← BASE64_ENCODE(iv);
20  payload[“encryptedMessage”] ← encrypted_msg_b64;
21  payload[“messageSignature”] ← CREATE_OBJECT();
22  payload[“messageSignature”][“r”] ← STRING(r);
23  payload[“messageSignature”][“s”] ← STRING(s);
24  payload[“messageSignature”][“messageHash”] ← STRING(hash_int_mod);
25  payload[“messageSignature”][“algorithm”] ← “ECDSA-P192”;
26  payload[“clientSignaturePublicKey”] ← CREATE_OBJECT();
27  payload[“clientSignaturePublicKey”][“x”] ← STRING(pk_sign.x);
28  payload[“clientSignaturePublicKey”][“y”] ← STRING(pk_sign.y);
29  endpoint ← server_url + “/message/send?userId=” + user_id;
30  response ← HTTP_POST(endpoint, payload, headers);
```



```
1 if response.status_code ≠ 200 then
2   OUTPUT "Send message failed";
3   return (FALSE, NULL, FALSE);
4 end
5 data ← JSON_PARSE(response.body);
6 if data.success ≠ TRUE then
7   return (FALSE, NULL, FALSE);
8 end
9 decrypted ← FALSE;
10 if data.encryptedResponse ≠ NULL then
11   OUTPUT "Server sent encrypted response";
12   enc_resp_blob ← BASE64_DECODE(data.encryptedResponse);
13   resp_iv ← EXTRACT_BYTES(enc_resp_blob, 0, 12);
14   resp_ciphertext ← EXTRACT_BYTES(enc_resp_blob, 12, LENGTH(enc_resp_blob) -
15     12);
15   resp_plaintext_bytes ← AES_GCM_DECRYPT(session_key, resp_ciphertext,
16     resp_iv);
16   resp_plaintext ← DECODE_UTF8(resp_plaintext_bytes);
17   OUTPUT "Server reply: " + resp_plaintext;
18   decrypted ← TRUE;
19 end
20 message_id ← data.messageId;
21 return (TRUE, message_id, decrypted);
```

Algorithm 4: send_message(plaintext)

Các hàm mã hóa chính

3.2.5 Các hàm mật mã cơ bản

(a) Tạo cặp khóa ECDH

Để tạo một cặp khóa ECDH, client cần tạo một số ngẫu nhiên trong phạm vi từ 1 tới hàng mục order của đường cong trừ đi 1. Số này là khóa riêng. Khóa công khai được tạo bằng cách nhân khóa riêng này với điểm sinh G của đường cong. Công khai khóa phải nằm trên đường cong elliptic, vì vậy client kiểm tra điều này.



Input: curve (EllipticCurve)
Output: (private_key, public_key)

```
1 PROCEDURE
2     private_key ← RANDOM_INTEGER(1, curve.order - 1);
3     OUTPUT "Generated random private key";
4     public_key ← SCALAR_MULTIPLY(private_key, curve.G);
5     if NOT IS_POINT_ON_CURVE(public_key, curve) then
6         RETURN ERROR;
7     end
8     RETURN (private_key, public_key);
```

Algorithm 5: generate_ECDH_keypair(curve)

(b) Tính ECDH Shared Secret

Shared secret được tính bằng cách nhân khóa riêng ECDH của mình với công khai khóa ECDH của người kia. Kết quả là một điểm trên đường cong. Chúng ta lấy tọa độ x của điểm này và chuyển nó thành bytes.

Input: curve (EllipticCurve), private_key (Integer), peer_public_key (ECPoint)
Output: shared_secret (Bytes)

```
1 PROCEDURE
2     shared_point ← SCALAR_MULTIPLY(private_key, peer_public_key);
3     OUTPUT "Computed shared secret point";
4     if shared_point.is_infinity() then
5         OUTPUT "Error: Shared secret is point at infinity";
6         RETURN ERROR;
7     end
8     coord_length ← GET_COORDINATE_LENGTH(curve);
9     shared_secret ← shared_point.x.to_bytes(coord_length, BIG_ENDIAN);
10    RETURN shared_secret;
```

Algorithm 6: ECDH_compute_shared_secret(curve, private_key, peer_public_key)

(c) Ký ECDSA

ECDSA yêu cầu chọn một số k ngẫu nhiên, tính toán điểm $R = k \times G$, rồi tính toán giá trị $r = R \cdot x \pmod{\text{order}}$ và $s = k^{-1} \times (\text{hash} + r \times sk) \pmod{\text{order}}$. Nếu r hoặc s bằng 0, chúng ta phải thử lại với k khác.



Input: curve (EllipticCurve), private_key (Integer), message_hash (Bytes)

Output: (r, s) - Signature components

1 PROCEDURE

```
2     hash_int ← BYTES_TO_INT(message_hash);
3     hash_bits ← BIT_LENGTH(hash_int);
4     order_bits ← BIT_LENGTH(curve.order);
5     if hash_bits > order_bits then
6         |   hash_int ← hash_int ≫ (hash_bits - order_bits);
7     end
8     hash_int ← hash_int mod curve.order;
9     if hash_int = 0 then
10        |   hash_int ← 1;
11    end
12     max_retries ← 10;
13     for attempt ← 1 to max_retries do
14         |   k ← RANDOM_INTEGER(1, curve.order - 1);
15         |   R ← SCALAR_MULTIPLY(k, curve.G);
16         |   if R.is_infinity() then
17             |       |   continue;
18         end
19         |   r ← R.x mod curve.order;
20         |   if r = 0 then
21             |       |   continue;
22         end
23         |   k_inv ← MODULAR_INVERSE(k, curve.order);
24         |   s ← (k_inv · (hash_int + r · private_key)) mod curve.order;
25         |   if s = 0 then
26             |       |   continue;
27         end
28         |   OUTPUT "Generated valid ECDSA signature";
29         |   RETURN (r, s);
30     end
31     OUTPUT "Failed to generate signature after max retries";
32     RETURN ERROR;
```

Algorithm 7: ECDSA_sign(curve, private_key, message_hash)

3.2.6 Implementation Code

Hệ thống được triển khai bằng Python với cấu trúc module như sau:

- `client_smc.py`: Lớp ClientSMC chứa logic giao thức chính



- `crypto_utils.py`: Các hàm mật mã (ECDH, ECDSA, AES-GCM, PBKDF2)

(a) Xác thực (Authentication)

```
1 def authenticate(self):
2     if self.state != "INIT":
3         return False
4
5     # P-192 (SECP192R1)
6     p_hex = "0xfffffffffffffffffffffeffffffffffff"
7     a_hex = "-3"
8     b_hex = "0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1"
9     Gx_hex = "0x188da80eb03090f67cbf20eb43a18800f4ff0af82ff1012"
10    Gy_hex = "0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811"
11    order_hex = "0xffffffffffffffffffff99def836146bc9b1b4d22831"
12
13    curve_params_internal = {
14        "p": p_hex, "a": a_hex, "b": b_hex,
15        "Gx": Gx_hex, "Gy": Gy_hex, "order": order_hex
16    }
17    self.curve = EllipticCurve(**curve_params_internal)
18
19    # ECDSA
20    self.sk_sign, self.pk_sign = ecdh_generate_keypair(self.curve)
21
22    # Payload
23    payload = {
24        "algorithm": "ecdh",
25        "curveParameters": {
26            "p": str(int(p_hex, 16)),
27            "a": a_hex,
28            "b": str(int(b_hex, 16)),
29            "Gx": str(int(Gx_hex, 16)),
30            "Gy": str(int(Gy_hex, 16)),
31            "order": str(int(order_hex, 16))
32        }
33    }
34
35    endpoint = f"{self.server_url}/session/create?userId={self.user_id}"
36    headers = {"Content-Type": "application/json", "x-user-id": self.user_id}
37    response = requests.post(endpoint, json=payload, headers=headers)
38    data = response.json()
39
40    if not data.get("success", False):
41        return False
42
43    self.token = data.get("sessionToken")
44
45    if "serverPublicKey" in data:
```



```
46     server_pub_json = data["serverPublicKey"]
47     sx = int(server_pub_json["x"])
48     sy = int(server_pub_json["y"])
49     self.pk_server = ECPoint(sx, sy, self.curve)
50
51     self.state = "AUTHENTICATED"
52     return True
```

Listing 1: Hàm authenticate()

(b) Trao đổi khóa (Key Exchange)

```
1 def key_exchange(self):
2     if self.state != "AUTHENTICATED":
3         return False
4
5     #Generate Ephemeral ECDH Key Pair
6     sk_c, pk_c = ecdh_generate_keypair(self.curve)
7     self.sk_client = sk_c
8     self.pk_client = pk_c
9
10    pk_c_bytes = serialize_ecdh_public_key(pk_c, self.curve)
11    coord_length = self.curve.get_coord_length() # 24 bytes cho P-192
12    x_coord = pk_c_bytes[1:1+coord_length]
13    y_coord = pk_c_bytes[1+coord_length:]
14
15    client_public_key = {
16        "x": str(int.from_bytes(x_coord, 'big')),
17        "y": str(int.from_bytes(y_coord, 'big'))
18    }
19
20    # Message & Hash
21    message_str = json.dumps(client_public_key, separators=(',', ':'))
22    message_bytes = message_str.encode('utf-8')
23    hash_digest_raw = hash_sha256(message_bytes)
24    hash_int = int.from_bytes(hash_digest_raw, 'big')
25    hash_int_mod = hash_int % self.curve.order
26    order_bytes_len = (self.curve.order.bit_length() + 7) // 8
27    hash_for_sign = hash_int_mod.to_bytes(order_bytes_len, 'big')
28
29    #Sign
30    r_int, s_int = ecdsa_sign(self.curve, sk_sign, hash_for_sign)
31
32    payload = {
33        "sessionToken": self.token,
34        "clientPublicKey": client_public_key,
35        "clientPublicKeySignature": {
36            "r": str(r_int),
37            "s": str(s_int),
```



```
38         "messageHash": str(hash_int_mod),
39         "algorithm": "ECDSA-P192"
40     },
41     "clientSignaturePublicKey": {
42         "x": str(pk_sign.x),
43         "y": str(pk_sign.y)
44     }
45 }
46
47 endpoint = f'{self.server_url}/session/exchange?userId={self.user_id}'
48 headers = {"Content-Type": "application/json", "x-user-id": self.user_id}
49 response = requests.post(endpoint, json=payload, headers=headers)
50 data = response.json()
51
52 if not data.get("clientSignatureVerified"):
53     return False
54
55 if "sessionToken" in data:
56     self.token = data["sessionToken"]
57
58 self.state = "KEY_EXCHANGED"
59 return True
```

Listing 2: Hàm key_exchange()

(c) Thiết lập phiên (Session Establishment)

```
1 def establish_session(self):
2     if self.state != "KEY_EXCHANGED":
3         return False
4
5     shared_secret = ecdh_compute_shared_secret(
6         self.curve, self.sk_client, self.pk_server)
7
8     salt = b'\x00' * 16 # 16 bytes zeros
9     self.session_key = kdf_pbkdf2(
10         shared_secret, salt, iterations=1000, length=32)
11     # session_key = 32 bytes cho AES-256
12
13     self.state = "SESSION_ESTABLISHED"
14     return True
```

Listing 3: Hàm establish_session()

(d) Gửi tin nhắn (Send Message)

```
1 def send_message(self, plaintext):
2     if self.state != "SESSION_ESTABLISHED":
3         return False, None, False
4
```



```
5   from crypto_utils import aes_256_gcm_encrypt, aes_256_gcm_decrypt
6
7   # Encrypt (AES-GCM)
8   iv = random_bytes(12) # 12 bytes IV
9   plaintext_bytes = plaintext.encode('utf-8')
10  ciphertext_with_tag = aes_256_gcm_encrypt(
11      self.session_key, plaintext_bytes, iv)
12
13  #IV || ciphertext || tag
14  final_encrypted_blob = iv + ciphertext_with_tag
15  encrypted_msg_b64 = base64_encode(final_encrypted_blob)
16
17  # Sign
18  msg_to_sign_str = encrypted_msg_b64
19  message_bytes = msg_to_sign_str.encode('utf-8')
20  hash_digest_raw = hash_sha256(message_bytes)
21
22  hash_int = int.from_bytes(hash_digest_raw, 'big')
23  hash_int_mod = hash_int % self.curve.order
24
25  order_bytes_len = (self.curve.order.bit_length() + 7) // 8
26  hash_for_sign = hash_int_mod.to_bytes(order_bytes_len, 'big')
27
28  r, s = ecdsa_sign(self.curve, self.sk_sign, hash_for_sign)
29
30  payload = {
31      "sessionToken": self.token,
32      "iv": base64_encode(iv),
33      "encryptedMessage": encrypted_msg_b64,
34      "messageSignature": {
35          "r": str(r), "s": str(s),
36          "messageHash": str(hash_int_mod), "algorithm": "ECDSA-P192"
37      },
38      "clientSignaturePublicKey": {
39          "x": str(self.pk_sign.x), "y": str(self.pk_sign.y)
40      }
41  }
42
43  endpoint = f"{self.server_url}/message/send?userId={self.user_id}"
44  headers = {
45      "Authorization": f"Bearer {self.token}",
46      "Content-Type": "application/json",
47      "x-user-id": self.user_id
48  }
49  response = requests.post(endpoint, json=payload, headers=headers)
50  data = response.json()
51
52  if not data.get("success", False):
```



```
53     return False, None, False
54
55     # Decrypt
56     decrypted_success = False
57     if "encryptedResponse" in data:
58         enc_resp_b64 = data["encryptedResponse"]
59         enc_resp_blob = base64_decode(enc_resp_b64)
60
61         resp_iv = enc_resp_blob[:12]
62         resp_ciphertext = enc_resp_blob[12:]
63
64         resp_plaintext_bytes = aes_256_gcm_decrypt(
65             self.session_key, resp_ciphertext, resp_iv)
66         resp_plaintext = resp_plaintext_bytes.decode('utf-8')
67         decrypted_success = True
68
69     msg_id = data.get("messageId", 0)
70     return True, msg_id, decrypted_success
```

Listing 4: Hàm send_message()

3.2.7 Kết quả thử nghiệm

Sau khi hoàn tất việc cài đặt lại giao thức phía Client, nhóm tiến hành chạy thử nghiệm để kết nối tới Server và gửi các tin nhắn yêu cầu. Kết quả bên dưới cho thấy Client đã thực hiện thành công toàn bộ quy trình bắt tay (handshake), thiết lập khóa phiên và giải mã tin nhắn bảo mật.

Kết quả của terminal với messages:

```
1 test_messages = [
2     "Hello, Secure Chat!",
3     "This is message 2",
4     "Final test message"
5 ]
```



```
1 =====
2     SMC Client Re-implementation
3 =====
4
5 Configuration:
6     User ID:      group-1
7     Server URL:   https://crypto-assignment.dangduongminhnhat2003.workers.dev
8
9 =====
10    Initialize Client
11 =====
12 [] Client initialized
13     State: INIT
14
15 =====
16     Authenticate
17 =====
18 [AUTH] Created curve with p=192 bits
19 [AUTH] Generating long-term ECDSA signing key...
20 [AUTH] Sending to https://crypto-assignment.dangduongminhnhat2003.workers.dev/
21     session/create?userId=group-1
22 [AUTH] Captured Server ECDH Public Key
23 [AUTH] Authenticated successfully
24
25 =====
26     Key Exchange (ECDH P-192)
27 =====
28 [KX] Sending payload...
29 [KX] Server verified client signature successfully
30 [KX] Session Token updated
31
32 =====
33     Establish Session
34 =====
35 [SESSION]     Session established (PBKDF2 Key Derived)
36
37 =====
38     Send Secure Messages
39 =====
40 [SEND] Sending message: 'Hello, Secure Chat!'
41 [SEND] Server responded with encrypted data!
42 [SEND] SERVER REPLY: Hey there, fellow cryptographer!
43 [SEND] Sending message: 'This is message 2'
44 [SEND] Server responded with encrypted data!
45 [SEND] SERVER REPLY: I'm ageless... born from pure JavaScript energy      (
46     somewhere in 2024).
47 [SEND] Sending message: 'Final test message'
48 [SEND] Server responded with encrypted data!
```



```
47 [SEND] SERVER REPLY: I'm ageless... born from pure JavaScript energy      (
    somewhere in 2024).
48
49 [] Sent 3/3 messages successfully
50 [] Attempted to decrypt 3 server response(s)
51
52 =====
53     Summary
54 =====
55
56 [] Client State:          SESSION_ESTABLISHED
57 [] Messages Sent:         3
58
59 [] FULL FLOW COMPLETED SUCCESSFULLY
```

3.3 Exploitation & Proof-of-Concept

3.3.1 Xác định lô hổng

Đối với **userId=group-1**, ta tóm tắt lại luồng hoạt động chính của ứng dụng SMC như sau:

- (a) Client gửi tên thuật toán trao đổi khóa (ECDH) và các tham số đường cong (P-192) cho Server.
 - (b) Client và Server sử dụng ECDH private key (dài hạn) để tính ra public key rồi gửi cho nhau.
 - (c) Client và Server cùng tính shared secret bằng private key của mình và public key của đối phương.
 - (d) Session key được sinh từ shared secret bằng thuật toán dẫn xuất khóa (PBKDF2-HMAC-SHA256).
 - (e) Client và Server sử dụng session key để mã hóa và giải mã các tin nhắn bằng thuật toán AES-GCM.

Ở gói tin đầu tiên, Client gửi các tham số của đường cong P-192 cho Server. Đây là một hành vi dư thừa và bất thường vì thông thường Client chỉ cần gửi mã tên của đường cong thỏa thuận, còn Server sẽ đọc ra các tham số đã được cài đặt sẵn trong code để tính toán.

Điều này dẫn đến câu hỏi liệu Server có trực tiếp sử dụng các giá trị này để tính toán public key? Nói cách khác, liệu Server có đang cho phép Client lựa chọn đường cong tùy ý?

Ta sẽ kiểm chứng bằng cách sử dụng Burp Suite để gửi một gói tin create với các tham số của một đường cong Elliptic khác.

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
<pre>1 POST /session/create?userId=group-1 HTTP/2 2 Host: crypto-assignment.dangdungominhnhat2003.workers.dev 3 User-Agent: python-requests/2.31.0 4 Accept-Encoding: gzip, deflate, br 5 Accept: */* 6 Connection: keep-alive 7 Content-Type: application/json 8 X-User-Id: group-1 9 Content-Length: 110 10 11 { 12 "algorithm": "ecdh", 13 "curveParameters": { 14 "p": "17", 15 "a": "1", 16 "b": "3", 17 "Gx": "2", 18 "Gy": "8", 19 "order": "17" 20 } 21 }</pre>	<pre>1 HTTP/2 200 OK 2 Date: Mon, 08 Dec 2025 10:15:02 GMT 3 Content-Type: application/json 4 Access-Control-Allow-Origin: * 5 Access-Control-Allow-Headers: Content-Type, X-Session-Token 6 Access-Control-Allow-Methods: GET, POST, OPTIONS 7 Vary: accept-encoding 8 Report-To: [{"group": "cf-nel", "max_age": 604800, "endpoints": [{"url": "https://a.nel.cloudflare.com/report/v4?e=OabBxQSNKTAuhsf03pWm1kFRSMYQk2FuwN%2FC2j7WM1Wz2BS5cbw1%2B51AbCvDyPTd0UJH%2B%2FvBaZ61NfTqebBdQdgvecqQY1kp9zFR9ir4f211u3CylLwdrqrAPT+ha20dAca%2Bsr%2B4V4K1sNA%3D%3D%"}}] 9 Nel: {"report-to": "cf-nel", "success_fraction": "0.1", "max_age": 604800} 10 Server: cloudflare 11 Cf-Ray: 9aab7cd7bb0b42b-HKG 12 Alt-Svc: h3=:443; ma=6400 13 14 { 15 "success": true, 16 "sessionToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkXVCi9j.yJpcMj0LTzWn1cmVdaGF0IiwiZWFIoixNzY1Mtg4OTAyLcJ1eHaiOjE3h0NlxZw0CkyD1sNtY1lMdjh3W4tELC1zawQ1Q1JmRmWe4MGuZTdm7uhYzhkYjVLMemQWNMjMyQMDm5yTcv1NlZw1Mzg2VG1MjNhZ0V1MjMyN21N0d1IiwiZwhnb3pdHf1oiZwhwAc1S1nBlYmx0ptleSe1gey14joiMyisInki10In0esImuy35ScR1ZEPhdGElQ1Ly1hnQ19Qz2e0MFLPsdVazd21WdQ1bBWS1QeQfb19N203PvH2m7tskSYatwsNv01LVZNhu1bz2jPQ842kdwLxTa2rShpRoTzYahkS05sRu1NnhoSFZSTFxZk10050y0p/enVd185CE9oewU8dkVuelNYS11GvmtSdm1lZvHHCsc5dLtzNaMEi1wL13dJUVcbyd2zw6lPwfRfwzd2aJnZhdloNvxsX015MwpV1ECRwxRnRtTWuZ20rg1hR1Z0TmPhyllnGt1vhra0000xpkb1LxwdRjLgxwvHRCTXfTEF2RjRzEwKbzb1jBpBXRlZipQVLPRMNs2y1slnyZwF02WPB1C16Mtc2NF4E0Okw1AxMyw1bGf2dJg1L2aXPS1joxNzY1Mtg4OTAyMDEf2q.U6aqgc4e0ctGtg4myWf1PLKu0nxTExRv3Km6gX8", 17 "algorithm": "ecdh", 18 "serverPublicKey": { 19 "x": "3", 20 "y": "4" 21 }, 22 "signatureSupported": true, 23 "serverSignaturePublicKey": { 24 "x": "8", 25 "y": "9" 26 }, 27 "sessionSignature": { 28 "x": "16", 29 "y": "13", 30 "messageHash": "9", 31 "algorithm": "ECDSA-P192" 32 }, 33 "signatureAlgorithm": "ECDSA-P192" 34 }</pre>



Kết quả cho thấy, Server chấp nhận một đường cong bậc 17 và thực hiện tính public key $P=kG$ trên đó (k là private key). Vì kết quả P chỉ có thể là 1 trong 17 điểm, ta dễ dàng vét cạn với 17 giá trị k' để tìm ra phương trình: $k \equiv k' \pmod{17}$.

```
p = 17
a = 1
b = 3
Gx, Gy = 2, 8
Px, Py = 3, 4

E = EllipticCurve(GF(p), [a, b])
order = E.order()

G = E(Gx, Gy)
P = E(Px, Py)

Q = E(0)
for k in range(order):
    if Q == P:
        print(f"k = {k} (mod {order})")
        break
    Q += G

k = 9 (mod 17)
```

Đến đây, ta có thể kết luận rằng ứng dụng đang tồn tại lỗ hổng *Tấn công đường cong không hợp lệ* (Invalid curve attack), cho phép kẻ tấn công gửi các đường cong Elliptic với nhóm cyclic tùy ý lên Server và nhận về kết quả của phép nhân private key với điểm sinh của nhóm đó. Nếu chọn được các nhóm cyclic yếu, nghĩa là bậc của nó đủ nhỏ để có thể vét cạn giá trị k , kẻ tấn công sẽ thu thập được các phương trình đồng dư của k với modulo nhỏ, cuối cùng khôi phục lại private key của Server và giải mã bất cứ tin nhắn nào. Điều này là do private key được Server sử dụng trực tiếp cho việc dẫn xuất session key với mọi Client.

3.3.2 Khai thác lỗ hổng

Để khai thác lỗ hổng này, ta sẽ lập trình để tự động hóa việc gửi gói tin create với các nhóm cyclic bậc p_i khác nhau, giải bài toán logarit rời rạc để tìm ra các giá trị k' tương ứng và cuối cùng tính k bằng *Định lí thặng dư Trung Hoa* (Chinese Remainder Theorem).

Trước hết, cần trả lời 2 câu hỏi sau:



- (a) Cần ít nhất bao nhiêu phương trình đồng dư khác nhau để khôi phục được giá trị có độ lớn 192 bit?
- (b) Làm sao để tìm ra nhóm cyclic của đường cong với bậc nguyên tố?

Đối với câu hỏi 1: Định lý Thăng dư Trung Hoa phát biểu rằng một hệ phương trình đồng dư của ẩn x theo các modulo p_1, p_2, \dots, p_n đối một nguyên tố cùng nhau.

$$\begin{cases} x \equiv a_1 \pmod{p_1} \\ x \equiv a_2 \pmod{p_2} \\ \vdots \\ x \equiv a_n \pmod{p_n} \end{cases}$$

Thì hệ phương trình này có nghiệm duy nhất theo modulo $M = p_1 \times p_2 \times \dots \times p_n$.

Như vậy, để khôi phục được 192 bit của private key, tức là tìm k duy nhất theo modulo 2^{192} , thì ta cần thu thập số phương trình đồng dư sao cho $\prod p_i \geq 2^{192}$. Để dễ dàng thỏa mãn điều kiện đối một nguyên tố cùng nhau, ta sẽ chọn các p_i nguyên tố. Theo tính toán, cần 32 phương trình đồng dư nếu chọn p bắt đầu từ 13.

Đối với câu hỏi 2: Có nhiều cách để giải quyết, nhưng ở đây ta sẽ chọn cách đơn giản, dễ hiện thực nhất vì ta có thể lựa chọn các nhóm điểm $E(F_p)$ với p nhỏ để tính toán nhanh.

Định lý Hasse về đường cong Elliptic E trên trường hữu hạn F_p phát biểu rằng $|N - (q+1)| \leq 2\sqrt{q}$, với N là số điểm trên $E(F_p)$ bao gồm điểm vô cực. Do đó, nếu ta thực hiện tìm kiếm trên p nhỏ thì bậc N sẽ không vượt quá $p + 1 + 2\sqrt{q}$.

Cụ thể như sau:

- (a) Chuẩn bị 1 mảng các p nguyên tố nhỏ tăng dần. Đây là các trường hữu hạn F_p mà ta thực hiện tìm đường cong có bậc nguyên tố.
- (b) Duyệt các đường cong E không suy biến với a, b trong khoảng $[0, p-1]$, chọn 1 điểm bất kì thuộc $E(F_p)$ làm điểm sinh G . Lần lượt cộng G vào cho đến khi gặp điểm vô cực, ta sẽ có được bậc N của nhóm cyclic sinh bởi G (Lưu ý rằng 1 nhóm $E(F_p)$ có thể chứa 1 hoặc nhiều nhóm cyclic với các bậc khác nhau).
- (c) Kiểm tra nếu N là nguyên tố và không trùng với modulo của các phương trình đồng dư trước đó thì chọn nhóm cyclic này.
- (d) Thực hiện gửi các tham số p, a, b, G, N đến Server. Thực hiện vét cạn k' trong khoảng $[1, N-1]$ ta có được phương trình đồng dư $k \equiv k' \pmod{N}$. Lưu lại cặp (k', N) để tính k sau này và để bỏ qua các nhóm cyclic trùng N trong quá trình tìm kiếm.
- (e) Cuối cùng, khi đã thu thập đủ số phương trình đồng dư sao cho $\prod N_i \geq 2^{192}$, ta dùng tìm kiếm và áp dụng Định lí Thăng dư Trung Hoa để tính k (private key của Server).



Kết quả chạy code tấn công:

```
[*] Searching on F_13: Order = 3, a=1, b=2, G=(9,5) -> [FAIL]
[*] Searching on F_17: Order = 17, a=1, b=3, G=(16,1) -> [OK] k = 9 (mod 17)
[*] Searching on F_19: Order = 19, a=1, b=4, G=(14,8) -> [OK] k = 1 (mod 19)
[*] Searching on F_23: Order = 29, a=1, b=4, G=(22,5) -> [OK] k = 19 (mod 29)
[*] Searching on F_29: Order = 11, a=1, b=4, G=(27,9) -> [FAIL]
[*] Searching on F_31: Order = 41, a=1, b=3, G=(30,1) -> [OK] k = 2 (mod 41)
[*] Searching on F_37: Order = 13, a=1, b=3, G=(36,1) -> [OK] k = 1 (mod 13)
[*] Searching on F_41: Order = 47, a=1, b=5, G=(39,6) -> [OK] k = 32 (mod 47)
[*] Searching on F_43: Order = 37, a=1, b=5, G=(41,9) -> [OK] k = 7 (mod 37)
[*] Searching on F_47: Order = 59, a=1, b=4, G=(46,7) -> [OK] k = 50 (mod 59)
[*] Searching on F_53: Order = 5, a=1, b=7, G=(48,6) -> [OK] k = 1 (mod 5)
[*] Searching on F_59: Order = 11, a=1, b=3, G=(58,1) -> [FAIL]
[*] Searching on F_61: Order = 11, a=1, b=3, G=(60,1) -> [FAIL]
[*] Searching on F_67: Order = 7, a=1, b=4, G=(60,18) -> [FAIL]
[*] Searching on F_71: Order = 79, a=1, b=8, G=(70,19) -> [OK] k = 73 (mod 79)
[*] Searching on F_73: Order = 7, a=1, b=9, G=(71,27) -> [OK] k = 2 (mod 7)
[*] Searching on F_79: Order = 43, a=1, b=1, G=(76,34) -> [OK] k = 7 (mod 43)
[*] Searching on F_83: Order = 23, a=1, b=14, G=(82,26) -> [OK] k = 16 (mod 23)
[*] Searching on F_89: Order = 101, a=1, b=9, G=(87,34) -> [OK] k = 63 (mod 101)
[*] Searching on F_97: Order = 97, a=1, b=1, G=(96,22) -> [OK] k = 87 (mod 97)
[*] Searching on F_101: Order = 109, a=1, b=46, G=(99,6) -> [OK] k = 16 (mod 109)
[*] Searching on F_103: Order = 103, a=1, b=4, G=(102,38) -> [OK] k = 10 (mod 103)
[*] Searching on F_107: Order = 107, a=1, b=4, G=(105,23) -> [OK] k = 26 (mod 107)
[*] Searching on F_109: Order = 53, a=1, b=35, G=(107,5) -> [OK] k = 14 (mod 53)
[*] Searching on F_113: Order = 113, a=1, b=9, G=(112,32) -> [OK] k = 30 (mod 113)
[*] Searching on F_127: Order = 71, a=1, b=5, G=(125,54) -> [OK] k = 12 (mod 71)
[*] Searching on F_131: Order = 137, a=1, b=9, G=(130,20) -> [OK] k = 43 (mod 137)
[*] Searching on F_137: Order = 157, a=1, b=17, G=(136,17) -> [OK] k = 131 (mod 157)
[*] Searching on F_139: Order = 3, a=1, b=1, G=(133,14) -> [FAIL]
[*] Searching on F_149: Order = 83, a=1, b=9, G=(148,56) -> [OK] k = 16 (mod 83)
[*] Searching on F_151: Order = 167, a=1, b=3, G=(150,1) -> [OK] k = 10 (mod 167)
[*] Searching on F_157: Order = 31, a=1, b=3, G=(156,1) -> [OK] k = 9 (mod 31)
[*] Searching on F_163: Order = 163, a=1, b=4, G=(156,44) -> [OK] k = 26 (mod 163)
[*] Searching on F_167: Order = 181, a=1, b=3, G=(166,1) -> [OK] k = 134 (mod 181)
[*] Searching on F_173: Order = 11, a=1, b=15, G=(172,79) -> [FAIL]
[*] Searching on F_179: Order = 89, a=1, b=6, G=(178,2) -> [OK] k = 9 (mod 89)
[*] Searching on F_181: Order = 199, a=1, b=5, G=(180,33) -> [OK] k = 183 (mod 199)
[*] Searching on F_191: Order = 197, a=1, b=6, G=(190,2) -> [OK] k = 58 (mod 197)
[*] Searching on F_193: Order = 193, a=1, b=5, G=(192,14) -> [OK] k = 26 (mod 193)
[*] Searching on F_197: Order = 211, a=1, b=28, G=(196,82) -> [OK] k = 1 (mod 211)
```

[+] Collected enough modulo equations for 192-bit key recovery.

[*] Number of equations: 33



[*] Total bit coverage: 198 bits

[+] RECOVERED PRIVATE KEY: 0x4b9a95c2326135c622fa761e25bfd1a8070dc3f61e6bcc6b

Một số nhóm cyclic bị FAIL là do private key chia hết cho bậc của nhóm này, dẫn đến tạo ra điểm vô cực. Server gặp lỗi khi tính toán và trả về.

The screenshot shows the Network tab of a browser's developer tools. A POST request is made to the URL `/session/create?userId=group-1`. The response is a 500 Internal Server Error with the following details:

```
HTTP/2 500 Internal Server Error
Date: Tue, 09 Dec 2025 06:25:44 GMT
Content-Type: application/json
Content-Length: 83
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Content-Type, X-Session-Token
Access-Control-Allow-Methods: GET, POST, OPTIONS
Vary: accept-encoding
Report-To:
{"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?<snip>"}]}
NEL: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
Server: cloudflare
Cf-Ray: 9ab26d4babdc0460-HKG
Alt-Svc: h3=":443"; ma=86400
15 {
    "success":false,
    "error":"TypeError: Cannot read properties of null (reading 'x')"
}
```

Để xác nhận private key tìm được là chính xác, ta sẽ sử dụng để tính toán public key và so sánh với public key trả về của Server.

```
[AUTH] Created curve with p=192 bits
[AUTH] Generating long-term ECDSA signing key...
[AUTH] Sending to
→ https://crypto-assignment.dangduongminhnhat2003.workers.dev/session/create?userId=group-1
[AUTH] Captured Server ECDH Public Key
[AUTH] Authenticated successfully
=====
[*] Server ECDH private key: 0x4b9a95c2326135c622fa761e25bfd1a8070dc3f61e6bcc6b
-----
[*] Calculated PubKey (x): 0x1f0aee0d86a8f12d26e337e03f177f5b028d930431dc1bd6
[*] Calculated PubKey (y): 0xc4eb307befa1f5689829919c9339ea5a3b2fc31b632efee
-----
[*] REAL Server PubKey (x): 0x1f0aee0d86a8f12d26e337e03f177f5b028d930431dc1bd6
[*] REAL Server PubKey (y): 0xc4eb307befa1f5689829919c9339ea5a3b2fc31b632efee
-----
[SUCCESS] Public key matches!
```



3.3.3 Khắc phục lỗ hổng

Lỗ hổng *Invalid curve attack* khai thác việc Server tin tưởng sử dụng tham số đường cong gửi từ Client để tính toán public key, do đó cách hiệu quả nhất để giải quyết chính là thay vì sử dụng tham số từ Client gửi, hai bên chỉ trao đổi mã tên của đường cong (ví dụ P-192, P-256, Ed25519), sau đó sẽ tính toán bằng tham số cài đặt sẵn trong code.

```
POST /session/create?userId=group-1 HTTP/1.1
Host: crypto-assignment.dangduongminhnhat2003.workers.dev
User-Agent: python-requests/2.28.2
Accept-Encoding: gzip, deflate, br
Accept: /*
Connection: keep-alive
Content-Type: application/json
x-user-id: group-1
Content-Length: 109

{"algorithm": "ecdh", "curve": "P-256"}
```

Việc lựa chọn đường cong cũng quan trọng không kém vì nếu sử dụng các đường cong tự tạo hoặc đến từ nguồn không tin cậy, hệ thống sẽ có nguy cơ bị lỗ hổng *Small subgroup attack* hoặc *Smart's attack*, khai thác điểm yếu về mặt toán học trên các đường cong và rút gọn về các bài toán dễ bị phá vỡ hơn. Do đó, cần phải lựa chọn các đường cong an toàn được khuyến nghị và chuẩn hóa bởi các tổ chức lớn và giới khoa học có chuyên môn.

Một biến thể khác của lỗ hổng Invalid curve attack xảy ra trong bước trao đổi khóa, khi Client gửi public key và Server thực hiện tính toán ra giá trị shared secret dùng để mã hóa tin nhắn. Nếu Server không thực hiện kiểm tra xem public key có nằm trên đường cong đã thống nhất hay không mà vẫn thực hiện tính toán thì kẻ tấn công có thể gửi một điểm thuộc đường cong yếu có cùng p, a với đường cong gốc, chỉ thay đổi giá trị b để tạo ra đường cong chứa các nhóm cyclic có bậc nhỏ, tạo điều kiện cho việc vét cạn k' . Khác với lỗ hổng trong bài tập lớn này một chút, thay vì nhận về public key để vét cạn tìm k' , thì kẻ tấn công sẽ vét cạn k' để tính toán khóa mã hóa tin nhắn và gửi một gói tin mà hắn biết rằng Server sẽ phản hồi khác biệt. Lý do mà mặc dù giá trị b không còn khớp với giá trị cài đặt trên Server nhưng shared secret tính toán ra vẫn thuộc đường cong yếu của kẻ tấn công lựa chọn là vì trong công thức cộng điểm và nhân đôi điểm không hề dùng đến tham số b . Do đó trước khi thực hiện tính toán, Server phải kiểm tra xem public key nhận được có thuộc đường cong gốc hay không.



Tài liệu tham khảo

- [1] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Self-published, 2015. <https://crypto.stanford.edu/~dabo/cryptobook/>.
- [2] Group 3. Source code for co3038 assignment hk251. <https://github.com/NiaLliceH/AdvancedCryptography-Assignment-HK251>, 2025.
- [3] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. Practical invalid curve attacks on tls-ecdh. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Vienna, Austria, 2015. Springer.
- [4] Nadim Kobeissi. Applied cryptography – part 1: Provable security, section 1.8: Elliptic curves and digital signatures. Lecture Slides, CMPS 297AD/396AI, 2025. <https://appliedcryptography.page>.