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

MÔN HỌC

MẬT MÃ HỌC VÀ MÃ HÓA THÔNG TIN

Assignment

GVHD: Nguyễn An Khương
SV thực hiện: Mai Bách Huyên - 2311269
Nguyễn Ngọc Thành Đạt - 2111013
Dương Bá Khang - 2311403

Tp. Hồ Chí Minh, tháng 11/2025

Mục lục

1 Team Contribution **3**

2 Lattices **3**
2.1 Các định nghĩa 3
2.2 Bài toán: Tìm đa thức tối tiểu từ giá trị xấp xỉ 5
2.2.1 Giả định bậc và tham số 5
2.2.2 Xây dựng Ma trận Cơ sở B 5
2.2.3 Thuật toán LLL và kết quả 6
2.2.4 Suy luận ra Đa thức Tối tiểu $f(x)$ 6
2.2.5 Kiểm chứng Đại số 6
2.3 Tìm số nguyên X từ giá trị xấp xỉ $\sqrt{X}$ 6
2.3.1 Thiết lập Mô hình Lattice và Tham số 6
2.3.2 Xây dựng Ma trận Cơ sở B' 7
2.3.3 Thuật toán LLL và Suy luận 7
2.3.4 Kiểm chứng Đại số 7

3 Attacking the Vigenere Cipher **8**
3.1 Giới thiệu bài toán 8
3.2 Cơ sở lý thuyết thám mã 8
3.2.1 Chỉ số trùng khớp (Index of Coincidence - IoC) 8
3.2.2 Phân tích tần suất (Chi-squared Test) 8
3.3 Quy trình thực hiện 8
3.3.1 Làm sạch dữ liệu (Data Cleaning) 8
3.3.2 Tự động tìm khóa 9
3.3.3 Giải mã (Decryption) 9
3.4 Toàn bộ code và chạy với file Ciphertext1.txt 9
3.5 Kết quả thực nghiệm 12
3.5.1 Khóa tìm được 13
3.5.2 Văn bản giải mã 13

4 SMC Exploitation **13**
4.1 Interception & API Analysis 13
4.1.1 API Table 13
4.1.2 Status 14
4.1.3 Create Session 16
4.1.4 Checking exchange session 20
4.1.5 Messaging 23
4.2 Re-implementation & Protocol Reconstruction 26
4.2.1 General protocol 26
4.2.2 Sequence diagram 26
4.3 Pseudocode 28
4.3.1 Create session 28
4.3.2 Exchange session 28
4.3.3 Encrypting and send message 29
4.4 Re-implementation 30
4.5 SMC Exploitation 31
4.5.1 Threat Model 31
4.5.2 Vulnerability Theory 31
4.5.3 Attack Methodology 32

4.5.4 PoC Implementation 32
4.6 Experimental Results 33
4.6.1 Kịch bản 1 33
4.6.2 Kịch bản 2 33
4.7 Conclusion & Improvement 34
4.7.1 Conclusion 34
4.7.2 Improvement 34

Project Deliverables:
Toàn bộ source code Re-implementation và Exploitation cho Problem 2 và 3 có thể được tìm thấy tại: Link Github
Live demo PoC phần 3: Link Demo

1 Team Contribution

Thành viên	Specific Tasks	Completed	% Hoàn thành
Mai Bách Huyên	Hiện thực hóa phần Vigenere Cipher	Hoàn thành	100%
Nguyễn Ngọc Thành Đạt	Hiện thực hóa phần Lattices	Hoàn thành	100%
Dương Bá Khang	Hiện thực hóa SMC Exploitation	Hoàn thành	100%

Bảng 1: *Bảng phân công công việc và mức độ đóng góp*

2 Lattices

2.1 Các định nghĩa

1. **Vector spaces (Không gian vector):** Một tập $V \subset \mathbb{R}^m$ được gọi là không gian vector nếu với mọi $v_1, v_2 \in V$ và mọi hệ số thực $\alpha_1, \alpha_2 \in \mathbb{R}$ ta có:

$$\alpha_1 v_1 + \alpha_2 v_2 \in V$$

Nói cách khác, không gian vector là tập con của $\mathbb{R}^m$ đóng dưới phép cộng và phép nhân vô hướng với các phần tử của $\mathbb{R}$.

2. **Linear combinations (Tổ hợp tuyến tính):** Với vector $v_1, ..., v_k \in V$, một tổ hợp tuyến tính là:

$$w = \alpha_1 v_1 + ... + \alpha_k v_k, \quad \alpha_i \in \mathbb{R}$$

Tập tất cả các tổ hợp tuyến tính gọi là tập sinh của v_i.

3. **Independence (Tập độc lập):** Tập các vector $v_1, ..., v_k$ độc lập (tuyến tính) nếu phương trình:

$$\alpha_1 v_1 + ... + \alpha_k v_k = 0$$

chỉ có nghiệm $\alpha_1 = ... = \alpha_k = 0$. Nếu có ít nhất một nghiệm $\alpha_i \neq 0$ thì tập đó phụ thuộc.

4. **Bases (Cơ sở):** Một cơ sở của V là một tập các vector độc lập tuyến tính mà sinh V. Điều này tương đương rằng với mỗi vector $w \in V$ có thể viết dưới dạng:

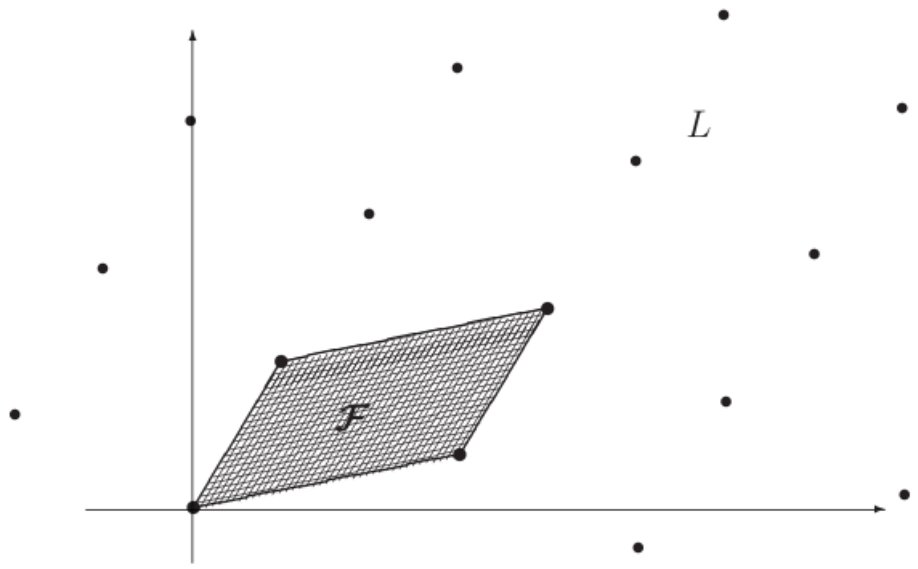
$$w = \alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n$$

cho mỗi $\alpha_1, ..., \alpha_n \in \mathbb{R}$ duy nhất. Số vector trong cơ sở gọi là dimension (số chiều).

5. **Orthogonal and orthonormal basis (Cơ sở trực giao và cơ sở trực chuẩn):**
   - Cơ sở trực giao: các vector thuộc cơ sở vuông góc từng cặp:
   $$v_i \cdot v_j = 0 \quad \forall i \neq j$$
   - Cơ sở trực chuẩn: là cơ sở trực giao mà thêm vào đó $\|v\| = 1$ với mọi i.

6. **Lattices (Lưới):** Cho một tập vector độc lập tuyến tính $v_1, ..., v_n \in \mathbb{R}^m$. Lattice L do các vector này sinh ra là tập các tổ hợp tuyến tính của $v_1, ..., v_n$ với hệ số nguyên:
   $$L = \{\alpha_1 v_1 + ... + \alpha_n v_n : \alpha_i \in \mathbb{Z}\}$$
   Một cơ sở của L là bất kỳ tập vector độc lập nào sinh ra L. Hai tập cơ sở bất kỳ đều có cùng số phần tử. Số chiều của L là số vector trong một cơ sở của L.

7. **Fundamental domain (Miền cơ bản):** Với cơ sở $v_1, ..., v_n$ của lattice L, miến cơ bản tương ứng là:
   $$\mathcal{F}(v_1, ..., v_n) = \{t_1 v_1 + ... + t_n v_n : 0 \leq t_i < 1\}$$



Hình 1: *Lattice L và miền cơ bản $\mathcal{F}$*

8. **Shortest Vector Problem (SVP):** Tìm một vector khác 0 ngắn nhất $v \in L$ có chuẩn Euclid nhỏ nhất, tức là tối thiểu $\|v\|$.

9. **Closet Vector Problem (CVP):** Cho trước vector $w \in \mathbb{R}^m$ không nằm trong L, tìm vector $v \in L$ sao cho $\|w - v\|$ nhỏ nhất.

10. **The Euclidean ball (Quả cầu Euclid):** Với bất kỳ $a \in \mathbb{R}^n$ và bán kính $R > 0$, quả cầu Euclid tâm a bán kính R là tập hợp:
    $$\mathbb{B}_R(a) = \{x \in \mathbb{R}^n : \|x - a\| \leq R\}$$
    Dùng để đếm số điểm lattice nằm trong bóng và liên hệ với thể tích của miền cơ bản để ước lượng độ dài vector ngắn nhất.

11. **The Gaussian expected shortest length (Độ dài ngắn nhất kỳ vọng Gauss):** Với lattice L có chiều n, độ dài ngắn nhất kỳ vọng Gauss được định nghĩa:

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}}(detL)^{1/n}$$

Heuristic Gaussian cho rằng một vector ngắn nhất khác 0 trong một lattice "được chọn ngẫu nhiên"thỏa mãn:

$$\|v_{shortest}\| \approx \sigma(L)$$

12. **The LLL algorithm (Thuật toán LLL):** Thuật toán LLL là một thuật toán đa thức được thiết kế để tìm một cơ sở gần như trực giao cho một lattice L cho trước.
Đặc điểm và mục đích:

- **Đầu vào:** Một cơ sở $\mathcal{B} = \{v_1, ..., v_n\}$
- **Đầu ra:** Một cơ sở $\mathcal{B}^* = \{v_1^*, ..., v_n^*\}$ được rút gọn LLL (LLL-reduced basis) cho cùng một lattice L.
- **Tính chất của Cơ sở Rút gọn LLL:** Cơ sở $\mathcal{B}^*$ thỏa mãn 2 điều kiện chính sau:
  - **Size Condition:**

$$|\mu_{i,j}| = \frac{|v_i \cdot v_j^*|}{\|v_j^*\|^2} \leq \frac{1}{2} \quad \forall 1 \leq j < i \leq n$$

  - **Lovász Condition:**

$$\|v_j^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right)\|v_{i-1}^*\|^2 \quad \forall 1 < i \leq n$$

- **Ứng dụng trong mật mã học:** LLL là một công cụ nền tảng trong lý thuyết lưới vì nó cung cấp một phương pháp hiệu quả để tìm các vector ngắn một cách tương đối trong lưới, giải quyết xấp xỉ Bài toán SVP và Bài toán CVP trong thời gian đa thức.

2.2 Bài toán: Tìm đa thức tối tiểu từ giá trị xấp xỉ

2.2.1 Giả định bậc và tham số

Ta giả định đa thức tối tiểu có bậc là $n = 4$ (dựa trên cấu trúc $\sqrt[4]{}$). Do đó, ta tìm vector hệ số $\mathbf{c} = (c_0, c_1, c_2, c_3, c_4)$ sao cho:

$$\sum_{i=0}^{4} c_i \beta^i \approx 0$$

- Giá trị xấp xỉ: $\beta \approx 22.8211602868$.
- Hằng số trọng số: $K = 10^{10}$ (tương ứng với độ chính xác 10 chữ số thập phân).

2.2.2 Xây dựng Ma trận Cơ sở B

Ma trận cơ sở B có kích thước 5×5, được xây dựng để tìm vector $\mathbf{v}$ ngắn nhất, trong đó c_i là các thành phần đầu và thành phần cuối cùng ϵ là sai số được khuếch đại:

$$\mathbf{v} = (c_0, c_1, c_2, c_3, c_4) \cdot B = \left(c_0, c_1, c_2, c_3, c_4, \ K \cdot \sum_{i=0}^{4} c_i \beta^i\right)$$

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & \lfloor K \cdot 1 \rfloor \\ 0 & 1 & 0 & 0 & \lfloor K \cdot \beta \rfloor \\ 0 & 0 & 1 & 0 & \lfloor K \cdot \beta^2 \rfloor \\ 0 & 0 & 0 & 1 & \lfloor K \cdot \beta^3 \rfloor \\ 0 & 0 & 0 & 0 & \lfloor K \cdot \beta^4 \rfloor \end{pmatrix}$$

2.2.3 Thuật toán LLL và kết quả

Ta áp dụng Thuật toán Thu gọn Lattice LLL trên ma trận B. Mục tiêu của LLL là tìm vector $\mathbf{v}_{LLL}$ có độ dài Euclide nhỏ nhất.

Lý thuyết: Do β xấp xỉ nghiệm của một đa thức có hệ số nguyên và bậc nhỏ, vector hệ số của đa thức đó ($\mathbf{c}$) sẽ tạo ra một vector $\mathbf{v}_{target}$ cực ngắn trong Lattice, nhờ thành phần cuối cùng $\epsilon \approx 0$.

Thực thi:Sau khi chạy thuật toán LLL, vector ngắn nhất tìm được (với các thành phần là số nguyên) là:

$$\mathbf{v}_{LLL} = (194470, -37044, 2646, -84, 1, \epsilon)$$

Trong đó ϵ là một số nguyên rất nhỏ (hoặc xấp xỉ 0 nếu tính toán không bị làm tròn).

2.2.4 Suy luận ra Đa thức Tối tiểu $f(x)$

Ta diễn giải các thành phần nguyên của $\mathbf{v}_{LLL}$ thành các hệ số đa thức $f(x) = c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$. Từ đó, ta suy luận ra đa thức tối tiểu $f(x)$:

$$f(x) = x^4 - 84x^3 + 2646x^2 - 37044x + 194470$$

2.2.5 Kiểm chứng Đại số

Sau khi tìm ra $f(x)$ từ Lattice, ta kiểm tra xem đa thức này có thực sự có α làm nghiệm hay không:

- Thay $x = \alpha$ vào đa thức $f(x)$:

$$f(\alpha) = \alpha^4 - 84\alpha^3 + 2646\alpha^2 - 37044\alpha + 194470$$

- Thay $\alpha = 21 + \sqrt[4]{11}$ vào đa thức trên.
- Ta có thể thấy rằng kết quả cho ra $f(\alpha) = 0$

Kết luận: Quá trình tìm kiếm dựa trên Lattice đã thành công xác định chính xác đa thức tối tiểu $f(x) = x^4 - 84x^3 + 2646x^2 - 37044x + 194470$.

2.3 Tìm số nguyên X từ giá trị xấp xỉ $\sqrt{X}$

Mục tiêu: Tìm số nguyên X bằng cách tìm vector hệ số (c_0, c_1, c_2) của đa thức tối tiểu $f(t) = c_2t^2 + c_1t + c_0$ mà $\sqrt{X}$ là nghiệm, sử dụng giá trị xấp xỉ γ.

2.3.1 Thiết lập Mô hình Lattice và Tham số

1. Giả định Bậc: Ta biết $\sqrt{X}$ là nghiệm của $t^2 - X = 0$. Do đó, ta giả định đa thức tối tiểu có bậc $n = 2$.
2. Giá trị xấp xỉ: Gọi γ là giá trị xấp xỉ của $\sqrt{X}$ chính xác đến d chữ số thập phân. Tức là $|\gamma - \sqrt{X}| < 10^{-d}$.

3. Hằng số Trọng số: Để khuếch đại sai số, ta chọn hằng số $K = 10^d$.

4. Điều kiện Khả thi: Phương pháp này hiệu quả khi hệ số c_0 (tức là $-X$) có độ lớn nhỏ hơn đáng kể so với hằng số trọng số K. Ta cần $X \ll 10^d$.

Yêu cầu: Tìm vector hệ số $\mathbf{u} = (c_0, c_1, c_2)$ sao cho $c_0 + c_1\gamma + c_2\gamma^2 \approx 0$.

2.3.2 Xây dựng Ma trận Cơ sở B'

Ma trận cơ sở B' có kích thước 3×3 (tương ứng với $n+1$ hệ số). Các vector cơ sở được xây dựng để tìm quan hệ giữa $1, \gamma, \gamma^2$:

$$B' = \begin{pmatrix} 1 & 0 & \lfloor K \cdot 1 \rfloor \\ 0 & 1 & \lfloor K \cdot \gamma \rfloor \\ 0 & 0 & \lfloor K \cdot \gamma^2 \rfloor \end{pmatrix}$$

Một vector $\mathbf{v}$ trong Lattice được sinh bởi tổ hợp tuyến tính $\mathbf{u} = (c_0, c_1, c_2)$ là:

$$\mathbf{v} = (c_0, c_1, c_2) \cdot B' = \big(c_0, c_1, c_2,\ K \cdot (c_0 + c_1\gamma + c_2\gamma^2)\big)$$

2.3.3 Thuật toán LLL và Suy luận

1. **Tìm Vector Ngắn nhất:** Ta áp dụng Thuật toán Thu gọn Lattice LLL trên ma trận B'. Do $t^2 - X = 0$ có hệ số nguyên, vector $\mathbf{u}_{target} = (-X, 0, 1)$ tạo ra một vector $\mathbf{v}_{target}$ cực ngắn trong Lattice vì thành phần cuối cùng ϵ của nó rất nhỏ:

   $$\mathbf{v}_{target} = \big(-X, 0, 1,\ K \cdot (-X + \gamma^2)\big)$$

   Với $\gamma \approx \sqrt{X}$, $K \cdot (-X + \gamma^2) \approx 0$.

2. **Diễn giải Kết quả LLL:** Vector ngắn nhất tìm được $\mathbf{u}_{LLL}$ sẽ là bội số nguyên của $\mathbf{u}_{target}$ (thường là ± 1):

   $$\mathbf{u}_{LLL} = (u_0, u_1, u_2) \approx \pm(-X, 0, 1)$$

3. **Thu hồi X:** Ta suy luận ra số nguyên X từ các thành phần của vector ngắn nhất:

   - Hệ số bậc 1 phải bằng 0: $u_1 \approx 0$.
   - Hệ số bậc 2 phải bằng ± 1: $u_2 = \pm 1$.
   - Hệ số tự do phải bằng X hoặc $-X$: $u_0 = \pm X$.

   Do đó, số nguyên X cần tìm là giá trị tuyệt đối của thành phần đầu tiên của vector ngắn nhất $\mathbf{u}_{LLL}$:

   $$X = |u_0|$$

2.3.4 Kiểm chứng Đại số

Sau khi thu hồi được giá trị $X_{found} = |u_0|$, ta phải kiểm tra xem giá trị này có thỏa mãn điều kiện xấp xỉ ban đầu hay không:

$$|\sqrt{X_{found}} - \gamma| < 10^{-d}$$

Nếu điều kiện này thỏa mãn, ta kết luận X_{found} chính là số nguyên ban đầu.

3 Attacking the Vigenere Cipher

3.1 Giới thiệu bài toán

Bài toán yêu cầu giải mã một đoạn văn bản tiếng Anh bị mã hóa bằng thuật toán Vigenère Cipher. Điểm đặc biệt của thử thách này là mã hóa chỉ tác động lên các chữ cái (A-Z), trong khi các ký tự đặc biệt (dấu cách, dấu chấm, phẩy...) được giữ nguyên và không làm thay đổi vị trí của khóa.

Mục tiêu là tìm ra khóa bí mật và khôi phục văn bản gốc để lấy cờ.

3.2 Cơ sở lý thuyết thám mã

Để phá vỡ mã Vigenère mà không cần biết trước khóa, ta sử dụng hai phương pháp thống kê chính:

3.2.1 Chỉ số trùng khớp (Index of Coincidence - IoC)

IoC dùng để xác định độ dài của khóa (k). Nó đo xác suất để hai chữ cái chọn ngẫu nhiên từ văn bản là giống nhau.

$$IC = \frac{\sum_{i=A}^{Z} f_i(f_i - 1)}{N(N-1)}$$

- Tiếng Anh chuẩn có $IC \approx 0.067$.
- Văn bản ngẫu nhiên có $IC \approx 0.038$.

Ta thử chia văn bản thành k nhóm. Nếu k là độ dài khóa đúng, các nhóm con sẽ có phân phối tần suất giống tiếng Anh ($IC \approx 0.067$).

3.2.2 Phân tích tần suất (Chi-squared Test)

Sau khi biết độ dài khóa, ta chia văn bản thành k cột. Mỗi cột tương ứng với một mã Caesar. Ta dùng kiểm định Chi bình phương để tìm ký tự khóa cho từng cột:

$$\chi^2 = \sum_{i=A}^{Z} \frac{(O_i - E_i)^2}{E_i}$$

Giá trị χ^2 nhỏ nhất sẽ chỉ ra ký tự khóa chính xác nhất.

3.3 Quy trình thực hiện

3.3.1 Làm sạch dữ liệu (Data Cleaning)

Do thuật toán mã hóa bỏ qua ký tự đặc biệt, việc đầu tiên là phải loại bỏ toàn bộ dấu cách, xuống dòng, dấu câu ra khỏi văn bản mã hóa (Ciphertext), chỉ giữ lại các ký tự chữ cái (A-Z).

```
# Chi giu lai chu cai de phan tich thong ke
clean_text = "".join(filter(str.isalpha, ciphertext.upper()))
```

3.3.2 Tự động tìm khóa

Sử dụng thuật toán IoC để đoán độ dài khóa, sau đó dùng Chi-squared để tìm từng ký tự.

```
def solve_key(ciphertext):
    key_len = find_best_key_length(clean_text)

    # Tim tung ky tu cua khoa dua tren tan suat
    found_key = ""
    for i in range(key_len):
        # ... (Thuat toan Chi-squared) ...
    return found_key
```

3.3.3 Giải mã (Decryption)

Sau khi tìm được khóa, ta viết hàm giải mã. Lưu ý quan trọng: biến đếm vị trí khóa (key_idx) chỉ tăng khi gặp chữ cái.

```
def decrypt(self, ciphertext):
    plaintext = ""
    key_idx = 0
    for char in ciphertext:
        if char in self.alphabet:
            # Giai ma va tang key_idx
            # ...
            key_idx += 1
        else:
            # Giu nguyen ky tu va KHONG tang key_idx
            plaintext += char
    return plaintext
```

3.4 Toàn bộ code và chạy với file Ciphertext1.txt

```
import string
from collections import Counter

# --- PHAN 1: DU LIEU TAN SUAT TIENG ANH ---
ENGLISH_FREQ = {
    'A': 0.08167, 'B': 0.01492, 'C': 0.02782, 'D': 0.04253, 'E':
       0.12702,
    'F': 0.02228, 'G': 0.02015, 'H': 0.06094, 'I': 0.06966, 'J':
       0.00153,
    'K': 0.00772, 'L': 0.04025, 'M': 0.02406, 'N': 0.06749, 'O':
       0.07507,
    'P': 0.01929, 'Q': 0.00095, 'R': 0.05987, 'S': 0.06327, 'T':
       0.09056,
    'U': 0.02758, 'V': 0.00978, 'W': 0.02360, 'X': 0.00150, 'Y':
       0.01974,
    'Z': 0.00074
}

ALPHABET = "abcdefghijklmnopqrstuvwxyz"
ALPHABET_UPPER = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

```
16
17  # --- PHAN 2: CAC HAM TIM KEY TU DONG ---
18
19  def get_index_of_coincidence(text):
20      """Tinh chi so trung khop (IoC)"""
21      N = len(text)
22      if N <= 1: return 0
23      counts = Counter(text)
24      numerator = sum(n * (n - 1) for n in counts.values())
25      return numerator / (N * (N - 1))
26
27  def find_best_key_length(text, max_len=15):
28      # Tim do dai Key co IoC gan voi tieng Anh nhat (0.067)
29      best_len = 1
30      best_diff = float('inf')
31
32      for k in range(1, max_len + 1):
33          groups = ['' for _ in range(k)]
34          for i, char in enumerate(text):
35              groups[i % k] += char
36
37          avg_ioc = sum(get_index_of_coincidence(g) for g in groups) / k
38
39          # Tieng Anh chuan co IoC khoang 0.0667
40          diff = abs(avg_ioc - 0.0667)
41          if diff < best_diff:
42              best_diff = diff
43              best_len = k
44
45      return best_len
46
47  def solve_key(ciphertext):
48      """Ham tong hop de tim Key tu van ban ma hoa"""
49      # Chi giu lai chu cai de phan tich
50      clean_text = "".join(filter(str.isalpha, ciphertext.upper()))
51
52      # 1. Tim do dai (gioi han 15 de tranh lap key)
53      key_len = find_best_key_length(clean_text, max_len=15)
54      print(f" -> Do dai Key du doan: {key_len}")
55
56      # 2. Tim tung ky tu cua Key
57      found_key = ""
58      for i in range(key_len):
59          sub_text = clean_text[i::key_len] # Lay cac ky tu o vi tri
                chia het cho key_len
60          min_chi = float('inf')
61          best_shift = 0
62
63          # Thu 26 phep dich (Caesar) cho nhom ky tu nay
64          for shift in range(26):
65              shifted_counts = Counter()
66              for char in sub_text:
67                  # Dich nguoc de tim ky tu goc
```

```
                orig = chr((ord(char) - ord('A') - shift) % 26 + ord('
                    A'))
                shifted_counts[orig] += 1

            # Tinh Chi-squared (do lech so voi tan suat chuan)
            chi_sq = 0
            for char in string.ascii_uppercase:
                observed = shifted_counts[char]
                expected = len(sub_text) * ENGLISH_FREQ[char]
                # Tranh chia cho 0
                if expected > 0:
                    chi_sq += ((observed - expected) ** 2) / expected

            if chi_sq < min_chi:
                min_chi = chi_sq
                best_shift = shift

        found_key += chr(best_shift + ord('A'))

    return found_key

# --- PHAN 3: CLASS GIAI MA ---

class VigenereCipher:
    def __init__(self, key):
        self.key = key.lower()
        self.alphabet = ALPHABET

    def decrypt(self, ciphertext: str) -> str:
        plaintext = ""
        key_idx = 0

        for char in ciphertext:
            # 1. Neu la chu thuong
            if char in self.alphabet:
                char_idx = self.alphabet.index(char)
                key_char = self.key[key_idx % len(self.key)]
                key_shift = self.alphabet.index(key_char)

                new_idx = (char_idx - key_shift) % 26
                plaintext += self.alphabet[new_idx]
                key_idx += 1

            # 2. Neu la chu Hoa
            elif char in ALPHABET_UPPER:
                char_idx = ALPHABET_UPPER.index(char)
                key_char = self.key[key_idx % len(self.key)]
                key_shift = self.alphabet.index(key_char)

                new_idx = (char_idx - key_shift) % 26
                plaintext += ALPHABET_UPPER[new_idx]
                key_idx += 1

```

```
            # 3. Neu la ky tu la
            else:
                plaintext += char

        return plaintext

# --- PHAN 4: CHAY CHUONG TRINH ---
if __name__ == "__main__":

    # 1. Doc file

    raw_text = open("ciphertext.txt", "r").read()

    found_key = solve_key(raw_text)
    print(f"\nTIM THAY KEY: {found_key}")

    # 3. Giai ma bang Key vua tim duoc
    print(f"Giai ma voi Key '{found_key}'...")
    cipher = VigenereCipher(found_key)
    decrypted_text = cipher.decrypt(raw_text)

    # 4. Hien thi ket qua
    print("\n--- TOAN BO NOI DUNG DA GIAI MA ---")
    print(decrypted_text)

    # 5. Luu ra file
    with open("final_result.txt", "w") as f:
        f.write(decrypted_text)
    print("\n(Da luu ket qua vao file 'final_result.txt')")
```

```
nhujoi@LAPTOP-73BAU1IC:/mnt/c/Users/ADMIN/Downloads/UwU/Assign-ACCT/Problem2$ python3 Problem2.py
 -> Do dai Key du doan: 10

TIM THAY KEY: HEAVENHELL
Giai ma voi Key 'HEAVENHELL'...

--- TOAN BO NOI DUNG DA GIAI MA ---
The concept of the afterlife has fascinated humanity for millennia, shaping cultures, religions, and philosophies across civilizations. At its core, the afterlife re
ers 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
he 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
 rich diversity of interpretations. From ancient mythologies to modern spiritual movements, the afterlife remains a central theme in understanding the nature of exis
ence, 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 di
ine 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 we
ghed 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 Me
dows 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, reinfor
ing 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 r
incarnation—a cyclical process where the soul is reborn into new forms based on karma, the moral sum of one's actions. This cycle, known as samsara, continues until
nlightenment or liberation (moksha or nirvana) is attained, freeing the soul from the endless cycle of birth and death. Such perspectives emphasize spiritual growth
nd moral responsibility, suggesting that life and death are not opposites but stages within a continuous spiritual journey. Meanwhile, in Abrahamic religions—Judaism
 Christianity, and Islam—the afterlifeon of one's influence, memories, and legacy in the world of the living. Others explore it through the lens of science and metap
ysics, examining phenomena such as near-death experiences and consciousness beyond the brain. While empirical evidence remains elusive, these inquiries reflect human
ty's enduring quest to understand the unknown. Whether interpreted as a spiritual realm, a cycle of rebirth, or a symbolic continuation of existence, the afterlife o
fers profound comfort and meaning, bridging the gap between life and death. Ultimately, belief in the afterlife continues to inspire hope, moral reflection, and the
nduring human desire to find purpose beyond mortality.

(Da luu ket qua vao file 'final_result.txt')
```

Hình 2: *Ket qua sau khi chay chuong trinh*

3.5 Kết quả thực nghiệm

Sau khi chạy chương trình Python đã xây dựng, ta thu được kết quả như sau:

3.5.1 Khóa tìm được

Chương trình xác định độ dài khóa là 10 và nội dung khóa là: VOCABULARY

3.5.2 Văn bản giải mã

Nội dung văn bản sau khi giải mã (trích đoạn đầu):

> *"Risk management is a systematic process of identifying, analyzing, and responding to project risks..."*

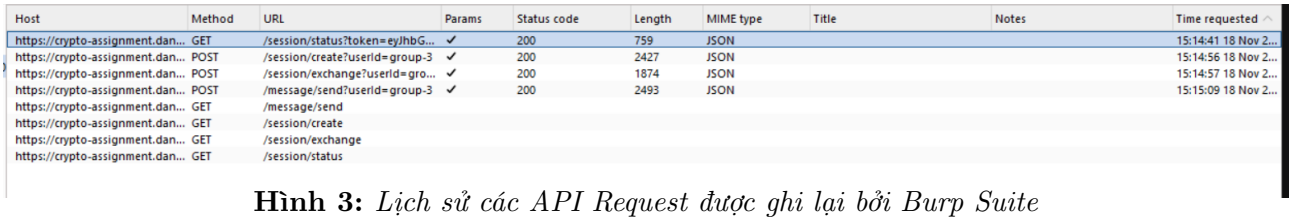
4 SMC Exploitation

4.1 Interception & API Analysis

4.1.1 API Table

Để phân tích giao thức truyền thông của thành phần Secure Messaging Component (SMC), nhóm thực hiện đã cấu hình Burp Suite hoạt động như một Intercepting Proxy và cài đặt chứng chỉ CA (Certificate Authority) trên thiết bị Android giả lập. Điều này cho phép đứng giữa (Man-in-the-Middle) để giải mã và bắt toàn bộ lưu lượng HTTPS giữa Client và Server.

Hình 3 dưới đây liệt kê lịch sử các Request được ghi nhận trong quá trình thực hiện một kịch bản đầy đủ: từ lúc mở ứng dụng, thực hiện bắt tay (handshake), đến khi gửi tin nhắn thành công. Từ hàng loạt các gói tin, chúng tôi đã lọc và xác định được 4 API Endpoint cốt lõi đóng vai trò quan trọng trong cơ chế bảo mật này.



Host	Method	URL	Params	Status code	Length	MIME type	Title	Notes	Time requested
https://crypto-assignment.dan...	GET	/session/status?token=eyJhbG...	✓	200	759	JSON			15:14:41 18 Nov 2...
https://crypto-assignment.dan...	POST	/session/create?userId=group-3	✓	200	2427	JSON			15:14:56 18 Nov 2...
https://crypto-assignment.dan...	POST	/session/exchange?userId=gro...	✓	200	1874	JSON			15:14:57 18 Nov 2...
https://crypto-assignment.dan...	POST	/message/send?userId=group-3	✓	200	2493	JSON			15:15:09 18 Nov 2...
https://crypto-assignment.dan...	GET	/message/send							
https://crypto-assignment.dan...	GET	/session/create							
https://crypto-assignment.dan...	GET	/session/exchange							
https://crypto-assignment.dan...	GET	/session/status							

Hình 3: *Lịch sử các API Request được ghi lại bởi Burp Suite*

Dựa trên kết quả phân tích từ Burp Suite, bảng dưới đây tóm tắt chức năng của các API Endpoint chính được sử dụng trong giao thức:

Method	Endpoint (URI)	Chức năng chính
GET	`/session/status`	Kiểm tra trạng thái phiên làm việc hiện tại (session) xem còn hiệu lực hay đã hết hạn.
POST	`/session/create`	Khởi tạo quy trình bắt tay (Handshake), gửi thuật toán mã hóa đề xuất và nhận về Public Key của Server.
POST	`/session/exchange`	Gửi Public Key (Ephemeral) của Client để thực hiện thỏa thuận khóa và xác thực chữ ký hai chiều.
POST	`/message/send`	Gửi tin nhắn đã được mã hóa (Encrypted) và ký số (Signed) lên Server.

Bảng 2: *Danh sách các API Endpoint quan trọng trong giao thức SMC*

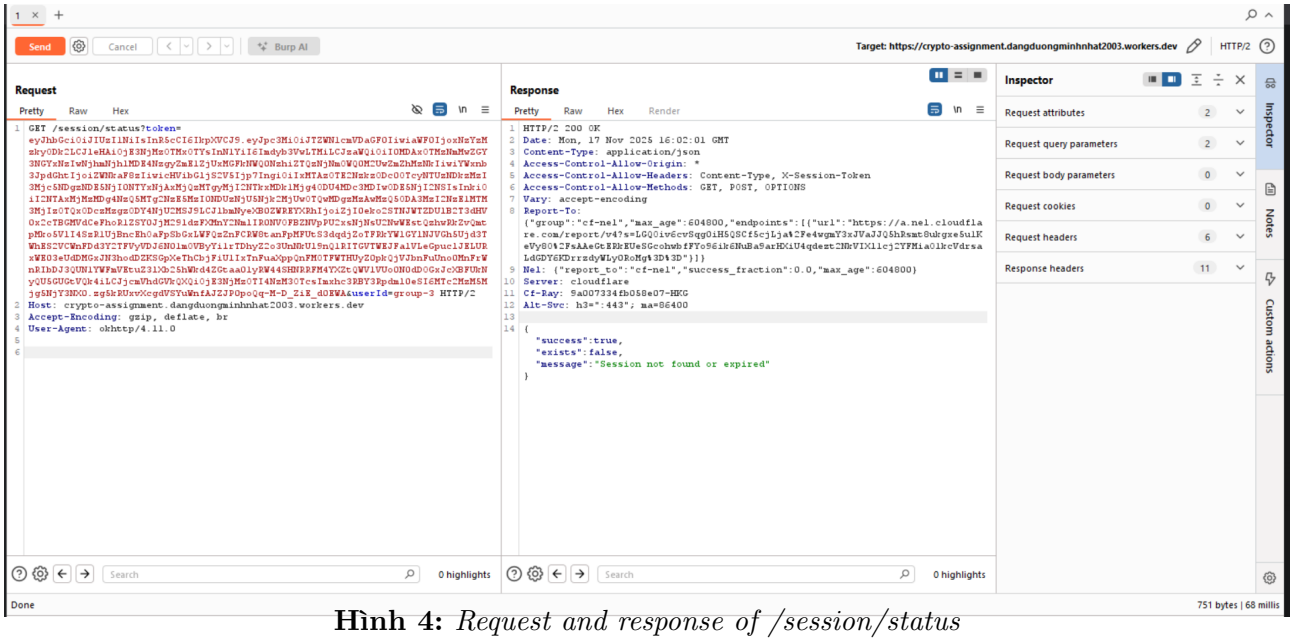
Quy trình hoạt động và quan sát kỹ thuật:

Quan sát lưu lượng mạng, nhóm nhận thấy quy trình giao tiếp tuân theo trình tự nghiêm ngặt:

- **Khởi tạo:** Đầu tiên, ứng dụng gọi `/session/status`. Nếu phiên không tồn tại hoặc hết hạn, quy trình tạo phiên mới được kích hoạt thông qua `/session/create` và `/session/exchange`.
- **Bảo mật đường truyền:** Tất cả các API đều sử dụng giao thức HTTPS (Port 443) để bảo vệ dữ liệu ở lớp mạng (Transport Layer).
- **Định dạng dữ liệu:** Client và Server giao tiếp hoàn toàn bằng định dạng JSON (`Content-Type: application/json`).
- **Định danh:** Các tham số như `userId` và `group` thường xuyên xuất hiện trên URL (Query String) để định tuyến request.

4.1.2 Status

Sau khi ứng dụng khởi động và người dùng nhập `userId` vào để đăng nhập, trước khi thực hiện bất kì giao thức bắt tay nào, client thực hiện kiểm tra session cũ (nếu có) còn hiệu lực hay không. Nếu không thì sẽ đi đến bước tiếp theo để thực hiện tiếp, còn nếu có sẽ restore session đang có sẵn để tiếp tục trò chuyện. Điều này giúp tối ưu hóa hiệu năng, tránh việc tạo lại session không cần thiết.



Hình 4: *Request and response of /session/status*

Field Name	Data Type	Explanation
`success`	Boolean	Cho biết yêu cầu API có được server xử lý thành công hay không (‘true‘ nghĩa là không có lỗi hệ thống).
`exists`	Boolean	Xác định token phiên làm việc còn hiệu lực hay không. Giá trị ‘false‘ báo hiệu phiên đã hết hạn hoặc không tồn tại, buộc Client phải thực hiện lại quy trình bắt tay.
`message`	String	Thông báo trạng thái từ server (ví dụ: "Session not found or expired") giúp debug lỗi dễ dàng hơn.

Bảng 3: *Giải thích các trường dữ liệu trong API `/session/status`*

- **Request**: Request được gửi dưới dạng GET đến endpoint `/session/status`. Quan sát trên URL ta có thể thấy rằng Client gửi kèm `savedSessionToken` và `userId` dưới dạng query parameters để Server định danh.
- **Response**: Trong hình 4, Server phản hội với mã 200 OK và JSON chứa body `exists: false`, điều này có nghĩa là Server đã kiểm tra và không tìm thấy token ở phía mình. Thông điệp nhằm ra tín hiệu cho Client biết token này không tồn tại hoặc đã hết hạn.

```
// LoginActiviy (line 484 - 488)
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (RootDetection.isDeviceRooted(this)) {
        showRootDetectedDialog();
        return;
    }
    setContentView(R.layout.activity_login);
    initViews();
    setupUI();
    this.client = buildClient();

    // Take the saved information
    String savedUserId = this.prefs.getString("userId", HttpUrl.
       FRAGMENT_ENCODE_SET);
    String savedSessionToken = this.prefs.getString("sessionToken",
       HttpUrl.FRAGMENT_ENCODE_SET);

    // If exist, try to restore session
    if (!savedUserId.isEmpty() && !savedSessionToken.isEmpty()) {
        attemptSessionRestore(savedUserId, savedSessionToken);
    }
}
```

Listing 1: Hàm onCreate kiểm tra savedSessionToken

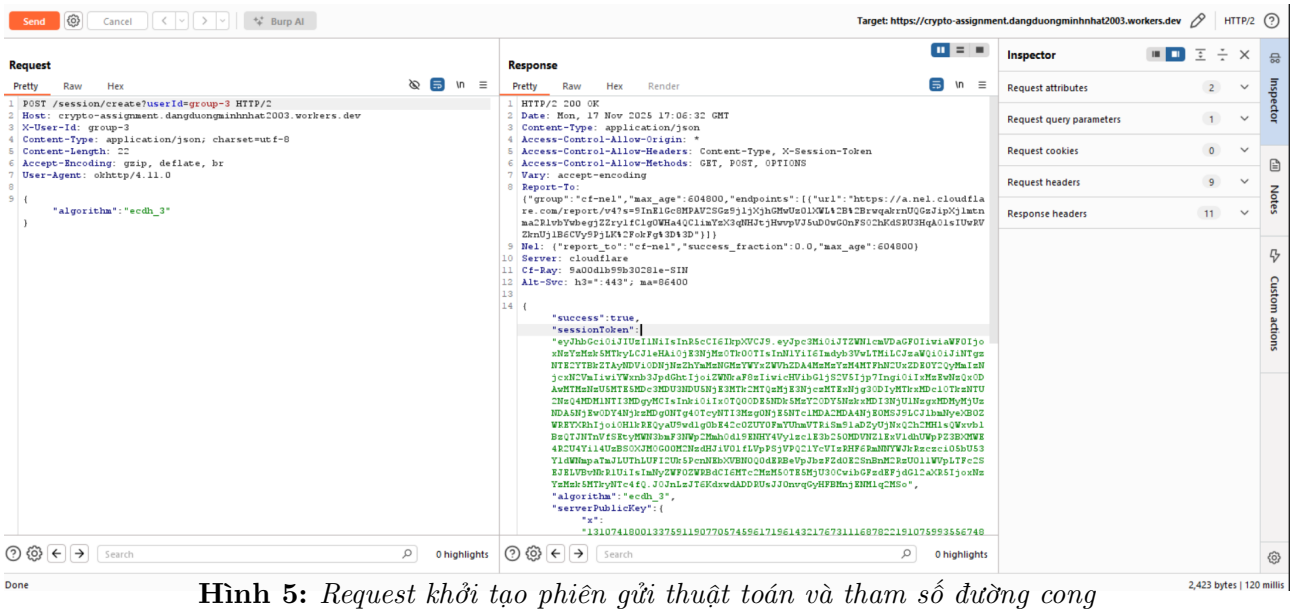
Tại đây, dòng 15-16 có thể thấy rằng hàm thực hiện lấy `sessionToken` đã lưu. Nếu cả 2 giá trị này đều tồn tại thì sẽ thử restore session.

```
// LoginActiviy (line 491 - 557)
private void attemptSessionRestore(String savedUserId, String
   savedSessionToken) {
    showProgress("Restoring session...");
    Request request = new Request.Builder()
        .url("https://.../session/status?token=" + savedSessionToken +
            "&userId=" + savedUserId)
        .get()
        .build();
    this.client.newCall(request).enqueue(new AnonymousClass4(
       savedUserId, savedSessionToken));
}
```

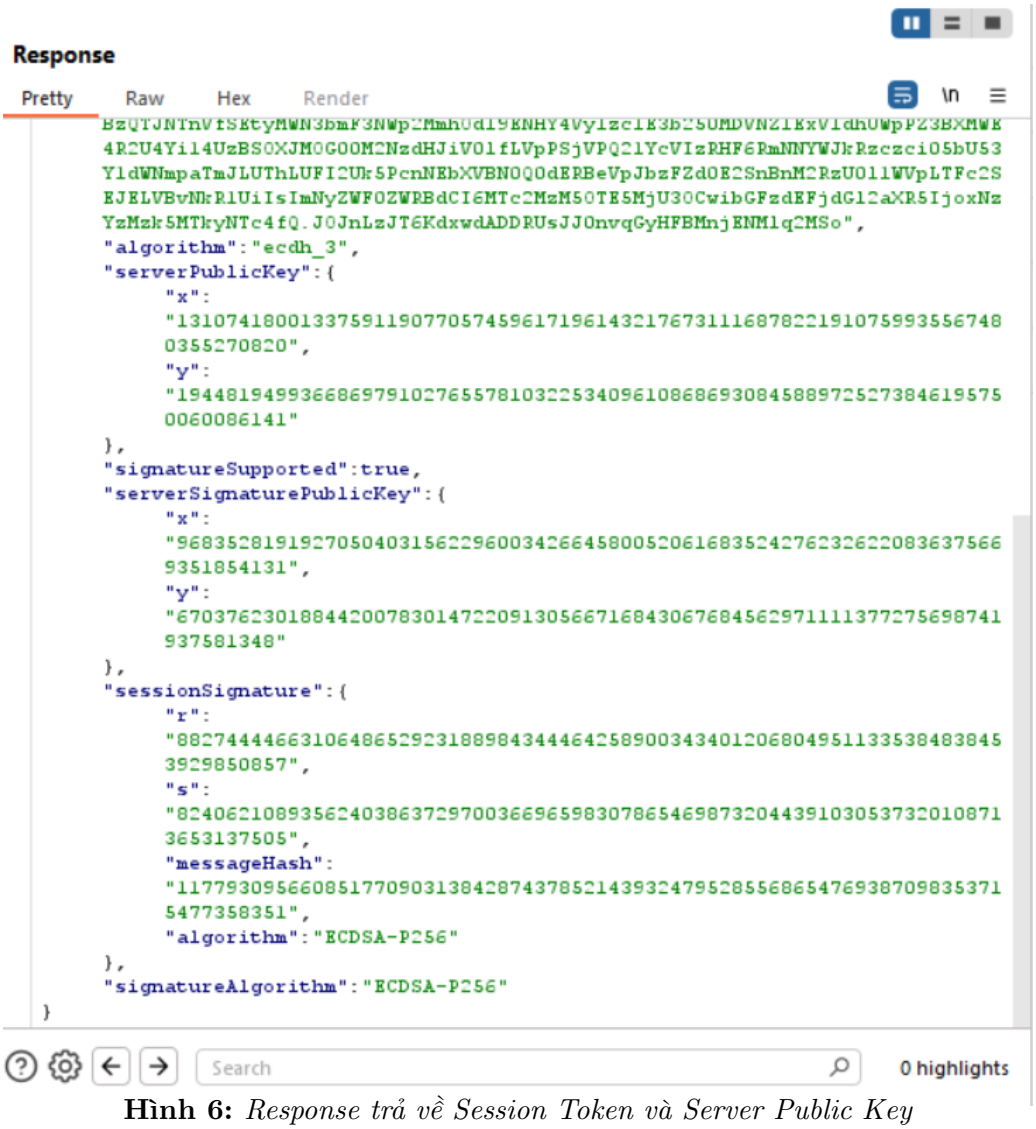
Listing 2: Hàm attemptSessionRestore kiểm tra savedSessionToken

4.1.3 Create Session

Sau khi xác định phiên làm việc cũ không còn hiệu lực (kết quả trả về từ `/session/status` là `exists: false`), Client tiến hành khởi tạo một phiên mới. Request được gửi dưới dạng POST đến endpoint `/session/create` kèm theo các thông số cấu hình mật mã mong muốn. Server sau đó phản hồi các thông tin thiết yếu để thiết lập kênh truyền an toàn, bao gồm Session Token và Khóa công khai của Server.



Hình 5: *Request khởi tạo phiên gửi thuật toán và tham số đường cong*



Hình 6: *Response trả về Session Token và Server Public Key*

Dựa trên kết quả phân tích gói tin, bảng dưới đây giải thích chi tiết các trường dữ liệu quan trọng trong quá trình khởi tạo phiên:

Field Name	Data Type	Explanation
Request (Client gửi lên)		
`algorithm`	String	Tên thuật toán thỏa thuận khóa mà Client đề xuất (quan sát được là `"ecdh_3"`).
`curveParameters`	Object	Các tham số của đường cong Elliptic (P-256) mà Client sử dụng để sinh khóa: $p, a, b, G_x, G_y, order$.
Response (Server trả về)		
`serverPublicKey`	Object	Khóa công khai của Server (Q_S) bao gồm tọa độ x và y, dùng để tính Shared Secret.
`sessionToken`	String (JWT)	Token định danh phiên làm việc, chứa thông tin xác thực và thời gian hết hạn.
`sessionSignature`	Object	Chữ ký số của Server ký lên các tham số phiên, đảm bảo Client đang giao tiếp với Server thật (chống tấn công MITM).

Bảng 4: *Giải thích các trường dữ liệu API `/session/create`*

Về mặt cài đặt, quá trình này được xử lý trong hàm `createSession` của `LoginActiviy`. Client thực hiện hardcode các tham số đường cong và gửi lên Server:

```
// LoginActiviy (line 176 - 217)
private void createSession() throws JSONException {
    String algorithm = AlgorithmSelector.getAlgorithmForUser(this.
       userId);

    JSONObject requestBody = new JSONObject();
    requestBody.put("algorithm", algorithm);

    if ("ecdh".equals(algorithm) || "ecdh_2".equals(algorithm)) {
        // ... Configure curve parameters ...
    }

    RequestBody body = RequestBody.create(requestBody.toString(),
       MediaType.parse("application/json"));
    Request request = new Request.Builder()
            .url("https://.../session/create?userId=" + this.userId)
            .addHeader("x-user-id", this.userId)
            .post(body)
            .build();

    this.client.newCall(request).enqueue(new AnonymousClass2());
}
```

Listing 3: Hàm createSession khởi tạo kết nối

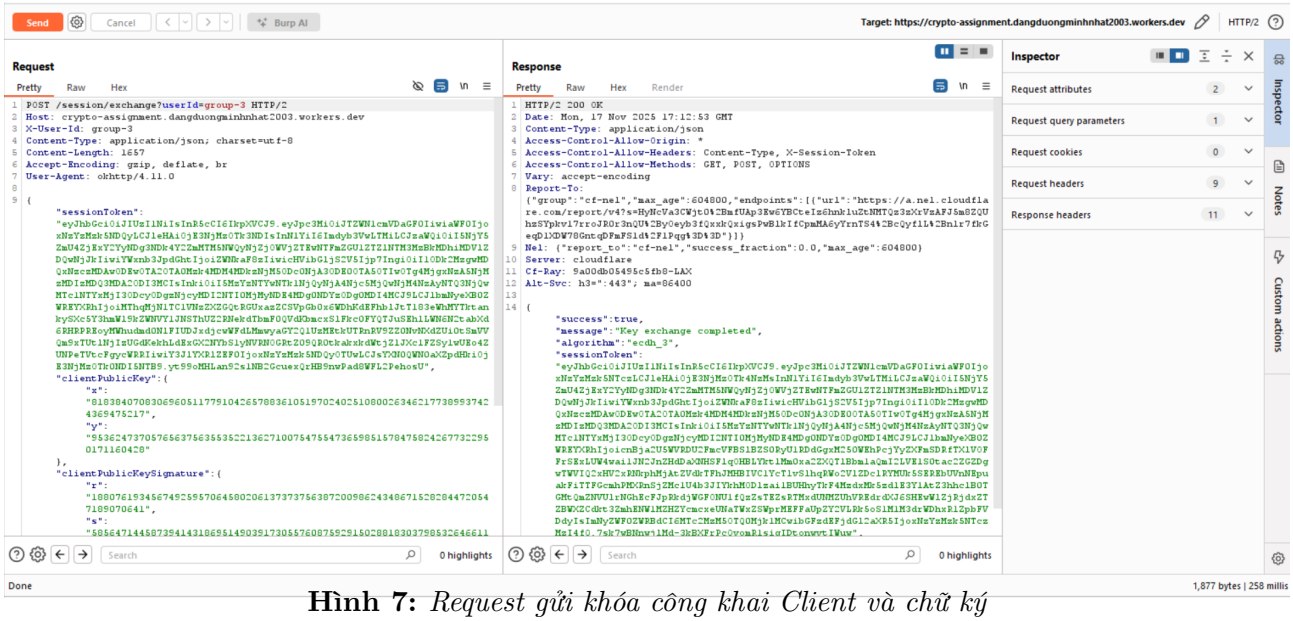
Khi nhận được phản hồi, Client sẽ trích xuất `sessionToken`, `serverPublicKey` và quan trọng nhất là kiểm tra chữ ký số (`sessionSignature`) để xác thực Server trước khi tiến hành bước trao đổi khóa tiếp theo:

```
// LoginActiviy (line 220 - 369)
```

```java
public void onResponse(Call call, Response response) throws IOException {
    // ... (responseBody) ...

    JSONObject jsonResponse = new JSONObject(finalResponseBody);

    if (!jsonResponse.getBoolean("success")) return;

    // 1. Session Token (JWT) and Server Public Key
    this.sessionToken = jsonResponse.getString("sessionToken");
    JSONObject serverPublicKey = jsonResponse.getJSONObject("serverPublicKey");

    // 2. Check security: Server MUST have signature
    if (!jsonResponse.has("sessionSignature")) {
        // ... Security alert ...
        return;
    }

    // 3. Signature Verification
    JSONObject sessionSignatureJson = jsonResponse.getJSONObject("sessionSignature");
    JSONObject serverSigPubKey = jsonResponse.getJSONObject("serverSignaturePublicKey");

    //  Verify info from JWT
    String sessionId = extractSessionIdFromJWT(this.sessionToken);
    JSONObject sessionData = new JSONObject();
    sessionData.put("sessionId", sessionId);
    // ... Add userId, createdAt ...

    // Call CryptoManager to verify
    boolean verified = this.cryptoManager.verifyServerSignature(
            sessionData.toString(),
            sessionSignatureJson,
            serverSigPubKey
    );

    if (verified) {
        // 4. Right signature -> Start exchange key
        this.performKeyExchange(serverPublicKey);
    } else {
        // Man-in-the-Middle alert
        Log.e(TAG, "SECURITY ALERT: Session signature verification FAILED!");
    }
}
```

Listing 4: Xử lý phản hồi và xác thực chữ ký Server

4.1.4 Checking exchange session



Hình 7: *Request gửi khóa công khai Client và chữ ký*

Hình 8: *Response xác nhận trao đổi khóa thành công*

Field Name	Data Type	Explanation
Request (Client gửi lên)		
`clientPublicKey`	Object	Chứa khóa công khai ECDH tạm thời (ephemeral) do Client sinh ra cho phiên này.
`.x`, `.y`	String (Hex)	Tọa độ x và y của điểm khóa công khai Q_C trên đường cong Elliptic.
`client...Signature`	Object	(clientPublicKeySignature) Chữ ký số của Client ký lên khóa công khai vừa sinh ra. Điều này đảm bảo tính toàn vẹn và ngăn chặn kẻ tấn công thay đổi khóa trên đường truyền.
`.r`, `.s`	String (Int)	Hai thành phần của chữ ký ECDSA (r, s) do Client tạo ra.
`.messageHash`	String (Hex)	Giá trị băm của thông điệp (khóa công khai) mà Client đã ký.
`.algorithm`	String	Tên thuật toán trao đổi khóa mà Client đề xuất sử dụng (quan sát được là `"ECDSA-P256"`).
`client...Key`	Object	(clientSignaturePublicKey) Khóa công khai xác thực của Client. Server sẽ dùng khóa này để kiểm tra chữ ký `clientPublicKeySignature` nhằm xác minh danh tính Client.
Response (Server trả về)		
`success`	Boolean	Xác nhận quá trình trao đổi khóa đã hoàn tất thành công (True).
`message`	String	Thông báo từ server: `"Key exchange completed"`.
`algorithm`	String	Thuật toán trao đổi khóa được sử dụng (quan sát được là `"ecdh_3"`).
`sessionToken`	String (JWT)	Đây là một chuỗi JSON Web Token (JWT) rất dài (bắt đầu bằng `ey...`). - Nó chứa **payload** bên trong bao gồm: Khóa công khai của Server (Q_S), Chữ ký số của Server (r, s), và thời hạn phiên. - Client sẽ phải giải mã (decode) token này để lấy các tham số đó phục vụ việc tính Shared Secret.
`client...Verified`	Boolean	(clientSignatureVerified) Server xác nhận rằng nó đã kiểm tra chữ ký điện tử của Client gửi lên (trong Request) và thấy hợp lệ. Điều này hoàn tất việc xác thực danh tính Client.

Bảng 5: *Giải thích chi tiết Request và Response của API `/session/exchange`*

```
// LoginActiviy (line 372 - 405)
public void performKeyExchange(JSONObject serverPublicKey) throws
    JSONException {
    // 1. Client creating ECDH key (Ephemeral Key)
```

```
    this.cryptoManager.generateKeyPair();
    JSONObject clientPublicKey = this.cryptoManager.getPublicKeyJson();

    // 2. Calculating Shared Secret right after received Server Public Key
    this.cryptoManager.computeSharedSecret(serverPublicKey);

    JSONObject requestBody = new JSONObject();
    requestBody.put("sessionToken", this.sessionToken);
    requestBody.put("clientPublicKey", clientPublicKey);

    // 3. Sign: Client sign into their own Public Key to verify
    // IMPORTANT TO PREVENT Man-in-the-Middle
    String publicKeyString = clientPublicKey.toString();
    CryptoManager.SignatureWithPublicKey signResult =
            this.cryptoManager.signMessageEphemeral(publicKeyString);

    // 4. Compress signature and verify key to Server
    requestBody.put("clientPublicKeySignature", signResult.signature.toJSON());
    requestBody.put("clientSignaturePublicKey", signResult.publicKey);

    // 5. send Request POST to /session/exchange
    Request request = new Request.Builder()
            .url(".../session/exchange?userId=" + this.userId)
            .addHeader("x-user-id", this.userId)
            .post(RequestBody.create(requestBody.toString(), ...))
            .build();

    this.client.newCall(request).enqueue(new AnonymousClass3());
}
```

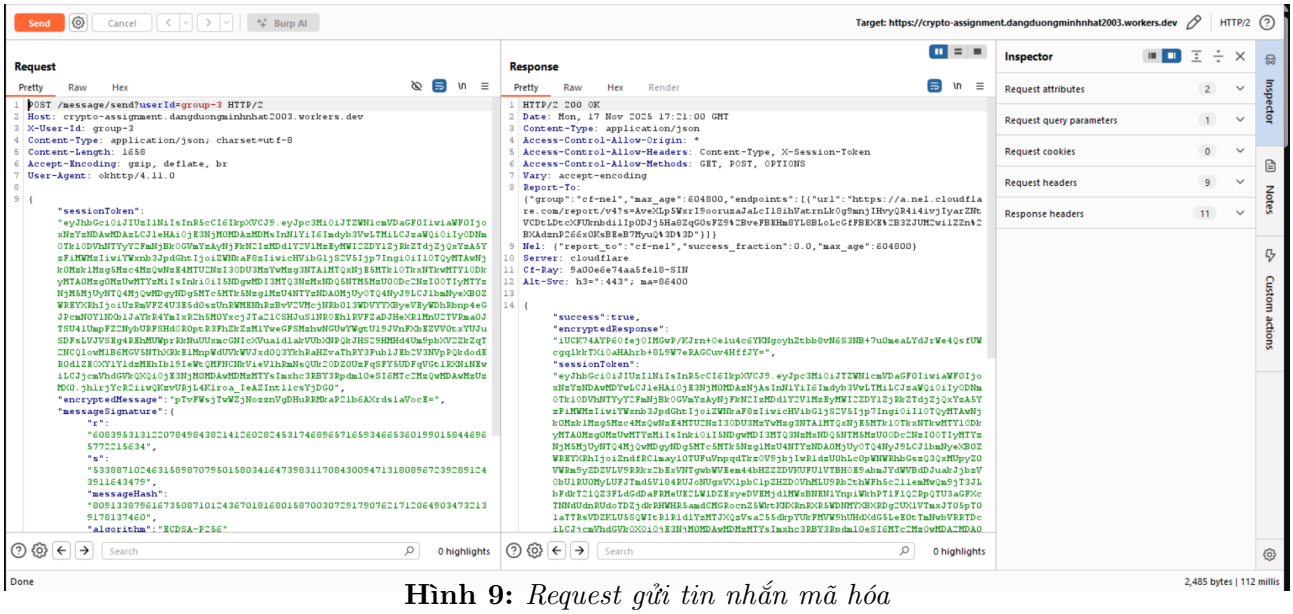
Listing 5: Hàm performKeyExchange: Client gửi khóa và chữ ký

```
// LoginActiviy (line 408 - 482)
public void onResponse(Call call, Response response) throws IOException {
    // ... (response body) ...

    if (isSuccessful) {
        JSONObject jsonResponse = new JSONObject(finalResponseBody);

        // 1. Update Session Token (if changed)
        this.sessionToken = jsonResponse.optString("sessionToken", this.sessionToken);

        // 2. Mutual Authentication
        if (!jsonResponse.has("clientSignatureVerified")) {
            // Error: Server can not verify Client
            return;
        }

        boolean clientSigVerified = jsonResponse.getBoolean("
```

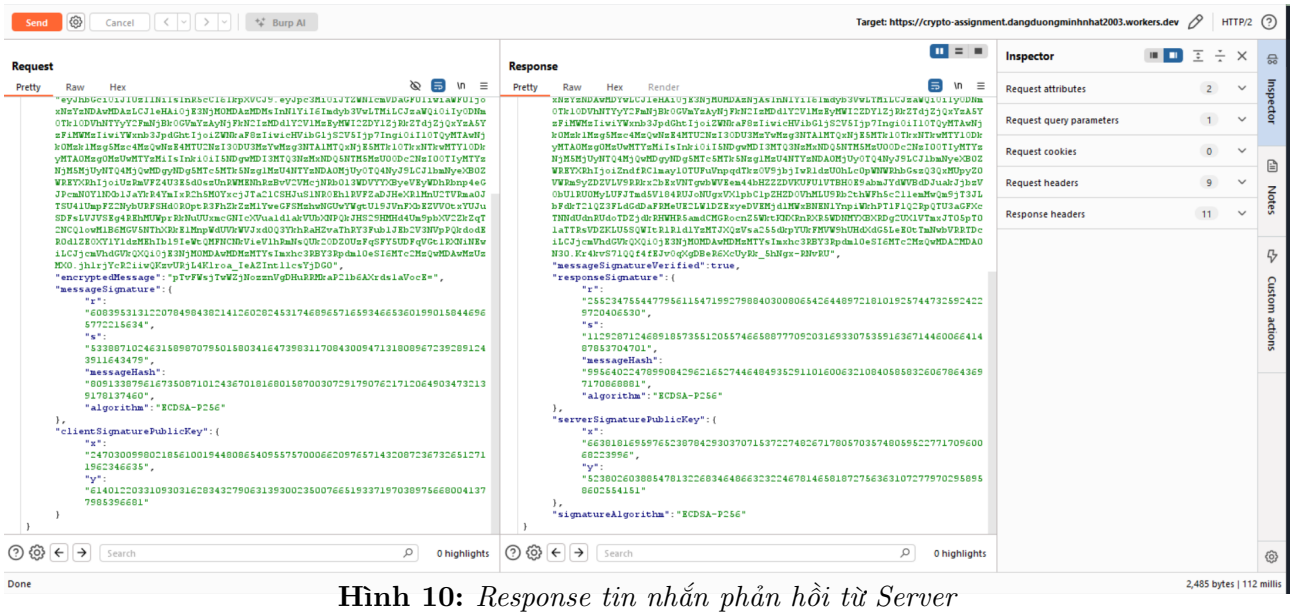
```
            clientSignatureVerified");

        if (clientSigVerified) {
            // 3. Completed: Server accepted Client
            if (this.cryptoManager.isKeyExchangeComplete()) {
                this.saveSuccessfulLogin();        // Save token
                this.proceedToChatActivity();      // Chat
            }
        } else {
            Log.e(TAG, "Server rejected client signature!");
        }
    }
}
```

Listing 6: Xử lý phản hồi xác thực từ Server

4.1.5 Messaging



Hình 9: *Request gửi tin nhắn mã hóa*



Hình 10: *Response tin nhắn phản hồi từ Server*

Field Name	Data Type	Explanation
Request		
`sessionToken`	String (JWT)	Token phiên làm việc hiện tại để xác thực người dùng.
`encryptedMessage`	String (Base64)	Nội dung tin nhắn đã được mã hóa (AES-GCM hoặc AES-CBC). Độ dài của trường này thay đổi dựa trên độ dài tin nhắn gốc.
`messageSignature`	Object	Chữ ký số của Client ký lên `encryptedMessage`, đảm bảo tính chống chối bỏ và toàn vẹn.
`client...Key`	Object	(clientSignaturePublicKey) Khóa công khai dùng để verify chữ ký tin nhắn.
Response		
`encryptedResponse`	String (Base64)	Phản hồi từ Server đã được mã hóa.
`responseSignature`	Object	Chữ ký số của Server ký lên phản hồi, giúp Client xác nhận phản hồi này thực sự đến từ Server.

Bảng 6: *Giải thích các trường dữ liệu API* `/message/send`

```java
private void sendMessage(String message) throws JSONException {
    JSONObject requestBody = new JSONObject();
    requestBody.put("sessionToken", this.sessionToken);

    try {
        // Check algorithm
        // If ecdh_3 -> Change to CBC (Block Cipher with Padding)
        // Step function
        if ("ecdh_3".equals(AlgorithmSelector.getAlgorithmForUser(this
            .userId))) {
            this.cryptoManager.setEncryptionMode("CBC");
        }

        // 1. Encrypting message
        // encryptedMessage same length with ciphertext
        String encryptedMessage = this.cryptoManager.encrypt(message);
        requestBody.put("encryptedMessage", encryptedMessage);

        // 2. Digital Signature
        if (this.cryptoManager.isSignatureSupported()) {
            // Encrypt-then-Sign
            CryptoManager.SignatureWithPublicKey signResult =
                    this.cryptoManager.signMessageEphemeral(
                        encryptedMessage);

            requestBody.put("messageSignature", signResult.signature.
                toJSON());
            requestBody.put("clientSignaturePublicKey", signResult.
                publicKey);
        }

```

```
        // 3. Send Request
        RequestBody body = RequestBody.create(requestBody.toString(),
            ...);
        Request request = new Request.Builder()
                .url(".../message/send?userId=" + this.userId)
                .post(body)
                .build();

        this.client.newCall(request).enqueue(new AnonymousClass2());

    } catch (Exception e) {
        // ...
    }
}
```

Listing 7: Hàm sendMessage: Mã hóa và đóng gói tin nhắn

```
// Class API /message/send
class AnonymousClass2 implements Callback {
    @Override
    public void onResponse(Call call, final Response response) throws
        IOException {
        if (response.isSuccessful()) {
            JSONObject jsonResponse = new JSONObject(responseBody);

            // 1. Rolling Session
            String newToken = jsonResponse.optString("sessionToken",
                null);
            if (newToken != null) {
                this.sessionToken = newToken;
            }

            // 2. Check integrity
            if (jsonResponse.has("responseSignature")) {
                String encryptedResponse = jsonResponse.getString("
                    encryptedResponse");
                JSONObject respSigJson = jsonResponse.getJSONObject("
                    responseSignature");
                JSONObject serverEphemeralPubKey = jsonResponse.
                    getJSONObject("serverSignaturePublicKey");

                // Verify server's signature
                boolean verified = cryptoManager.
                    verifySignatureWithPublicKey(
                        encryptedResponse,
                        SignatureBase.Signature.fromJSON(...),
                        serverEphemeralPubKey
                );

                if (!verified) {
                    Log.e(TAG, "SECURITY ALERT: Invalid signature!");
                    return;
                }

```

```
32                  // 3. Decrypting
33                  String decryptedResponse = cryptoManager.decrypt(
                        encryptedResponse);
34                  addMessage(decryptedResponse, false);
35              }
36          }
37      }
38  }
```

Listing 8: Xử lý phản hồi tin nhắn và xác thực chữ ký

4.2 Re-implementation & Protocol Reconstruction

4.2.1 General protocol

Dựa vào quá trình phân tích lưu lượng mạng và mã nguồn giải mã được, chúng tôi đã tái dựng lại toàn bộ logic của phía client. Giao thức Secure Messaging Component (SMC) hoạt động dựa trên cơ chế hybrid encryption, kết hợp giữa ECDH để trao đổi khóa phiên và AES-CBC để mã hóa dữ liệu.

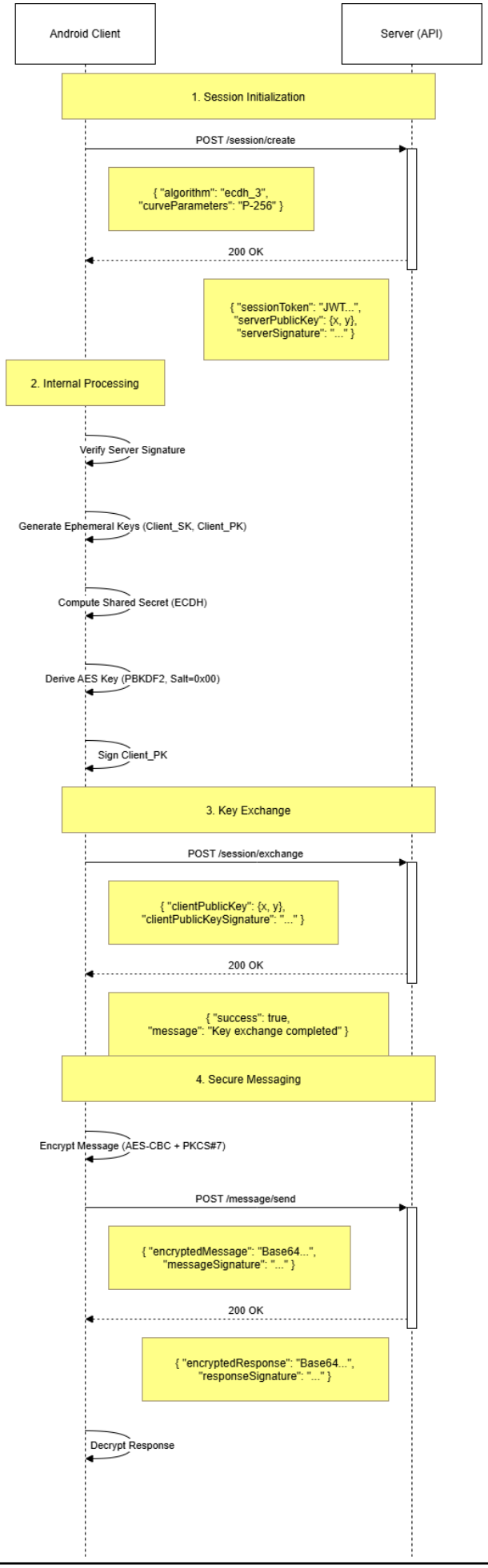
Quy trình hoạt động bao gồm 3 giai đoạn chính:

1. Session Initialization: Thiết lập phiên và nhận tham số từ server.
2. Key Exchange: Tạo khóa chung và dẫn xuất khóa mã hóa.
3. Secure Messaging: Mã hóa tin nhắn và gửi kèm chữ ký số để xác thực.

4.2.2 Sequence diagram

Quy trình trao đổi thông điệp giữa Client và Server được tóm tắt như sau:

1. Client $\xrightarrow{POST}$ Server (`/session/create`): Gửi thông số đường cong Eliptic (Curve P-256) và thuật toán yêu cầu (`ecdh_3`).
2. Server $\xrightarrow{200OK}$ Client: Trả về Session Token (JWT) và khóa công khai của Server (PK_{Server}).
3. Client (Internal):
   - Sinh cặp khóa ECDH tạm thời $(SK_{Client}, PK_{Client})$.
   - Tính toán shared secret: $S = \text{ECDH}(SK_{Client}, PK_{Server})$.
   - Dẫn xuất khóa AES: $K_{AES} = \text{PBKDF2}(S, \text{Salt}, \text{Iterations})$.
   - Sinh cặp khóa ký tạm thời $(SignSK, SignPK)$.
   - Ký lên PK_{Client}.
4. Client $\xrightarrow{POST}$ Server (`/session/exchange`): Gửi PK_{Client}, chữ ký (r, s) và $SignPK$.
5. Client $\xrightarrow{POST}$ Server (`/message/send`): Gửi tin nhắn mã hóa AES ($C = \text{Enc}(M)$) cùng với chữ ký số của gói tin đó.


Android Client
Server (API)
1. Session Initialization
POST /session/create
{ "algorithm": "ecdh_3",
"curveParameters": "P-256" }
200 OK
{ "sessionToken": "JWT...",
"serverPublicKey": {x, y},
"serverSignature": "..." }
2. Internal Processing
Verify Server Signature
Generate Ephemeral Keys (Client_SK, Client_PK)
Compute Shared Secret (ECDH)
Derive AES Key (PBKDF2, Salt=0x00)
Sign Client_PK
3. Key Exchange
POST /session/exchange
{ "clientPublicKey": {x, y},
"clientPublicKeySignature": "..." }
200 OK
{ "success": true,
"message": "Key exchange completed" }
4. Secure Messaging
Encrypt Message (AES-CBC + PKCS#7)
POST /message/send
{ "encryptedMessage": "Base64...",
"messageSignature": "..." }
200 OK
{ "encryptedResponse": "Base64...",
"responseSignature": "..." }
Decrypt Response

4.3 Pseudocode

4.3.1 Create session

Client gửi yêu cầu khởi tạo phiên với các tham số đường cong SECP256R1 (P-256) đã được hardcode.

Pseudocode:

```
FUNCTION Step1_CreateSession():
    CONSTANTS:
        ALGORITHM = "ecdh_3"
        CURVE_PARAMS = { p, a, b, Gx, Gy, order } // NIST P-256
           Parameters

    PAYLOAD = {
        "algorithm": ALGORITHM,
        "curveParameters": CURVE_PARAMS
    }

    RESPONSE = HTTP_POST("/session/create", PAYLOAD)

    IF RESPONSE.status == 200 AND RESPONSE.success == True:
        SESSION_TOKEN = RESPONSE.sessionToken
        SERVER_PUB_KEY = parsed(RESPONSE.serverPublicKey)
        RETURN SERVER_PUB_KEY
    ELSE:
        THROW Error("Session creation failed")
```

Listing 9: Mã giả cho bước tạo phiên

4.3.2 Exchange session

Client sinh khóa ECDH, tính toán shared secret với server, sau đó sử dụng hàm KDF (Key Derivation Function) để tạo ra khóa AES đối xứng. Một điểm đặc biệt trong cài đặt này là Client đã sử dụng PBKDF2 với Salt rỗng (16 bytes 0x00) để dẫn xuất khóa.

Pseudocode:

```
FUNCTION Step2_KeyExchange(SERVER_PUB_KEY):
    // 1. Generate Ephemeral ECDH Key Pair
    CLIENT_PRIV_KEY, CLIENT_PUB_KEY = EC_Generate_Key(SECP256R1)

    // 2. Compute Shared Secret (ECDH)
    SHARED_SECRET = ECDH_Exchange(CLIENT_PRIV_KEY, SERVER_PUB_KEY)

    // 3. Derive AES Key using PBKDF2
    // Observed Logic: Salt is 16 bytes of nulls
    SALT = 0x00 * 16
    AES_KEY = PBKDF2(
        Algorithm: SHA256,
        Length: 32 bytes (256 bits),
        Salt: SALT,
        Iterations: 1000,
        Input: SHARED_SECRET
    )

```

```
    // 4. Sign Client's Public Key for Integrity
    // Note: Implementation creates a fresh signing key for this
       request
    SIGN_PRIV, SIGN_PUB = EC_Generate_Key(SECP256R1)
    SIGNATURE = ECDSA_Sign(SIGN_PRIV, CLIENT_PUB_KEY)

    PAYLOAD = {
        "sessionToken": SESSION_TOKEN,
        "clientPublicKey": CLIENT_PUB_KEY,
        "clientPublicKeySignature": SIGNATURE,
        "clientSignaturePublicKey": SIGN_PUB
    }

    RESPONSE = HTTP_POST("/session/exchange", PAYLOAD)

    IF RESPONSE.clientSignatureVerified == True:
        STORE AES_KEY
        PRINT "Key Exchange Successful"
    ELSE:
        PRINT "Warning: Signature verification failed"
```

Listing 10: Mã giả cho bước trao đổi khóa

4.3.3 Encrypting and send message

Client sử dụng khóa AES_KEY đã tạo ở bước 2 để mã hóa tin nhắn theo chế độ AES-CBC với PKCS7 Padding. Vector khởi tạo (IV) được sinh ngẫu nhiên mỗi lần gửi và được ghép vào trước ciphertext.

Pseudocode:

```
FUNCTION Encrypt_Message_CBC(PLAINTEXT, AES_KEY):
    // 1. Generate IV
    IV = Random_Bytes(16)

    // 2. Pad Plaintext (PKCS#7 Block Size 128 bits)
    PADDED_TEXT = PKCS7_Pad(PLAINTEXT, BlockSize=16)

    // 3. Encrypt
    CIPHERTEXT = AES_Encrypt(Mode=CBC, Key=AES_KEY, IV=IV, Input=
       PADDED_TEXT)

    // 4. Encode
    RESULT = Base64_Encode(IV + CIPHERTEXT)
    RETURN RESULT

FUNCTION Send_Message(MESSAGE):
    ENCRYPTED_MSG = Encrypt_Message_CBC(MESSAGE, AES_KEY)

    // Generate ephemeral signature key for this message
    SIGN_PRIV, SIGN_PUB = EC_Generate_Key(SECP256R1)
    SIGNATURE = ECDSA_Sign(SIGN_PRIV, ENCRYPTED_MSG)

    PAYLOAD = {
        "sessionToken": SESSION_TOKEN,
```

```
        "encryptedMessage": ENCRYPTED_MSG,
        "messageSignature": SIGNATURE,
        "clientSignaturePublicKey": SIGN_PUB
    }

    HTTP_POST("/message/send", PAYLOAD)
```

Listing 11: Mã giả cho bước gửi tin nhắn

4.4 Re-implementation

Dựa trên thuật toán mã giả đã thiết kế ở phần 2.3, nhóm đã tiến hành hiện thực hóa mã nguồn Client hoàn chỉnh bằng ngôn ngữ Python. Source code sử dụng các thư viện chuẩn để đảm bảo tính chính xác nguyên thủy mật mã.

Môi trường và thư viện:

- Ngôn ngữ: Python 3.11
- Thư viện HTTP: `requests` (xử lý session, header và REST API)
- Thư viện mật mã: `cryptography` (cung cấp các module hazmat cho EC P-256, PBKDF2HMAC, AES-CBC và PKCS7 Padding).
- Cơ chế an toàn: Hàm `_safe_post` được cài đặt để tự động delay 1.1 giây giữa các request, nhằm đảm bảo server không chịu quá nhiều request cùng lúc.

Kết quả thực nghiệm: Bao gồm các bước sau:

1. Kết nối đến Server (`/session/create`).
2. Thực hiện exchange key (`/session/exchange`).
3. Dẫn xuất AES key.

Log thực tế khi chạy script python:

```
[*] Step 1: Creating session for group-3...
    [Wait] Sleeping 1.1s to respect rate limit...
    [+] Session Token: eyJhbGciOiJIUzI1NiIs...
[*] Step 2: Performing Key Exchange...
    [+] AES Key Derived successfully
    [Wait] Sleeping 1.1s to respect rate limit...
    [+] Key Exchange Complete.
```

Listing 12: Log kết quả chạy script Python

Phân tích kết quả xuất hiện:

- **Session Creation:** Server phản hồi mã 200 OK và cấp phát Session Token (JWT).
- **AES Key Derivation:** Client đã tính toán thành công shared secret từ public key của Server và dẫn xuất ra khóa AES 256-bit.
- **Key Exchange Complete:** Thông báo này xác nhận rằng Server đã kiểm tra chữ ký số (`clientPublicKeySignature`) mà Client gửi lên và chấp nhận thiết lập kênh bảo mật. Đây là bằng chứng cho thấy source code đang hoạt động ổn định và hoàn toàn tương thích với Server mục tiêu.

4.5 SMC Exploitation

4.5.1 Threat Model

Trong kịch bản khai thác này, giả định rằng là mô hình kẻ tấn công với các đặc điểm sau:

1. **Vị trí:** Kẻ tấn công là một thực thể đứng giữa (MITM) trên đường truyền mạng giữa thiết bị Android và Server. Kẻ tấn công có thể nằm trên cùng mạng Wi-Fi cục bộ hoặc kiểm soát gateway
2. **Khả năng:**
   - Giám sát lưu lượng: Kẻ tấn công có thể chặn bắt và quan sát toàn bộ các gói tin HTTP/HTTPS được trao đổi
   - Phân tích gói tin: Kẻ tấn công nhìn thấy được cấu trúc JSON của các request, đặc biệt là trường `encryptedMessage` được mã hóa Base64.
3. **Giới hạn:**
   - Kẻ tấn công không sở hữu các khóa bí mật hoặc khóa phiên
   - Kẻ tấn công không thể giải mã trực tiếp bản mã để đọc nội dung tin nhắn
4. **Mục tiêu:** Xác định chính xác (hoặc ước lượng độ dài khoảng) độ dài của tin nhắn gốc chỉ dựa trên kích thước của dữ liệu mã hóa bị rò rỉ, mà không cần giải mã nội dung.

4.5.2 Vulnerability Theory

1. **Padding in AES-CBC**

   Giao thức SMC sử dụng thuật toán mã hóa AES với chế độ CBC. Do AES là block cipher hoạt động trên các khối dữ liệu 128-bit, tin nhắn có độ dài bất kỳ cần phải được padding để chia hết cho kích thước khối trước khi mã hóa.

   SMC sử dụng chuẩn PKCS#7 Padding. Quy tắc của PKCS#7 là thêm N bytes có giá trị N vào cuối tin nhắn sao cho tổng độ dài chia hết cho 16. Số lượng byte đệm luôn nằm trong khoảng từ 1 tới 16 ($1 \leq \text{Padding} \leq 16$).

   Công thức tính độ dài bản mã (L_{cipher}) dựa trên độ dài tin nhắn gốc (L_{plain}):

   $$L_{cipher} = L_{IV} + \text{RoundUp}_{16}(L_{plain} + 1)$$

   Trong đó:

   - L_{IV} là độ dài Vector khởi tạo (luôn là 16 byte).
   - Hàm RoundUp_{16} làm tròn lên bội số gần nhất của 16.

2. **Information Leakage**

   Vấn đề nằm ở chỗ kích thước của bản mã (L_{cipher}) tiết lộ trực tiếp khoảng độ dài tin nhắn gốc. Mặc dù kẻ tấn công không thể giải mã được nội dung tin nhắn, nhưng họ có thể phân biệt được độ dài của tin nhắn. Ví dụ như có thể phân biệt được một câu trả lời ngắn với tin nhắn có chứa mật khẩu dài hơn.

   Đây là một side-channel vi phạm Perfect Secrecy, cho phép kẻ tấn công suy ra tính chất của dữ liệu được truyền tải.

4.5.3 Attack Methodology

Mô hình đe dọa: Eavesdropper nằm giữa Client và Server (MITM), có khả năng giữ các gói tin HTTP nhưng không có khóa để giải mã.

Quy trình tấn công:

1. **Sniffing:** Kẻ tấn công sử dụng công cụ để chặn gói tin JSON gửi đến endpoint `/message/send`.
2. **Extraction:** Trích xuất trường `encryptedMessage` (chuỗi Base64).
3. **Analysis:**
   - Giải mã Base64 để lấy chuỗi byte thô.
   - Tính độ dài chuỗi byte (L_{total}).
   - Trừ đi 16 bytes đầu tiên (IV) để lấy độ dài phần tải ($L_{payload}$).
   - Áp dụng công thức ngược của PKCS#7 để xác định khoảng độ dài tin nhắn gốc (L_{real}):

$$L_{payload} - 16 \leq L_{real} \leq L_{payload} - 1$$

4.5.4 PoC Implementation

Nhóm đã xây dựng một module phân tích trong source code được implement sau đây. Hàm `analyze_cipher_length` đóng vai trò là công cụ khai thác, tự động tính toán khoảng độ dài tin nhắn ngay khi bắt được gói tin mã hóa.

```
def analyze_cipher_length(b64_cipher_from_json):
    """
    Analyzes the 'encryptedMessage' field from the JSON to determine
       plaintext length.
    """
    # Attacker sees the Base64 string length in the JSON
    base64_len = len(b64_cipher_from_json)

    # Attacker decodes to get raw bytes
    cipher_bytes = base64.b64decode(b64_cipher_from_json)
    raw_len = len(cipher_bytes)

    # Attacker calculates length based on AES-CBC PKCS7
    # Structure: IV (16 bytes) + Encrypted Blocks
    iv_len = 16
    payload_len = raw_len - iv_len

    # Logic: PKCS7 always adds padding (1 to 16 bytes).
    # Real_Length = Payload_Length - Padding
    # Since Padding is 1..16, Real_Length is between (Payload - 16)
       and (Payload - 1)

    min_real_len = payload_len - 16
    max_real_len = payload_len - 1

    if min_real_len < 0: min_real_len = 0

    return base64_len, raw_len, (min_real_len, max_real_len)
```

Listing 13: Hàm tính toán độ dài tin nhắn

4.6 Experimental Results

Để kiểm chứng rằng suy đoán của nhóm đang đi đúng hướng thì nhóm đã thực hiện gửi các tin nhắn có độ dài khác nhau từ Client. Công cụ tấn công đã bắt được gói tin và đưa ra dự đoán chính xác.

4.6.1 Kịch bản 1

Tin nhắn ngắn ("a")

- **Input:** Tin nhắn "a"(1 byte)
- **Ciphertext:** `TLfnpJ2hF0...`
- **Phân tích:**
  - Raw Length: 32 bytes
  - Payload (trừ IV): 16 bytes
  - Khoảng dự đoán: 0 tới 15 bytes
- **Kết quả:** Chính xác vì $1 \in [1; 15]$.

```
--- [Scenario] User sends: 'a' ---
[*] Sending message: 'a'
    [Wait] Sleeping 1.1s to respect rate limit...
    [ATTACKER VIEW]
    Intercepted JSON Value: 'TLfnpJ2hF0U9UmI...'
    Base64 String Length:   44 chars
    Decoded Raw Length:     32 bytes
    >> DEDUCTION: Real message is between 0 and 15 bytes.
    [SUCCESS] Actual length 1 falls within attacker's estimated range!
```

Hình 11: *Kết quả chạy thực nghiệm Kịch bản 1*

4.6.2 Kịch bản 2

Tin nhắn dài ("This is a much longer message to force a new block")

- **Input:** Tin nhắn dài 50 bytes
- **Ciphertext:** `l4rP7MAJ9Y9KIPv...`
- **Phân tích:**
  - Raw length: 80 bytes
  - Payload: 64 bytes
  - Khoảng dự đoán: [48; 63] bytes
- **Kết quả:** Chính xác vì $50 \in [48; 63]$.

```
--- [Scenario] User sends: 'This is a much longer message to force a new block' ---
[*] Sending message: 'This is a much longer message to force a new block'
    [Wait] Sleeping 1.1s to respect rate limit...
    [ATTACKER VIEW]
    Intercepted JSON Value: 'l4rP7MAJ9Y9KIPv...'
    Base64 String Length:   108 chars
    Decoded Raw Length:     80 bytes
    >> DEDUCTION: Real message is between 48 and 63 bytes.
    [SUCCESS] Actual length 50 falls within attacker's estimated range!
```

Hình 12: *Kết quả chạy thực nghiệm Kịch bản 2*

4.7 Conclusion & Improvement

4.7.1 Conclusion

Qua quá trình phân tích và thực nghiệm, nhóm đã thực hiện được các mục tiêu đề ra cho đề tài này.

1. **Protocol Reconstruction:** Đã phân tích thành công luồng giao tiếp API, cơ chế trao đổi khóa ECDH và dẫn xuất khóa AES-CBC thông qua việc bắt gói tin và đảo ngược mã nguồn.
2. **Exploitation:** Đã chứng minh được rằng việc sử dụng AES-CBC với PKCS#7 padding tiêu chuẩn mà không có cơ chế che giấu độ dài dẫn đến việc rò rỉ thông tin kích thước tin nhắn.
3. **Result:** Script tấn công của nhóm có khả năng dự đoán chính xác khoảng độ dài của tin nhắn gốc (L_{real}) dựa trên độ dài của bản mã (L_{cipher}) thu thập được từ kênh truyền mạng, xác nhận rằng tính bí mật của metadata của người dùng không được đảm bảo

4.7.2 Improvement

Lỗ hổng này không nằm ở thuật toán mã hóa mà nằm ở cách cài đặt giao thức. Để khắc phục triệt để vấn đề rò rỉ độ dài, có một số các đề xuất giải pháp kỹ thuật sau:

1. **Sử dụng Content-Length Padding:** Thay vì chỉ đệm dữ liệu để chia hết cho block size của AES, giao thức nên quy định một kích thước cố định cho mọi gói tin (ví dụ 256 bytes)
    - **Cơ chế:** Nếu tin nhắn ngắn gọn hơn kích thước quy định, hệ thống sẽ chèn thêm các dummy bytes cho đến khi đạt được kích thước chuẩn trước khi mã hóa.
    - **Hiệu quả:** Kẻ tấn công sẽ thấy mọi gói tin mã hóa đều có kích thước y hệt nhau, bất kể nội dung bên trong là "Hello"hay là một đoạn văn dài

```
1         BLOCK_SIZE = 16
2         TARGET_LENGTH = 512 # Fixed length to be 512 bytes
3
4         def pad_to_fixed_size(plaintext):
5             # 1. PKCS#7 Padding
6             padded_text = pkcs7_pad(plaintext, BLOCK_SIZE)
7
8             # 2.Add dummy bytes to reach TARGET_LENGTH
9             current_len = len(padded_text)
```

```
            if current_len < TARGET_LENGTH:
                dummy_data = generate_random_bytes(TARGET_LENGTH
                    - current_len)
                return padded_text + dummy_data
            return padded_text
```

2. **Random Padding Strategy** Nếu việc cố định kích thước gây lãng phí băng thông quá lớn, có thể sử dụng phương pháp đệm ngẫu nhiên một lượng byte (random amount of bytes) vào mỗi tin nhắn.

   - **Cơ chế:** Thêm một trường `padding` chứa dữ liệu rác vào cấu trúc JSON hoặc plaintext trước khi mã hóa
   - **Hiệu quả:** Làm nhiễu mối tương quan tuyến tính giữa L_{cipher} và L_{plain}. Cùng một tin nhắn "Hello"khi gửi 2 lần sẽ có độ dài bản mã hoàn toàn khác nhau.

3. **Encryption Mode:** Cân nhắc chuyển sang sử dụng các chế độ mã hóa hiện đại hỗ trợ tốt hơn việc che giấu luồng dữ liệu hoặc sử dụng các giao thức như TLS Record Padding để hỗ trợ việc che giấu độ dài traffic một cách tự nhiên hơn, thay vì tự triển khai cơ chế padding ở tầng ứng dụng.

Tài liệu

[1] Google Developers. *Start the Emulator from the command line*. Android Studio User Guide. https://developer.android.com/studio/run/emulator-commandline

[2] Google Developers. *Android Debug Bridge (adb) commands*. Android Studio User Guide. https://developer.android.com/tools/adb

[3] Frida. *Frida Documentation: Installation and Usage*. https://frida.re/docs/home/

[4] P. Cipolloni. *Universal Android SSL Pinning Bypass with Frida*. Frida CodeShare. https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/