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

Mật mã học và mã hoá thông tin (CO3083)

BÁO CÁO BÀI TẬP LỚN

Giảng viên hướng dẫn: TS. Nguyễn An Khương
Lớp: L01
Thành viên: Nguyễn Lê Anh Đức - 2210796
Nguyễn Vũ Tường - 2313834

Thành Phố Hồ Chí Minh, Tháng 12 Năm 2025

BẢNG PHÂN CÔNG CÔNG VIỆC

No.	Họ tên	MSSV	Công việc	Đánh giá
1	Nguyễn Lê Anh Đức	2210796	Viết báo cáo, thực hiện bài 1	100%
2	Nguyễn Vũ Tường	2313834	Viết báo cáo, thực hiện bài 2, bài 3	100%

Bảng 1: Bảng phân công công việc

Mục lục

1 Lattices **4**

1.1 Báo cáo thuật ngữ 4

1.2 Tìm đa thức tối tiểu của $\alpha = 4 + \sqrt[3]{7}$ 5

1.3 Tìm X từ d-chữ số thập phân của $\sqrt{X}$ 9

2 Multi-time pad exploitation **11**

2.1 Tổng quan 11

2.2 Nguyên lý hoạt động 11

3 SMC Exploitation **13**

3.1 Interception & API Analysis 13

3.1.1 Tổng quan về các Endpoint API 13

3.1.2 Phân tích Chi tiết Request và Response 16

3.1.2.1 Endpoint: `/session/create` 16

3.1.2.2 Endpoint: `/session/exchange` 18

3.1.2.3 Endpoint: `/message/send` 19

3.1.2.4 Endpoint: `/session/delete` 21

3.1.2.5 Endpoint: `/session/status` 22

3.2 Re-implementation & Protocol Reconstruction 24

3.2.1 Luồng Giao thức Bảo mật 24

3.2.2 Mô tả Mã giả (Pseudocode) các Giải thuật Mật mã 25

3.2.2.1 Cơ chế Trao đổi khóa (ECDH & Key Derivation) 25

3.2.2.2 Cơ chế Mã hóa (AES-256-CBC) 26

3.2.2.3 Cơ chế Ký và Xác thực (ECDSA) 26

3.2.3 Quản lý Trạng thái Client 27

3.2.4 Ánh xạ API Endpoint và Hàm xử lý 27

3.2.5 Chạy thử 28

3.3 Exploitation & Proof-of-Concept 29
3.3.1 Phân tích Lỗ hổng Kênh kề: Rò rỉ Độ dài (Side-Channel Leakage) 29
3.3.1.1 Cơ chế 29
3.3.1.2 Kịch bản Khai thác Thực nghiệm (PoC) 29
3.3.1.3 Đánh giá Tác động 30
3.3.2 Phân tích Lỗ hổng: Tấn công Padding Oracle (Padding Oracle Attack) 31
3.3.2.1 Cơ chế Kỹ thuật và Nguyên nhân 31
3.3.2.2 Nguyên lý Khôi phục Dữ liệu (Byte-by-Byte Decryption) 31
3.3.2.3 Proof-of-Concept 32
3.3.2.4 Kết luận 33

External link

1. Github repository https://github.com/Axiza2k5/ACCT_ASM
2. Youtube demo link https://youtu.be/MuB8KgGaTLo

1 Lattices

1.1 Báo cáo thuật ngữ

1. **Vector spaces (Không gian vector):** là tập Là tập hợp các vector có thể được cộng với nhau và nhân với các số vô hướng (scalars) mà vẫn nằm trong tập hợp đó (tính đóng). Ví dụ: R^n
2. **Linear combinations (Tổ hợp tuyến tính):** Một vector v là tổ hợp tuyến tính của các vector v_1, v_2, ..., v_k nếu nó có dạng: v = c_1 v_1+ c_2 v_2 + ... + c_k v_k (với c_i là các hệ số).
3. **Independence (Độc lập tuyến tính):** Một tập hợp các vector được gọi là độc lập nếu không có vector nào trong số đó có thể được biểu diễn thành tổ hợp tuyến tính của các vector còn lại.
4. **Bases (Cơ sở):** Một tập hợp các vector độc lập tuyến tính có thể tạo ra (span) toàn bộ không gian vector đó. Số lượng vector trong cơ sở chính là chiều (dimension) của không gian.
5. **Orthogonal and orthonormal basis:**
    - $Orthogonal(Trcgiao)$: Các vector trong cơ sở đôi một vuông góc với nhau (tích vô hướng bằng 0).
    - $Orthonormal(Trcchun)$: Là cơ sở trực giao và độ dài của mỗi vector đều bằng 1.

6. **Lattices (Mạng tinh thể):** Là một tập hợp các điểm rời rạc trong không gian n-chiều, được tạo thành bởi tất cả các tổ hợp tuyến tính **nguyên** (integer linear combinations) của một tập cơ sở.

    - Công thức: $L = a_1v_1 + ... + a_nv_n | a_i \in Z$

7. **Fundamental domains (Miền cơ bản):** Là khối hình học cơ bản (thường là hình hộp song song tạo bởi các vector cơ sở) mà khi lặp lại nó (theo phép tịnh tiến) sẽ lấp đầy toàn bộ không gian.

8. **Shortest Vector Problem (SVP):** Bài toán tìm một vector khác 0 trong mạng tinh thể có độ dài ngắn nhất (theo chuẩn Euclidean). Đây là bài toán khó (NP-hard) làm nền tảng cho mật mã.

9. **Closest Vector Problem (CVP):** Cho một điểm bất kỳ trong không gian (không nhất thiết thuộc mạng), tìm điểm thuộc mạng gần điểm đó nhất.

10. **The Euclidean ball:** Tập hợp các điểm có khoảng cách đến tâm nhỏ hơn hoặc bằng một bán kính r nào đó.

11. **The Gaussian expected shortest length:** Một công thức ước lượng độ dài của vector ngắn nhất trong một mạng tinh thể ngẫu nhiên, dựa trên định lý Minkowski (thường tỉ lệ với căn bậc n của định thức mạng).

12. **The LLL algorithm (Lenstra–Lenstra–Lovász):** Một thuật toán đa thức giúp tìm ra một cơ sở "tốt" (gần như trực giao) cho mạng tinh thể. Nó giải quyết bài toán SVP một cách xấp xỉ (tìm ra vector khá ngắn, tuy chưa chắc là ngắn nhất nhưng đủ dùng).

1.2 Tìm đa thức tối tiểu của $\alpha = 4 + \sqrt[3]{7}$

Đề bài: Cho $\alpha = 4 + \sqrt[3]{7}$ và giá trị xấp xỉ β của α (đúng đến 10 chữ số thập phân). Tìm đa thức tối tiểu $f(x)$ của α bằng cách chuyển về bài toán Lattice.

Bước 1: Dự đoán bậc của đa thức (Hint 1)

Trước khi tìm đa thức $f(x)$, ta cần đoán xem bậc cao nhất của nó là bao nhiêu.

- Nhìn vào số $\alpha = 4 + \sqrt[3]{7}$.
- Ta thấy có căn bậc 3 ($\sqrt[3]{...}$).
- **Suy luận:** Để làm mất căn bậc 3, ta thường phải mũ 3 lên. Vậy đa thức này khả năng cao là **đa thức bậc 3**.

Dạng tổng quát của đa thức cần tìm:

$$f(x) = c_3x^3 + c_2x^2 + c_1x + c_0 \tag{1}$$

Trong đó c_0, c_1, c_2, c_3 là các số nguyên (Integers) mà ta cần tìm.

Bước 2: Quan hệ giữa α và β (Hint 2)

- Ta biết $f(\alpha) = 0$.
- Vì β rất gần với α ($\beta \approx \alpha$), nên $f(\beta)$ sẽ **rất gần với 0**.

$$c_3\beta^3 + c_2\beta^2 + c_1\beta + c_0 \approx 0 \tag{2}$$

- **Mục tiêu:** Tìm các số nguyên c_i sao cho biểu thức trên nhỏ nhất có thể, đồng thời các hệ số c_i cũng không được quá lớn (để tạo thành đa thức tối tiểu đơn giản nhất).

Bước 3: Xây dựng cơ sở Lattice (Hint 3)

Ta sử dụng một hằng số lớn K (ví dụ $K = 10^{10}$ tương ứng với độ chính xác của β) để khuếch đại sai số.

Ta xây dựng một ma trận cơ sở B kích thước 4×5 với các vector hàng như sau:

- Vector ứng với c_0: $b_0 = (1, 0, 0, 0, K \cdot \beta^0)$

- Vector ứng với c_1: $b_1 = (0, 1, 0, 0, K \cdot \beta^1)$
- Vector ứng với c_2: $b_2 = (0, 0, 1, 0, K \cdot \beta^2)$
- Vector ứng với c_3: $b_3 = (0, 0, 0, 1, K \cdot \beta^3)$

Ma trận biểu diễn:

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & K \\ 0 & 1 & 0 & 0 & K\beta \\ 0 & 0 & 1 & 0 & K\beta^2 \\ 0 & 0 & 0 & 1 & K\beta^3 \end{pmatrix}$$

Nếu ta lấy tổ hợp tuyến tính các vector này với các hệ số nguyên (c_0, c_1, c_2, c_3), ta sẽ được một vector tổng V:

$$V = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \cdot B$$

$$V = (c_0, c_1, c_2, c_3, \quad K \cdot \underbrace{(c_0 + c_1\beta + c_2\beta^2 + c_3\beta^3)}_{f(\beta)})$$

Trong Lattice, thuật toán giảm cơ sở (như LLL) sẽ tìm vector ngắn nhất. Một vector ngắn nghĩa là:

1. Các thành phần c_0, c_1, c_2, c_3 phải nhỏ (các hệ số đơn giản).
2. Thành phần cuối cùng $K \cdot f(\beta)$ phải nhỏ. Vì K rất lớn, nên bắt buộc $f(\beta)$ phải **cực kỳ nhỏ (xấp xỉ 0)**.

Bước 4: Dùng thuật toán LLL

Ta dùng thuật toán LLL trên ma trận B để tính ra vector V. kết quả của V sẽ là:

$$V = \Big(c_0, c_1, c_2, c_3, \epsilon\Big)$$

Bước 5: Kết quả Giải tay minh họa logic

Nhẩm bằng đại số, ta có đáp án:

$$\begin{aligned} x &= 4 + \sqrt[3]{7} \\ x - 4 &= \sqrt[3]{7} \\ (x-4)^3 &= 7 \\ x^3 - 12x^2 + 48x - 64 &= 7 \\ x^3 - 12x^2 + 48x - 71 &= 0 \end{aligned}$$

1.3 Tìm X từ d-chữ số thập phân của $\sqrt{X}$

Bài toán này là trường hợp tổng quát của bài toán tìm đa thức tối tiểu bậc 2. Ta tìm $X \in \mathbb{Z}$ là hệ số của đa thức $f(x) = x^2 - X = 0$.

1. Thiết lập Vấn đề

- Cho β, xấp xỉ của $\sqrt{X}$ đến d chữ số.
- Ta tìm vector hệ số nguyên $\mathbf{c} = (c_0, c_1, c_2)$ sao cho $c_0 \cdot 1 + c_1 \cdot \beta + c_2 \cdot \beta^2 \approx 0$.
- Vector hệ số cần tìm là $\mathbf{c}_{\text{target}} = (-X, 0, 1)$ (với $c_0 = -X, c_1 = 0, c_2 = 1$).

2. Xây dựng Lattice (Phương pháp Tổng quát)

Sử dụng ma trận cơ sở B có kích thước $(m+1) \times (m+1)$ với $m = 2$ (tìm 3 hệ số c_0, c_1, c_2). Chọn $N = 10^{d+k}$ (k là hệ số an toàn).

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ N & \lfloor N\beta \rfloor & \lfloor N\beta^2 \rfloor & 0 \end{pmatrix}^T = \begin{pmatrix} 1 & 0 & 0 & N \\ 0 & 1 & 0 & \lfloor N\beta \rfloor \\ 0 & 0 & 1 & \lfloor N\beta^2 \rfloor \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

(Trong thực tế, ta sử dụng ma trận 4×4 với $c_3 = 0$ ngầm định).

3. Áp dụng LLL

- Chạy LLL trên ma trận B.
- Vector ngắn nhất $\mathbf{v}_{LLL} = (c_0^*, c_1^*, c_2^*, \epsilon)$ tìm được sẽ thỏa mãn $\|\mathbf{v}_{LLL}\| \ll \lambda_1(L)$.
- Thành phần cuối cùng ϵ sẽ rất nhỏ, tương ứng với $N \cdot (c_0^* + c_1^*\beta + c_2^*\beta^2) \approx 0$.
- Ta kỳ vọng $\mathbf{v}_{LLL}$ sẽ gần với $\mathbf{c}_{\text{target}} = (-X, 0, 1, \epsilon)$.

Từ $\mathbf{v}_{LLL}$, ta suy ra:

1. $c_1^* \approx 0$.

2. c_2^* **là hệ số của** x^2 (thường là 1).

3. $c_0^* \approx -X$.

Giá trị X cần tìm chính là $\mathbf{X} = -c_0^*$.

2 Multi-time pad exploitation

2.1 Tổng quan

Theo nguyên tắc của One-Time Pad (OTP), mỗi khóa chỉ được phép sử dụng duy nhất một lần. Việc dùng lại một khóa cho nhiều plaintext (Multi-Time Pad) sẽ tạo ra lỗ hổng bảo mật nghiêm trọng. Cụ thể, khi thực hiện phép XOR giữa hai bản mã, khóa sẽ bị triệt tiêu, chỉ còn lại kết quả XOR của hai bản rõ:

$$\begin{aligned} C_1 \oplus C_2 &= (P_1 \oplus K) \oplus (P_2 \oplus K) \\ &= P_1 \oplus K \oplus P_2 \oplus K \\ &= P_1 \oplus P_2 \oplus K \oplus K \\ &= P_1 \oplus P_2 \oplus (K \oplus K) \\ &= P_1 \oplus P_2 \end{aligned}$$

việc khai thác tính chất trên, kết hợp với các phán đoán thống kê (tập trung vào việc đoán vị trí dấu cách (space)) để khôi phục lại khóa chung và giải mã toàn bộ tin nhắn.

2.2 Nguyên lý hoạt động

Script áp dụng kỹ thuật tương tự như "crib dragging" (kéo mẫu), với trọng tâm là tìm kiếm các dấu cách (space):

Algorithm 1: Phá mã Multi-Time Pad (Heuristic Dấu cách)

Input: Danh sách bản mã $C = \{C_1, \ldots, C_n\}$

Output: Khóa K và danh sách bản rõ P

$L \leftarrow \text{MaxLength}(C)$;

Khởi tạo mảng $K[0..L]$ với giá trị Null;

// Duyệt qua từng vị trí byte

for $i \leftarrow 0$ **to** $L - 1$ **do**

  foreach C_j *trong* C **do**

    if $i \geq Length(C_j)$ **then**

      continue

    // Giả định C_j tại i là dấu cách (0x20)

    $k_{candidate} \leftarrow C_j[i] \oplus 0x20$;

    $valid\_count \leftarrow 0$;

    // Kiểm tra khóa với các bản mã khác

    foreach C_m *trong* C **do**

      if $i < Length(C_m)$ **then**

        $p_{char} \leftarrow C_m[i] \oplus k_{candidate}$;

        if $IsEnglishChar(p_{char})$ **then**

          $valid\_count$++;

    // Nếu khóa hoạt động tốt với hầu hết bản mã

    if $valid\_count \approx Length(C)$ **then**

      $K[i] \leftarrow k_{candidate}$;

      break;

// Bước cuối: Giải mã toàn bộ tin nhắn

foreach C_j *trong* C **do**

  $P_j \leftarrow C_j \oplus K$;

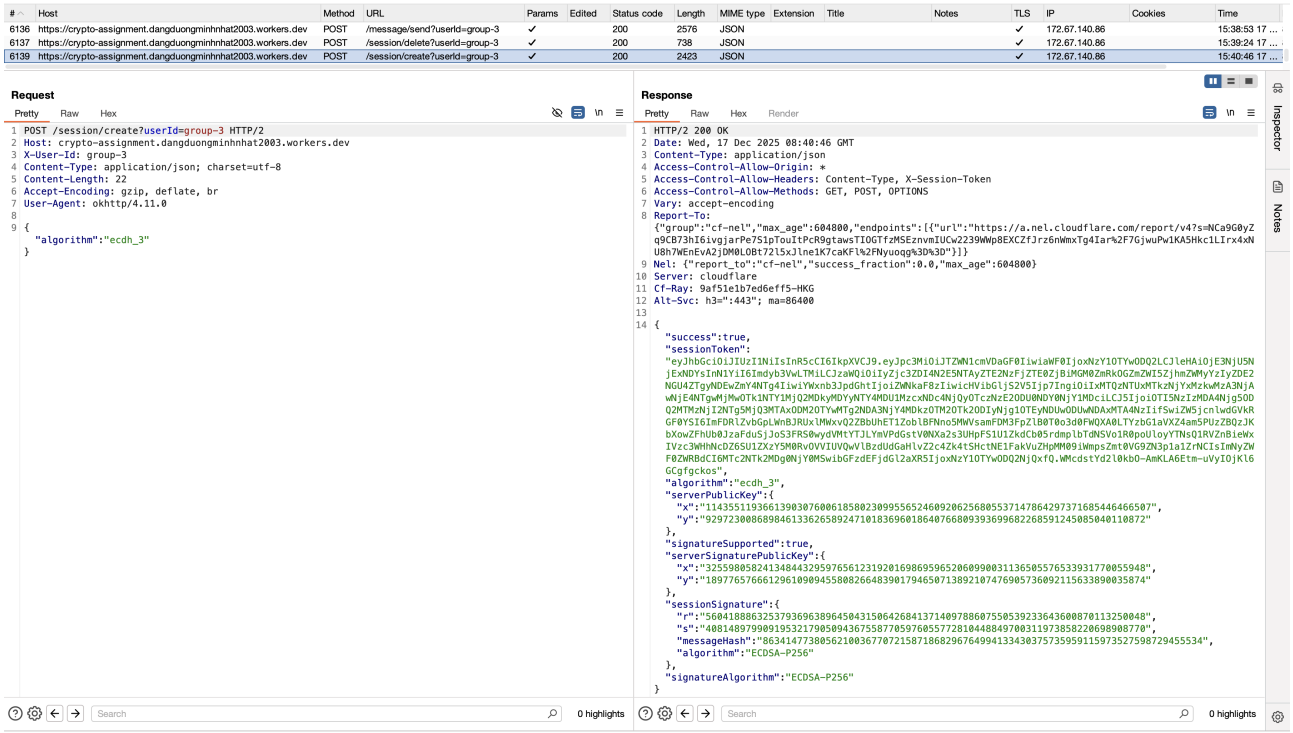
return K, P;

3 SMC Exploitation

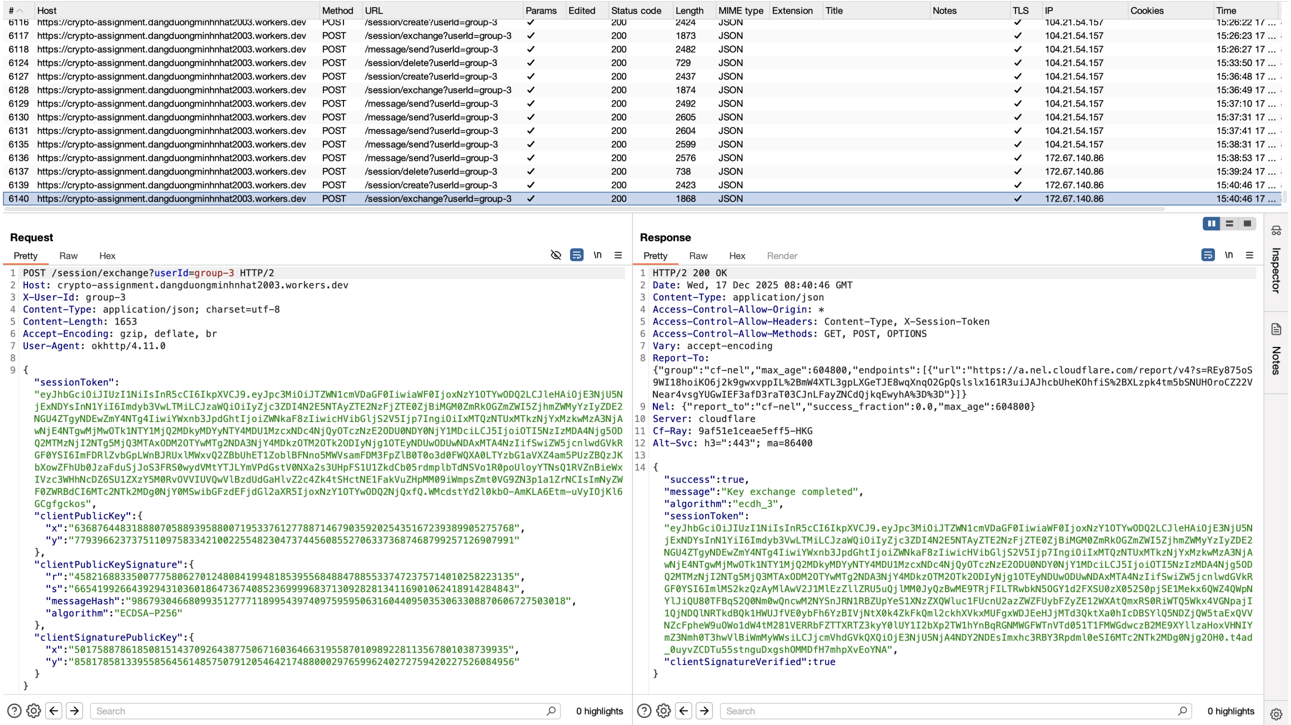
3.1 Interception & API Analysis

3.1.1 Tổng quan về các Endpoint API

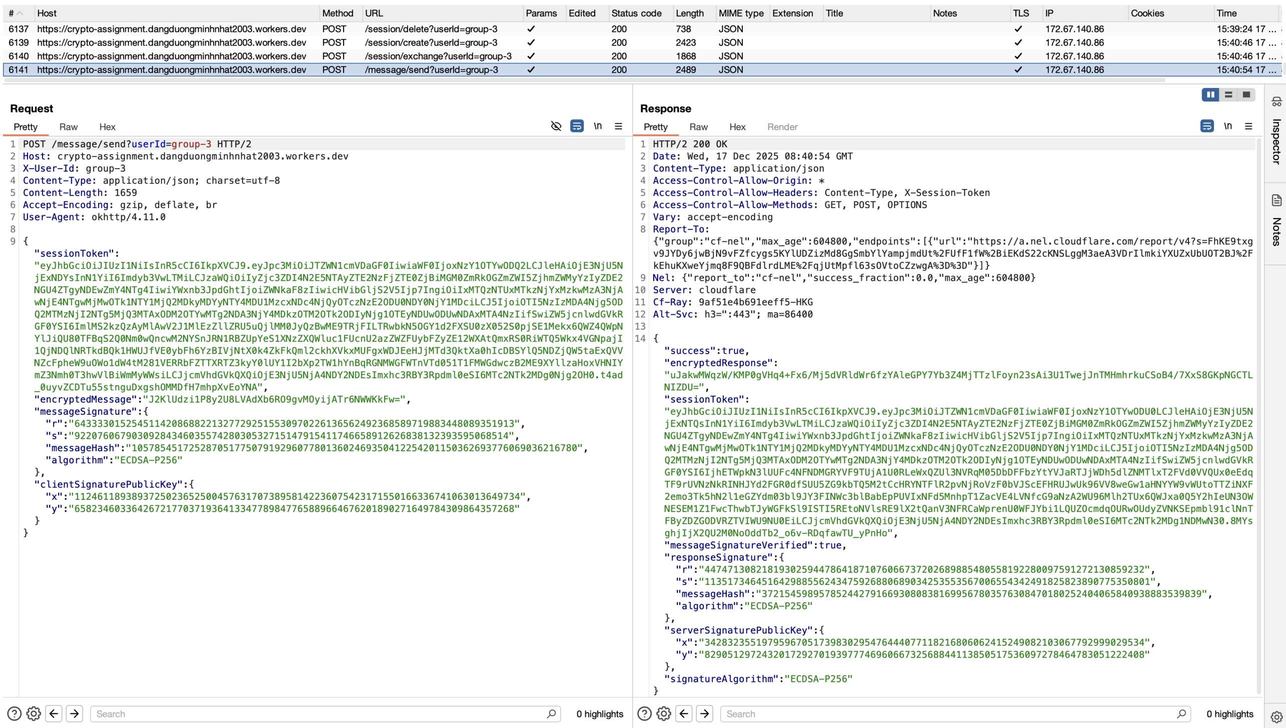
1. Hệ thống cung cấp 4 endpoint chính để quản lý phiên làm việc và trao đổi tin nhắn bảo mật:



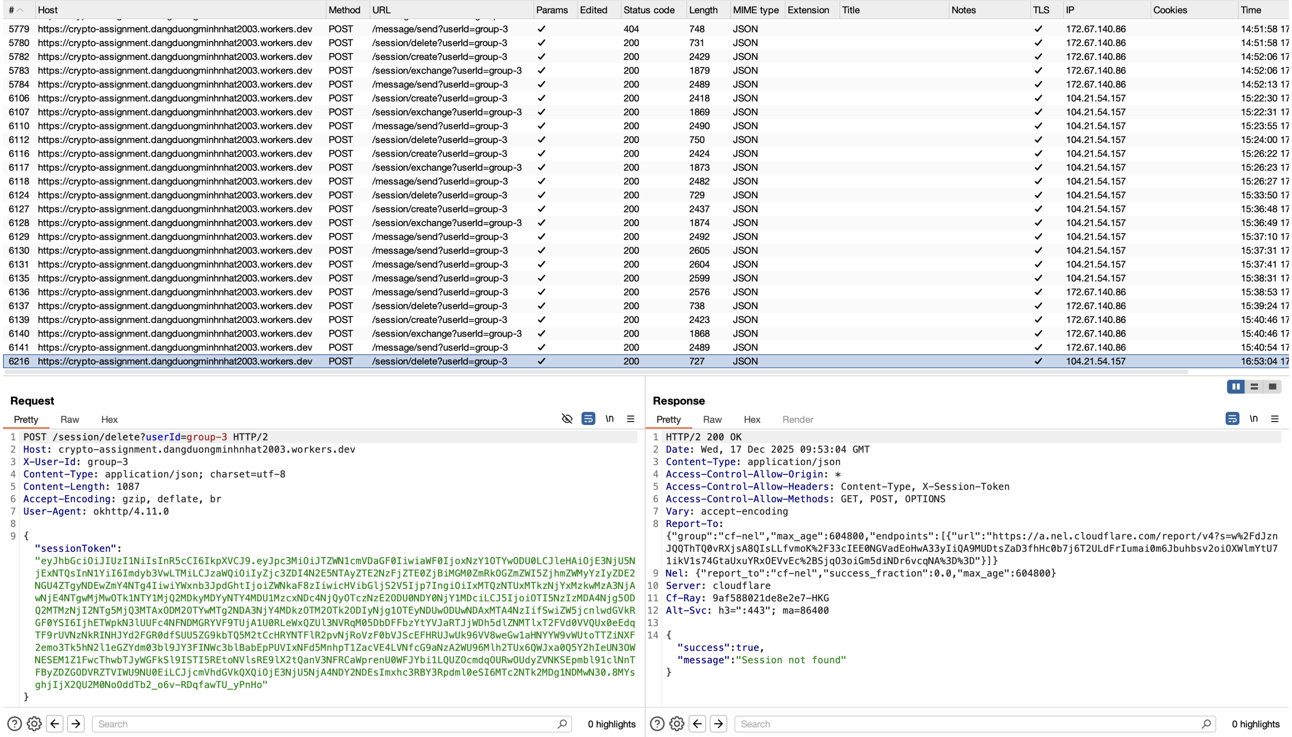
Hình 1: `/session/create`: Thiết lập phiên làm việc mới với máy chủ (Server).



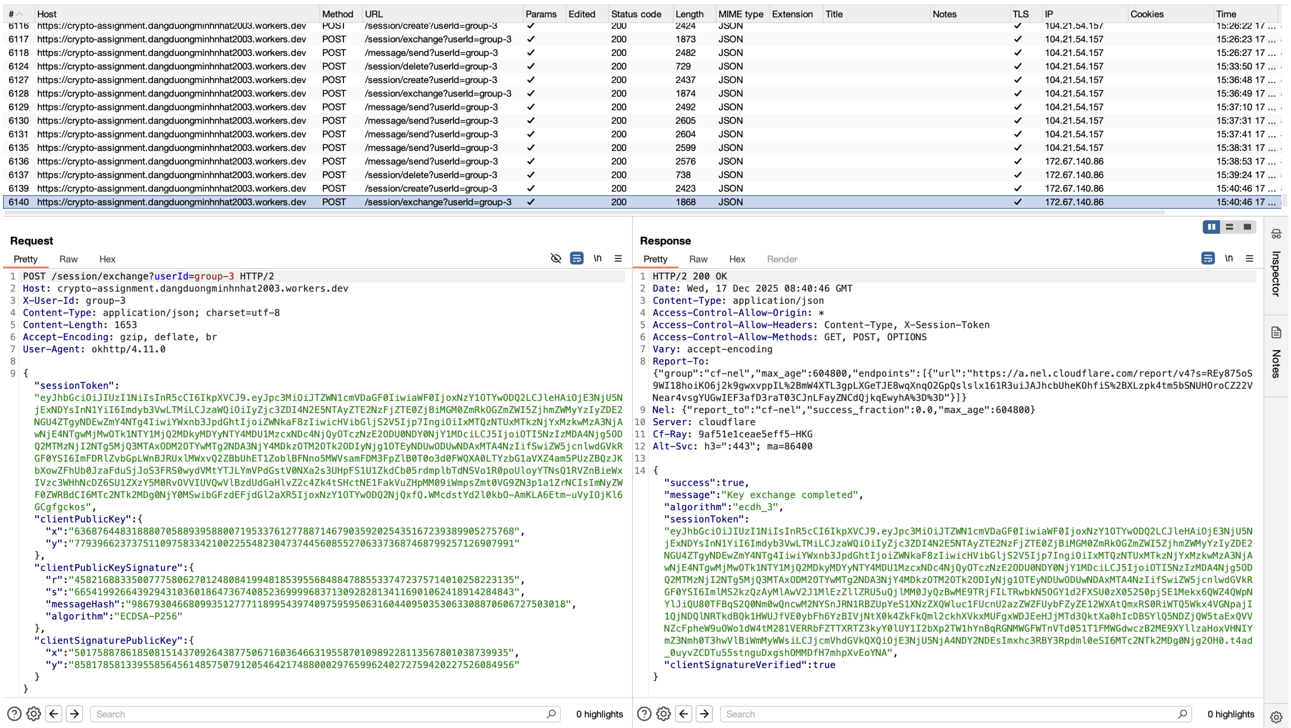
Hình 2: /session/exchange: Thực hiện giao thức trao đổi khóa Diffie-Hellman (ECDH) nhằm thiết lập khóa bí mật chung (Shared Secret).



Hình 3: /message/send: Truyền tải tin nhắn đã mã hóa tới Server và tiếp nhận phản hồi.



Hình 4: /session/delete: Kết thúc phiên làm việc hiện tại và hủy hiệu lực của sessionToken.



Hình 5: /session/status: Truy vấn trạn thái hiện tại của sessionToken.

3.1.2 Phân tích Chi tiết Request và Response

3.1.2.1 Endpoint: `/session/create`

- **Mục đích**: Khởi tạo phiên giao tiếp, tiếp nhận `sessionToken` và khóa công khai (Public Key) từ phía Server.
- **Request**:
  - **Method**: `POST`
  - **URL Parameter**: `userId={tên_nhóm}` (Ví dụ: `userId=group-3`)
  - **Body (JSON)**:

```
{
    "algorithm": "ecdh_3"
}
```

Mô tả tham số:

* `algorithm`: Định danh thuật toán trao đổi khóa được sử dụng. Trong trường hợp này là `ecdh_3` (Elliptic Curve Diffie-Hellman phiên bản 3).

- **Response (Thành công)**:
  - **Body (JSON)**: Trả về `sessionToken`, khóa công khai ECDH và khóa công khai ECDSA của Server phục vụ việc xác thực chữ ký.

```
{
    "success": true,
    "sessionToken": "...",
    "serverPublicKey": {
        "x": "...",
        "y": "..."
    },
    "signatureSupported": true,
    "serverSignaturePublicKey": {
```

```
        "x": "...",
        "y": "..."
    },
    "sessionSignature": {
        "r": "...",
        "s": "...",
        "messageHash": "...",
        "algorithm": "ECDSA-P256"
    },
    "signatureAlgorithm": "ECDSA-P256"
}
```

Mô tả tham số:

- `success`: Biến boolean xác nhận trạng thái thành công của yêu cầu.
- `sessionToken`: Chuỗi JWT (JSON Web Token) duy nhất đại diện cho phiên hiện hành, dùng để xác thực các yêu cầu tiếp theo.
- `serverPublicKey`: Khóa công khai ECDH của Server (gồm tọa độ x và y).
- `signatureSupported`: Xác nhận Server có hỗ trợ tính năng chữ ký số hay không.
- `serverSignaturePublicKey`: Khóa công khai ECDSA của Server, dùng để Client xác minh chữ ký từ Server.
- `sessionSignature`: Chữ ký số của Server cho phiên vừa tạo, bao gồm các thành phần r, s, giá trị băm của thông điệp (`messageHash`) và thuật toán (`algorithm`).
- `signatureAlgorithm`: Thuật toán chữ ký tổng thể được áp dụng cho phiên.

3.1.2.2 Endpoint: `/session/exchange`

- **Mục đích**: Cho phép Client chia sẻ khóa công khai với Server, từ đó cả hai phía có thể độc lập tính toán khóa bí mật chung (Shared Secret).
- **Request**:
  - **Method**: `POST`
  - **URL Parameter**: `userId={tên_nhóm}`
  - **Body (JSON)**: Bao gồm `sessionToken`, khóa công khai ECDH của Client kèm chữ ký số và khóa xác thực tương ứng.

```
{
    "sessionToken": "...",
    "clientPublicKey": { "x": "...", "y": "..." },
    "clientPublicKeySignature": {
        "r": "...",
        "s": "...",
        "messageHash": "...",
        "algorithm": "ECDSA-P256"
    },
    "clientSignaturePublicKey": { "x": "...", "y":
       "..." }
}
```

Mô tả tham số:

  - `clientPublicKey`: Khóa công khai ECDH của Client (gồm tọa độ x, y).
  - `clientPublicKeySignature`: Chữ ký số do Client tạo ra trên khóa công khai ECDH của chính nó, đảm bảo tính toàn vẹn và chống chối bỏ.
  - `clientSignaturePublicKey`: Khóa công khai ECDSA của Client, được Server dùng để xác minh `clientPublicKeySignature`.

- **Response (Thành công)**:
  - **Body (JSON)**: Xác nhận quy trình trao đổi khóa hoàn tất và cấp phát token phiên mới.

```
{
    "success": true,
    "message": "Key exchange completed",
    "algorithm": "ecdh_3",
    "sessionToken": "...",
    "clientSignatureVerified": true
}
```

Mô tả tham số:

* `message`: Thông báo trạng thái hoàn tất trao đổi khóa.
* `sessionToken`: Token phiên mới (được làm mới sau mỗi request để tăng cường bảo mật).
* `clientSignatureVerified`: Kết quả xác minh chữ ký của Client từ phía Server.

3.1.2.3 Endpoint: `/message/send`

- **Mục đích**: Truyền tải dữ liệu an toàn thông qua mã hóa.
- **Request**:
  - **Method**: `POST`
  - **URL Parameter**: `userId={tên_nhóm}`
  - **Header**: `X-User-Id: {tên_nhóm}`
  - **Body (JSON)**: Chứa tin nhắn đã mã hóa (Base64) và chữ ký xác thực.

```
{
    "sessionToken": "...",
    "encryptedMessage": "...", // Base64 encoded
    "messageSignature": {
```

```
        "r": "...",
        "s": "...",
        "messageHash": "...",
        "algorithm": "ECDSA-P256"
    },
    "clientSignaturePublicKey": {
        "x": "...",
        "y": "..."
    }
}
```

Mô tả tham số:

- `encryptedMessage`: Bản tin gốc được mã hóa bằng AES-256-CBC, sau đó encode sang định dạng Base64.
- `messageSignature`: Chữ ký số của Client.
- `messageHash`: Giá trị Hash của thông điệp mã hóa.

- **Response (Thành công)**:
  - **Body (JSON)**: Phản hồi từ Server dưới dạng mã hóa, kèm chữ ký số và token mới.

```
{
    "success": true,
    "encryptedResponse": "...", // Base64 encoded
    "sessionToken": "...",
    "responseSignature": {
        "r": "...",
        "s": "...",
        "messageHash": "...",
        "algorithm": "ECDSA-P256"
    },
    "serverSignaturePublicKey": {
```

```
        "x": "...",
        "y": "..."
    },
    "signatureAlgorithm": "ECDSA-P256"
}
```

Mô tả tham số:

* `encryptedResponse`: Dữ liệu phản hồi từ Server (đã mã hóa và encode Base64).
* `responseSignature`: Chữ ký của Server trên `encryptedResponse` để Client xác thực nguồn gốc.

3.1.2.4 Endpoint: `/session/delete`

- **Mục đích**: Hủy bỏ phiên làm việc.
- **Request**:
  - **Method**: `POST`
  - **URL Parameter**: `userId={tên_nhóm}`
  - **Body (JSON)**:

```
{
    "sessionToken": "..."
}
```

- **Response (Thành công)**:
  - **Body (JSON)**:

```
{
    "success": true,
    "message": "Session deleted successfully"
}
```

Mô tả tham số:

* `message`: Thông báo xác nhận phiên đã được xóa thành công khỏi hệ thống.

3.1.2.5 Endpoint: `/session/status`

- **Mục đích**: Cung cấp khả năng truy vấn và xác minh trạng thái hoạt động hiện tại của một phiên làm việc cụ thể.
- **Request**:
  - **Method**: `GET`
  - **URL Parameters**:
    * `token={sessionToken}`: Chuỗi JWT định danh cho phiên cần kiểm tra.
    * `userId={tên_nhóm}`: Định danh của nhóm người dùng (Ví dụ: `group-3`).
  - **Header**: `X-User-Id: {tên_nhóm}`
- **Response (Thành công)**:
  - **Body (JSON)**: Trả về thông tin chi tiết về sự tồn tại, thuật toán và thời hạn của phiên.

```
{
    "success": true,
    "exists": true,
    "algorithm": "ecdh_3",
    "expiresAt": 1765888210,
    "message": "Session is active"
}
```

Mô tả tham số:

* `success`: Trạng thái xử lý của yêu cầu (true/false).

* `exists`: Biến boolean xác nhận phiên làm việc có đang tồn tại hợp lệ trên hệ thống hay không.
* `algorithm`: Thuật toán trao đổi khóa đang được áp dụng cho phiên này (ví dụ: `ecdh_3`).
* `expiresAt`: Mốc thời gian (định dạng Unix Timestamp) mà tại đó phiên sẽ hết hiệu lực và bị hủy bỏ.
* `message`: Thông báo mô tả trạng thái hiện tại của phiên ("Session is active"/"Session not found or expired").

3.2 Re-implementation & Protocol Reconstruction

3.2.1 Luồng Giao thức Bảo mật

Quy trình giao tiếp an toàn của Client được thực hiện qua các bước tuần tự sau:

1. **Khởi tạo (Initialization)**:
    - Sinh cặp khóa ECDH (Private/Public Key) phục vụ cho giao thức trao đổi khóa.
    - Sinh cặp khóa ECDSA (Private/Public Key) dùng để ký số và xác thực tính toàn vẹn.
2. **Thiết lập Phiên (Session Establishment)**:
    - Gửi yêu cầu khởi tạo đến endpoint `/session/create`.
    - Tiếp nhận và lưu trữ `sessionToken` cùng `serverPublicKey`.
    - Thực hiện tính toán **Shared Secret** bằng cách kết hợp khóa riêng ECDH của Client và khóa công khai ECDH của Server (hàm `derive_shared_secret`).
    - Dẫn xuất khóa mã hóa đối xứng AES-256 từ Shared Secret thông qua hàm `PBKDF2-HMAC-SHA256`.
3. **Trao đổi Khóa (Key Exchange)**:
    - Client gửi khóa công khai ECDH của mình lên `/session/exchange`.
    - Kèm theo gói tin là chữ ký số (tạo bởi khóa riêng ECDSA) nhằm chứng minh quyền sở hữu khóa.
    - Sau khi Server xác thực thành công, kênh truyền an toàn được thiết lập và `sessionToken` được cập nhật.
4. **Truyền tải Tin nhắn (Message Transmission)**:Quá trình xử lý cho mỗi tin nhắn bao gồm:

a. **Mã hóa:** Tin nhắn được mã hóa bằng thuật toán AES-256-CBC sử dụng khóa đã dẫn xuất. Một vector khởi tạo (IV) ngẫu nhiên được sinh ra cho mỗi lần mã hóa.

b. **Ký số:** Bản mã (Ciphertext) sau khi được mã hóa Base64 sẽ được ký bằng khóa riêng ECDSA.

c. **Gửi:** Gửi bản mã và chữ ký tới `/message/send`.

d. **Nhận phản hồi:** Client nhận dữ liệu, xác thực chữ ký của Server và giải mã nội dung bằng khóa AES.

5. **Hủy Phiên (Session Termination)**:

- Gửi yêu cầu tới `/session/delete` kèm theo `sessionToken` hiện hành để kết thúc phiên làm việc.

3.2.2 Mô tả Mã giả (Pseudocode) các Giải thuật Mật mã

Dưới đây là mô tả logic của các hàm mật mã cốt lõi:

3.2.2.1 Cơ chế Trao đổi khóa (ECDH & Key Derivation)

```
function derive_shared_secret(client_ecdh_private_key, server_ecdh_
    public_key):
    // Tinh toan Shared Secret theo giao thuc Diffie-Hellman
    shared_secret = client_ecdh_private_key.exchange(server_ecdh_
        public_key)

    // Dan xuat khoa AES tu Shared Secret
    salt = "0x0000000000000000" // 16 bytes null
    aes_key = PBKDF2(
        password=shared_secret,
        salt=salt,
        key_length=32,
        iterations=1000,
        hash_function=SHA256
```

```
    )
    return aes_key
```

Đoạn 1: Trao đổi khóa và dẫn xuất khóa AES

3.2.2.2 Cơ chế Mã hóa (AES-256-CBC)

```
function encrypt(plaintext, aes_key):
    iv = generate_random_bytes(16) // Sinh IV ngau nhien

    // Them padding PKCS7 de dat do dai block 128-bit
    padder = PKCS7(128).padder()
    padded_plaintext = padder.update(plaintext) + padder.finalize()

    // Ma hoa AES che do CBC
    cipher = AES_CBC(key=aes_key, iv=iv)
    ciphertext = cipher.encrypt(padded_plaintext)

    // Ket qua gom IV + Ciphertext, sau do ma hoa Base64
    return base64_encode(iv + ciphertext)
```

Đoạn 2: Quy trình mã hóa tin nhắn

3.2.2.3 Cơ chế Ký và Xác thực (ECDSA)

```
function sign(message, ecdsa_private_key):
    // Ky thong diep su dung SHA256 hashing
    signature = ecdsa_private_key.sign(message, hash_function=SHA256)
    return (r, s) from signature

function verify(message, signature, ecdsa_public_key):
    // Xac thuc chu ky
    return ecdsa_public_key.verify(signature, message, hash_function=
        SHA256)
```

Đoạn 3: Ký số và Xác thực

3.2.3 Quản lý Trạng thái Client

Client duy trì các biến trạng thái quan trọng trong suốt vòng đời ứng dụng:

- `ecdh_private_key`, `ecdsa_private_key`: Được khởi tạo một lần duy nhất và bất biến.
- `shared_secret`: Giá trị ban đầu là `null`, được tính toán ngay sau khi thiết lập phiên thành công.
- `aes_key`: Giá trị ban đầu là `null`, là kết quả dẫn xuất từ `shared_secret`.
- `session_token`: Giá trị ban đầu là `null`. Token này được cấp phát khi tạo phiên và được làm mới (xoay vòng) sau mỗi lần gọi `exchange_keys` hoặc `send_message` thành công để chống tấn công replay.

3.2.4 Ánh xạ API Endpoint và Hàm xử lý

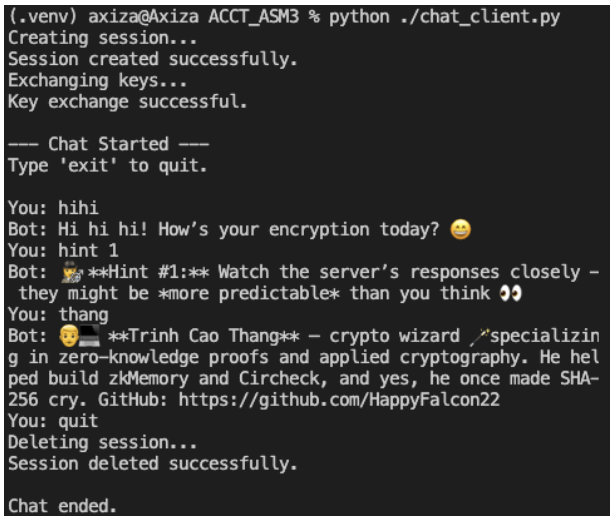
Bảng dưới đây mô tả mối quan hệ giữa các endpoint API và các phương thức được triển khai trong lớp `ChatAPI` của file Runnable `chat_client.py`:

Endpoint API	Hàm trong `chat_client.py`	Mô tả chức năng
`/session/create`	`ChatAPI.create_session()`	Khởi tạo phiên, nhận khóa server và tính toán khóa bí mật chung.
`/session/exchange`	`ChatAPI.exchange_keys()`	Gửi khóa công khai ECDH của Client lên Server để đồng bộ hóa khóa bí mật.
`/message/send`	`ChatAPI.send_message()`	Thực hiện mã hóa, ký số tin nhắn gửi đi và giải mã phản hồi nhận về.
`/session/delete`	`ChatAPI.delete_session()`	Gửi yêu cầu chấm dứt phiên làm việc và hủy token.

Bảng 2: Bảng ánh xạ chức năng và API Endpoint

3.2.5 Chạy thử

`chat_client.py` sẽ dựa trên flow trên để tương tác với server được cung cấp và có kết quả như hình dưới.



Hình 6: Chạy thử `chat_client.py`

3.3 Exploitation & Proof-of-Concept

3.3.1 Phân tích Lỗ hổng Kênh kề: Rò rỉ Độ dài (Side-Channel Leakage)

3.3.1.1 Cơ chế

Mã hóa AES ở chế độ CBC kết hợp với đệm PKCS7 (PKCS7 Padding) nhưng thiếu các biện pháp làm nhiễu độ dài (Traffic Padding). Do đặc thù của block cipher, độ dài của bản mã (L_{cipher}) có mối tương quan tuyến tính trực tiếp với độ dài dữ liệu gốc (L_{plain}) theo công thức:

$$L_{cipher} = \left\lceil \frac{L_{plain} + 1}{16} \right\rceil \times 16$$

Điều này có nghĩa là mọi sự thay đổi về độ dài của thông điệp đầu vào (dù nhỏ) đều được phản ánh trực tiếp lên kích thước của gói tin HTTP gửi đi.

3.3.1.2 Kịch bản Khai thác Thực nghiệm (PoC)

Kẻ tấn công có thể thụ động lắng nghe trên đường truyền và chặn bắt các gói tin gửi tới endpoint `/message/send`. Quy trình suy luận thông tin diễn ra như sau:

1. Trích xuất chuỗi Base64 từ trường `encryptedMessage`.
2. Giải mã Base64 để thu được chuỗi byte thô và loại bỏ 16 byte đầu (Vector khởi tạo - IV).
3. Tính toán kích thước payload mã hóa (L_{enc}) và suy ngược ra khoảng độ dài của tin nhắn gốc.

Đoạn mã Python dưới đây minh họa khả năng tự động hóa việc suy luận độ dài tin nhắn từ dữ liệu bị rò rỉ:

```
import base64

def analyze_traffic_leak(b64_ciphertext):
    # 1. Giai ma Base64 lay chuoi byte tho
```

```
    raw_bytes = base64.b64decode(b64_ciphertext)

    # 2. Tru di 16 bytes cua IV (Initialization Vector)
    enc_payload_len = len(raw_bytes) - 16

    # 3. Suy luan khoang do dai dua tren padding PKCS7 (1-16 bytes)
    max_len = enc_payload_len - 1
    min_len = enc_payload_len - 16

    print(f"[+] Encrypted Block Size: {enc_payload_len} bytes")
    print(f"[!] Inferred Plaintext: {min_len} to {max_len} bytes")
```

Đoạn 4: Script PoC: Suy luận độ dài tin nhắn từ bản mã

3.3.1.3 Đánh giá Tác động

Mặc dù không trực tiếp lộ nội dung văn bản, lỗ hổng này cho phép kẻ tấn công phân loại các mẫu phản hồi. Ví dụ: dễ dàng phân biệt giữa câu trả lời ngắn ("Yes") và câu trả lời dài ("No, because..."), hoặc xác định loại lệnh được gửi đi dựa trên kích thước đặc trưng.

Giải pháp Khắc phục: Để triệt tiêu vector tấn công này, hệ thống cần áp dụng cơ chế **Constant-size Padding** (Đệm kích thước cố định).

- Mọi thông điệp trước khi mã hóa phải được đệm thêm dữ liệu ngẫu nhiên để đạt đến một độ dài chuẩn thống nhất (ví dụ: cố định 1024 bytes cho mọi gói tin).
- Điều này đảm bảo rằng tất cả các bản mã gửi qua mạng đều có kích thước y hệt nhau, khiến việc phân tích lưu lượng trở nên vô hiệu.

3.3.2 Phân tích Lỗ hổng: Tấn công Padding Oracle (Padding Oracle Attack)

3.3.2.1 Cơ chế Kỹ thuật và Nguyên nhân

Lỗ hổng xuất phát từ việc hệ thống sử dụng chế độ mã hóa khối CBC (Cipher Block Chaining) kết hợp với chuẩn đệm PKCS#7, đồng thời trả về các thông báo lỗi khác biệt cho các trạng thái giải mã khác nhau.

1. **Cơ chế AES CBC và PKCS#7:** Trong quá trình giải mã, Server thực hiện giải mã khối bản mã C_i bằng khóa AES, sau đó XOR kết quả với khối bản mã trước đó C_{i-1} để thu được bản rõ P_i.

$$P_i = D_K(C_i) \oplus C_{i-1}$$

   Sau khi giải mã xong, Server sẽ kiểm tra tính hợp lệ của lớp đệm (padding) ở cuối khối cuối cùng theo chuẩn PKCS#7 (ví dụ: byte cuối là `0x01`, hoặc 2 byte cuối là `0x02 0x02`,...).

2. **The Oracle:** Server đóng vai trò là một "Oracle" khi nó phản hồi các thông báo lỗi khác nhau:

   - **Padding Error:** Trả về khi lớp đệm sau giải mã không đúng định dạng.
   - **Application Error:** Trả về khi lớp đệm đúng, nhưng dữ liệu không khớp với chữ ký ECDSA.

   Sự khác biệt này cho phép kẻ tấn công xác định xem một bản mã giả mạo có tạo ra lớp đệm hợp lệ hay không.

3.3.2.2 Nguyên lý Khôi phục Dữ liệu (Byte-by-Byte Decryption)

Kẻ tấn công sở hữu bản mã C gồm các khối $C_1, C_2, \ldots$. Để giải mã khối C_2, kẻ tấn công cần tìm P_2. Quy trình thực hiện như sau:

Khái niệm Trạng thái Trung gian (Intermediate State - I_2): Giá trị sau khi

giải mã AES nhưng chưa XOR với khối trước đó được gọi là I_2.

$$I_2 = D_K(C_2)$$

$$P_2 = I_2 \oplus C_1$$

Nếu tìm được I_2, ta sẽ tìm được P_2 vì C_1 đã biết.

Quy trình tấn công (Ví dụ với byte cuối cùng):

1. Kẻ tấn công tạo một khối mã giả C_1' và gửi chuỗi $C_1'|C_2$ lên Server.
2. Server tính toán: $P_2' = I_2 \oplus C_1'$.
3. Kẻ tấn công thay đổi byte cuối cùng của C_1' (từ 0 đến 255) cho đến khi Oracle không báo lỗi Padding. Lúc này, byte cuối của P_2' chắc chắn là `0x01` (padding hợp lệ độ dài 1).
4. Tại thời điểm đó:

$$I_2[15] \oplus C_1'[15] = \texttt{0x01} \Rightarrow I_2[15] = C_1'[15] \oplus \texttt{0x01}$$

5. Sau khi tìm được $I_2[15]$, kẻ tấn công khôi phục byte bản rõ gốc:

$$P_2[15] = I_2[15] \oplus C_1[15]$$

Quy trình này được lặp lại ngược từ byte cuối lên byte đầu của khối, sau đó lặp lại cho toàn bộ các khối bản mã khác.

3.3.2.3 Proof-of-Concept

`padding_oracle.py` dùng để tự động hóa hoàn toàn quy trình tấn công này. Công cụ thực hiện các bước sau:

1. **Khởi tạo:** Thiết lập phiên làm việc và lấy một mẫu `sessionToken` hoặc tin nhắn mã hóa hợp lệ làm mục tiêu.

2. **Phân mảnh:** Chia bản mã mục tiêu thành các khối 16 bytes.

3. **Vét cạn (Brute-force):** Với mỗi byte cần giải mã, công cụ gửi tối đa 256 yêu cầu HTTP tới endpoint `/message/send` với phần payload đã bị chỉnh sửa.

4. **Phân tích phản hồi:** Dựa vào HTTP Status Code hoặc nội dung lỗi trả về để xác định byte chính xác.

Kết quả chạy PoC minh họa:

```
(.venv) axiza@Axiza ACCT_ASM3 % python ./padding_oracle.py

Attacking block 1...
Found byte 15: 0x0e ('<non-printable>')
  [+] Detected padding of length 14. Propagating...
Found byte 1: 0x69 ('i')
Found byte 0: 0x68 ('h')

Decrypted Result (Hex): 68690e0e0e0e0e0e0e0e0e0e0e0e0e0e
Decrypted Message: hi
```

Hình 7: Giải mã tin nhắn gửi đến server

3.3.2.4 Kết luận

Kết quả thực nghiệm chứng minh rằng kẻ tấn công có thể giải mã toàn bộ dữ liệu nhạy cảm được truyền tải trong hệ thống mà không cần biết khóa AES. Đây là lỗi thiết kế nghiêm trọng cần được khắc phục ngay lập tức bằng cách chuẩn hóa thông báo lỗi hoặc chuyển sang sử dụng các thuật toán mã hóa có xác thực (Authenticated Encryption) như AES-GCM.

```
Attacking block 4...
Found byte 1/16: 10 ('<non-printable>')
  [+] Detected padding of length 16. Propagating...

Attacking block 3...
Found byte 1/16: 03 ('<non-printable>')
Found byte 2/16: 03 ('<non-printable>')
Found byte 3/16: 03 ('<non-printable>')
Found byte 4/16: 84 ('<non-printable>')
Found byte 5/16: 98 ('<non-printable>')
Found byte 6/16: 9f ('<non-printable>')
Found byte 7/16: f0 ('<non-printable>')
Found byte 8/16: 20 (' ')
Found byte 9/16: 3f ('?')
Found byte 10/16: 79 ('y')
Found byte 11/16: 61 ('a')
Found byte 12/16: 64 ('d')
Found byte 13/16: 6f ('o')
Found byte 14/16: 74 ('t')
Found byte 15/16: 20 (' ')
Found byte 16/16: 6e ('n')

Attacking block 2...
Found byte 1/16: 6f ('o')
Found byte 2/16: 69 ('i')
Found byte 3/16: 74 ('t')
Found byte 4/16: 70 ('p')
Found byte 5/16: 79 ('y')
Found byte 6/16: 72 ('r')
Found byte 7/16: 63 ('c')
Found byte 8/16: 6e ('n')
Found byte 9/16: 65 ('e')
Found byte 10/16: 20 (' ')
Found byte 11/16: 72 ('r')
Found byte 12/16: 75 ('u')
Found byte 13/16: 6f ('o')
Found byte 14/16: 79 ('y')
Found byte 15/16: 20 (' ')
Found byte 16/16: 73 ('s')

Attacking block 1...
Found byte 1/16: 99 ('<non-printable>')
Found byte 2/16: 80 ('<non-printable>')
Found byte 3/16: e2 ('<non-printable>')
Found byte 4/16: 77 ('w')
Found byte 5/16: 6f ('o')
Found byte 6/16: 48 ('H')
Found byte 7/16: 20 (' ')
Found byte 8/16: 21 ('!')
Found byte 9/16: 69 ('i')
Found byte 10/16: 68 ('h')
Found byte 11/16: 20 (' ')
Found byte 12/16: 69 ('i')
Found byte 13/16: 68 ('h')
Found byte 14/16: 20 (' ')
Found byte 15/16: 69 ('i')
Found byte 16/16: 48 ('H')

Decrypted Result (Hex): 486920686920686921204 86f77e280997320796f757220656e637279
7074696f6e20746f6461793f20f09f9884030303101010101010101010101010101010
Decrypted Message: Hi hi hi! How’s your encryption today? 😀
```

Hình 8: Giải mã tin nhắn nhận được từ server