

BÁO CÁO THỰC HÀNH

Thực hành An toàn mạng máy tính

Lab 6: Review of Encryption Algorithms

GVTH: Nguyễn Ngọc Trưởng

Ngày báo cáo: 08/12/2025

Nhóm 6 – NT101.Q11.1

THÔNG TIN CHUNG

| MSSV | Họ và tên | Nội dung báo cáo | % công việc |
|----------|-------------|-------------------|-------------|
| 23520007 | Lê Chánh Ân | Câu 1, 2, 3, 4, 5 | 100% |

Mục lục:

| | |
|---|----|
| 1. Caesar cipher. | 3 |
| a) Ý tưởng và ngôn ngữ được sử dụng. | 3 |
| b) Chức năng chính của chương trình. | 3 |
| c) Giải thuật. | 3 |
| d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử. | 5 |
| 2. Mono-alphabetic substitution cipher. | 7 |
| a) Ý tưởng và ngôn ngữ được sử dụng. | 7 |
| b) Chức năng chính của chương trình. | 7 |
| c) Giải thuật. | 7 |
| d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử. | 10 |
| 3. Vigenère cipher. | 11 |
| a) Ý tưởng và ngôn ngữ được sử dụng. | 11 |
| b) Chức năng chính của chương trình. | 12 |
| c) Giải thuật. | 12 |
| d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử. | 14 |
| 4. DES — Cài đặt các chế độ hoạt động. | 16 |
| a) Ý tưởng và ngôn ngữ được sử dụng. | 16 |
| b) Chức năng chính của chương trình. | 17 |
| c) Giải thuật. | 17 |
| d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử. | 22 |
| 5. AES — Cài đặt các chế độ hoạt động. | 24 |
| a) Ý tưởng và ngôn ngữ được sử dụng. | 24 |
| b) Chức năng chính của chương trình. | 25 |
| c) Giải thuật. | 25 |
| d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử. | 31 |
| 6. Tài liệu hướng dẫn. | 33 |

1. Caesar cipher.

a) Ý tưởng và ngôn ngữ được sử dụng.

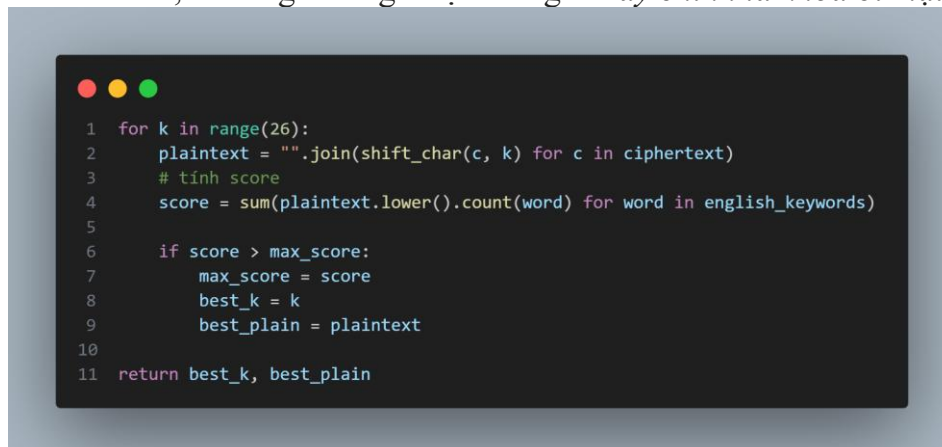
- Ý tưởng:
 - o Mục tiêu xây dựng một "Công cụ phá mã Caesar tự động trên Web"
 - o Nó sẽ tìm một plaintext duy nhất bằng cách thử tất cả các khóa khả dĩ.
- Ngôn ngữ được sử dụng:
 - o Python (Core): Ngôn ngữ lập trình chính xử lý logic tính toán, chuỗi và vòng lặp.
 - o Html, css: xử lý giao diện và màu sắc.
 - o Flask (Web Framework): Giúp biến python từ một code logic thành 1 trang web dễ sử dụng.

b) Chức năng chính của chương trình.

- Chức năng chính của chương trình này là Tự động phá mã Caesar (Caesar Cipher Auto-Cracker) trên giao diện Web.
- Caesar Cipher là một phương pháp mã hoá cổ điển, trong đó mỗi ký tự trong plaintext được dịch chuyển (shift) một số vị trí trong bảng chữ cái.
- Nó phá mã mà không cần khóa, tự thử tất cả 26 key để giải mã.
- Ngoài ra, em còn sử dụng thêm html, css để tạo giao diện, dễ dàng sử dụng và trực quan hơn.

c) Giải thuật.

- Giải thuật được sử dụng trong chương trình này là sự kết hợp giữa kỹ thuật **Brute-force** và **Phân tích Heuristic (Dựa trên từ khóa)**.
- Dưới đây là lưu đồ và phân tích chi tiết từng bước của thuật toán:
- **Tên giải thuật: Brute-force với Hàm Đánh giá (Scoring Function)**
- Quy trình xử lý đi qua 4 bước chính:
- **Bước 1: Vòng lặp Loop**
 - o Vì mã Caesar chỉ có 26 cách dịch chuyển (tương ứng 26 chữ cái trong bảng chữ cái tiếng Anh), thuật toán sẽ thử từng key.
 - o Khởi tạo một vòng lặp k chạy từ 0 đến 25.
 - o Với mỗi k, chương trình giả định rằng: "*Đây chính là khóa bí mật*".




Hình 1. Vòng lặp.

- Bước 2: Giải mã thử nghiệm (The Decryption Logic)

- Với mỗi giá trị k trong vòng lặp, chương trình thực hiện phép dịch ngược (giải mã) từng ký tự c trong đoạn văn bản ciphertext.
- Công thức toán học được sử dụng trong hàm `shift_char`:

$$P = (C - k) \bmod 26$$
- Trong đó:
 - P: Ký tự rõ (Plaintext).
 - C: Ký tự mã hóa (Ciphertext).
 - k: Số bước dịch chuyển đang thử.
 - mod 26: Phép chia lấy dư để đảm bảo ký tự quay vòng (ví dụ: 'A' trừ 1 sẽ quay lại 'Z').
- Trong code Python: `chr((ord(c) - ord('A') - k) % 26 + ord('A'))` -> Code này chuyển ký tự sang số ASCII, đưa về hệ 0-25, trừ k, lấy phần dư, rồi chuyển ngược lại thành ký tự.



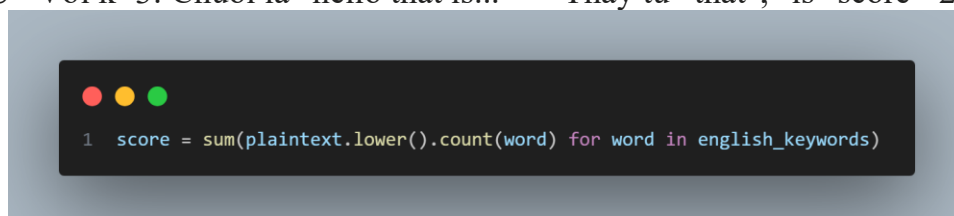
```

1 def shift_char(c, k):
2     if 'A' <= c <= 'Z':
3         return chr((ord(c) - ord('A') - k) % 26 + ord('A'))
4     elif 'a' <= c <= 'z':
5         return chr((ord(c) - ord('a') - k) % 26 + ord('a'))
6     else:
7         return c
  
```

Hình 2. Logic.

- Bước 3: Score

- Đây là phần giúp máy tính nhận biết đâu là kết quả đúng.
- Sau khi giải mã thử với khóa k, ta được một chuỗi plaintext tạm thời. Thuật toán sẽ chấm điểm chuỗi này:
- Danh sách từ khóa: Chuẩn bị sẵn một danh sách các từ tiếng Anh phổ biến: ["the", "and", "that", "is", "you", ...].
- Score: Đếm xem trong chuỗi plaintext tạm thời đó xuất hiện bao nhiêu từ nằm trong danh sách trên.
- Ví dụ:
 - Với k=1: Chuỗi là "ifmmp..." -> Không thấy từ nào có nghĩa score = 0.
 - Với k=3: Chuỗi là "hello that is..." -> Thấy từ "that", "is" score= 2.



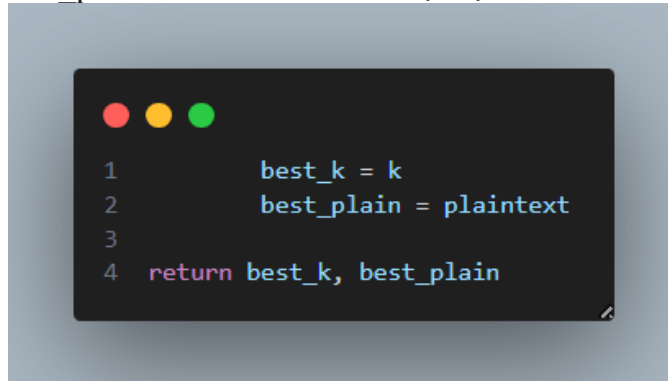
```

1 score = sumplaintext.lower().count(word) for word in english_keywords
  
```

Hình 3. Score.

- Bước 4: So sánh và Chọn lọc (Selection)

- Thuật toán duy trì một biến `max_score` (điểm cao nhất) và `best_plain` (kết quả tốt nhất).
- Nếu `score` hiện tại $>$ `max_score`:
- Cập nhật `max_score` = Điểm hiện tại.
- Lưu lại `best_k` = `k` hiện tại.
- Lưu lại `best_plain` = Chuỗi văn bản hiện tại.



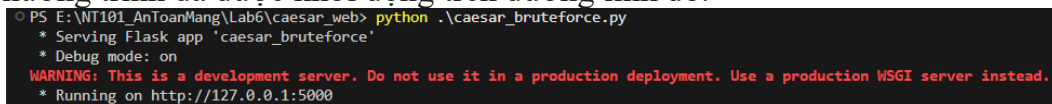
Hình 4. Selection.

- Sau khi chạy hết 26 vòng lặp, kết quả được lưu trong `best_plain` chính là kết quả có khả năng đúng cao nhất.

d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử.

Khởi động Server:

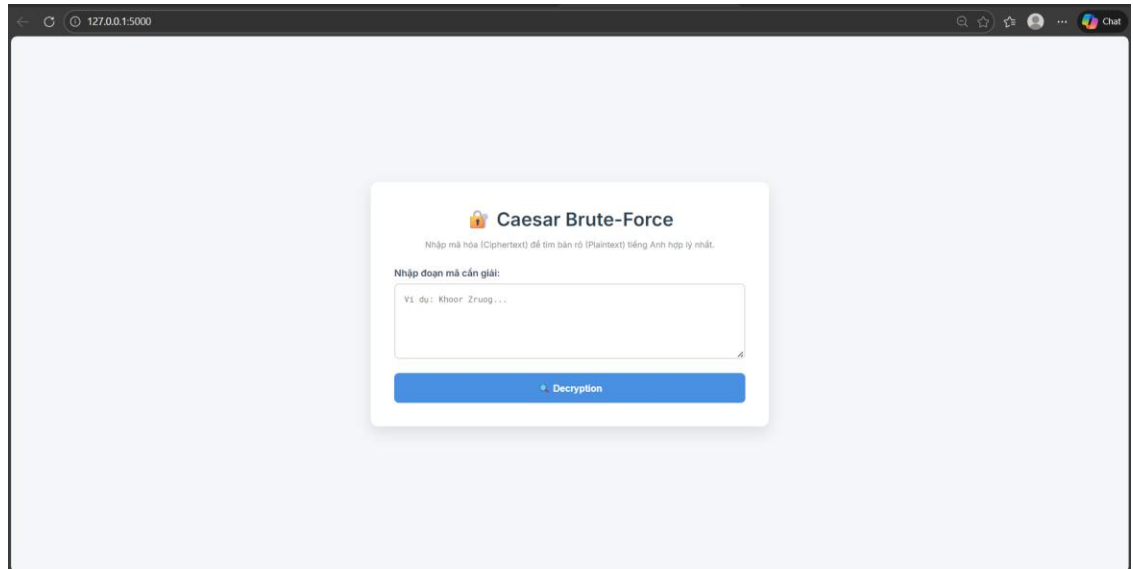
- Biên dịch và chạy: Sử dụng lệnh `python <tên file>` để chạy file python và Flask sẽ khởi tạo một Web Server cục bộ (Localhost).
- Trên terminal sẽ hiển thị thông báo Running on <http://127.0.0.1:5000>. Thì lúc này chương trình đã được khởi động trên đường link đó.



Hình 5. Url.

Truy cập Web:

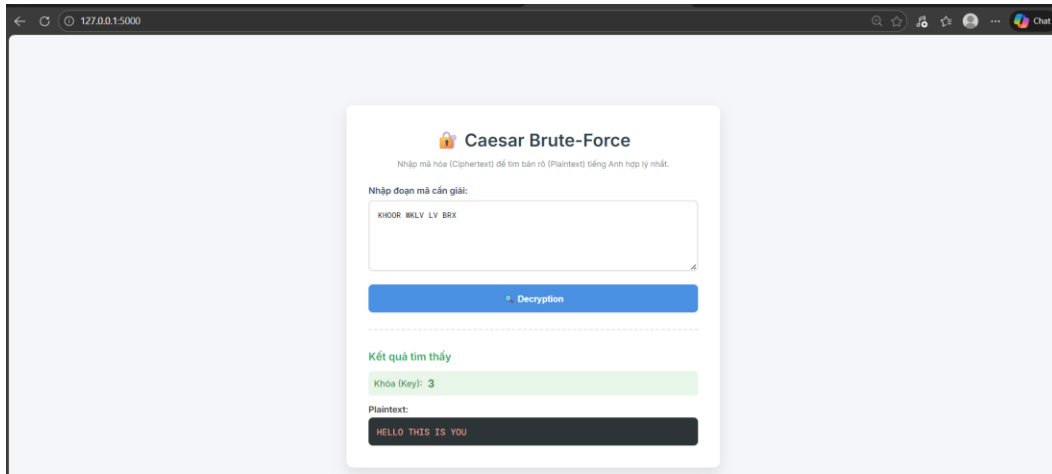
- Mở trình duyệt và truy cập <http://127.0.0.1:5000>
- Trình duyệt sẽ gửi đến server yêu cầu để truy cập web.
- Server phản hồi: Hàm `index()` trong code được gọi và server lấy file `index.html`, điền các giá trị rỗng (vì chưa có kết quả) và gửi về trình duyệt.
- Cuối cùng ta sẽ hiển thị được giao diện web như hình dưới đây.



Hình 6. Giao diện Caesar.

Decryption:

- Đây là giai đoạn quan trọng nhất, diễn ra khi bạn nhập đoạn mã (ví dụ: KHOOR WKLV LV BRX) và nhấn nút:
- 1. Gửi dữ liệu: Trình duyệt đóng gói chữ “KHOOR WKLV LV BRX” và gửi một yêu cầu POST về Server.
- 2. Tiếp nhận:
 - o Hàm index() lại được gọi.
 - o Lần này request.method là POST.
 - o Code lấy dữ liệu từ ô nhập: `input_cipher = " KHOOR WKLV LV BRX"`.
- 3. Kích hoạt Logic thuật toán:
 - o Server gọi hàm `caesar_bruteforce("KHOOR WKLV LV BRX")`.
 - o Vòng lặp chạy 26 lần (thử $k=0$ đến $k=25$).
 - o Máy tính toán, so sánh với danh sách từ khóa (the, and...) và tìm ra $k=3$ có điểm cao nhất (ra chữ “HELLO THIS IS YOU”).
- 4. Trả kết quả:
 - o Hàm trả về $k=3$ và `plaintext="HELLO"`.
 - o Server dùng Jinja2 để chèn số 3 và chữ “HELLO THIS IS YOU” vào đúng vị trí trong file html.
- 5. Hiển thị kết quả: Trình duyệt tải lại trang, nhưng lần này bên dưới ô nhập liệu sẽ hiện ra dòng chữ:
- Key tìm được: 3 Nội dung: HELLO THIS IS YOU



Hình 7. Output.

2. Mono-alphabetic substitution cipher.

a) Ý tưởng và ngôn ngữ được sử dụng.

- Ý tưởng:
 - o Bài toán này giải quyết việc bẻ khóa Mật mã thay thế đơn bảng (Monoalphabetic Substitution Cipher).
 - o Sử dụng thuật toán Hill Climbing kết hợp với thống kê N-gram.
- Ngôn ngữ được sử dụng:
 - o Python (Core): Ngôn ngữ lập trình chính xử lý logic tính toán, chuỗi và vòng lặp.
 - o Html, css: xử lý giao diện và màu sắc.
 - o Flask (Web Framework): Giúp biến python từ một code logic thành 1 trang web dễ sử dụng.

b) Chức năng chính của chương trình.

- Chức năng chính của chương trình là dùng phương pháp thống kê để biến một đoạn mã vô nghĩa thành một đoạn văn bản tiếng Anh có nghĩa.
- Chương trình này không chỉ giải mã mà còn tìm ra bảng quy tắc mã hóa.
- Sử dụng Hill Climbing liên tục thử sai và sửa sai. Tráo đổi vị trí các chữ trong khóa hàng nghìn lần và cho ra đoạn text có nghĩa tiếng Anh hơn và loại bỏ những lần tệ đi.
- Ngoài ra, em còn sử dụng thêm html, css để tạo giao diện, dễ dàng sử dụng và trực quan hơn.

c) Giải thuật.

- Giải thuật chính được em sử dụng trong đoạn code này là **Hill Climbing** kết hợp với **Phân tích thống kê Quadgram**.
- Đây là một thuật toán thuộc nhóm **Local Search**. Thay vì thử tất cả các trường hợp (quá lâu), nó bắt đầu từ một giải pháp "tạm ổn" và liên tục sửa đổi nhỏ để leo lên vị trí có kết quả tốt hơn.
- Bước 1: Nạp kiến thức về tần suất chuẩn trong tiếng Anh.

- Nó thực hiện việc này bằng cách nạp bảng xác suất của các cụm 4 chữ cái (Quadgrams) và chuyển sang dạng Logarithm để tính toán nhanh hơn.
- Logic của nó sẽ là đọc file -> tính xác suất số lần xuất hiện -> lưu dưới dạng $\log_{10}(F)$ để biến phép nhân xác suất thành phép cộng (tránh lỗi số quá nhỏ)

```

1 def _load_quadgram_stats(self, filename):
2     quad_counts = {}
3     total_count = 0
4     try:
5         if not os.path.exists(filename):
6             raise FileNotFoundError(f"File not found at: {filename}")
7
8         with open(filename, 'r', encoding='utf-8') as f:
9             for line in f:
10                parts = line.split()
11                if len(parts) == 2:
12                    quad = parts[0].lower()
13                    count = int(parts[1])
14                    quad_counts[quad] = count
15                    total_count += count
16
17                for quad, count in quad_counts.items():
18                    self.quadgram_logs[quad] = math.log10(float(count) / total_count)
19
20                if quad_counts:
21                    min_prob = math.log10(0.01 / total_count)
22                    self.floor = min_prob

```

Hình 8. Quadgram.

- Bước 2: Khởi tạo.

- Thay vì bắt đầu dò tìm từ con số 0, thuật toán sẽ bắt đầu bằng cách so sánh tần suất ký tự đơn.

```

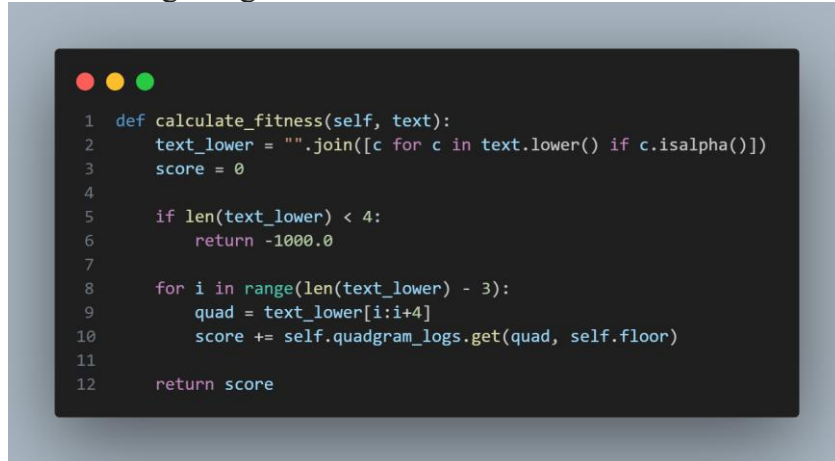
1 def create_initial_key(self, ciphertext):
2     char_freq = self.analyze_text(ciphertext)
3     sorted_cipher = sorted(char_freq.items(), key=lambda x: x[1], reverse=True)
4     sorted_english = sorted(self.english_freq.items(), key=lambda x: x[1], reverse=True)
5
6     key = {}
7     for i, (cipher_char, _) in enumerate(sorted_cipher):
8         if i < len(sorted_english):
9             key[cipher_char] = sorted_english[i][0]
10
11     all_letters = set('abcdefghijklmnopqrstuvwxyz')
12     used = set(key.values())
13     unused = list(all_letters - used)
14
15     for char in all_letters:
16         if char not in key:
17             key[char] = unused.pop(0) if unused else char
18     return key

```

Hình 9. Initial.

- Đầu tiên nó sẽ phân tích tần suất mật mã và lấy tần suất trong bảng tần suất chuẩn của tiếng Anh bằng lệnh sorted.

- Sau đó sẽ ghép cặp các ký tự của mật mã với của bảng theo tần suất bằng vòng lặp for.
- Bước 3: Hàm điểm.
 - Hàm này giúp ta có thể biết được đoạn văn bản được giải mã gần giống với đoạn văn bản tiếng Anh chuẩn nhất.



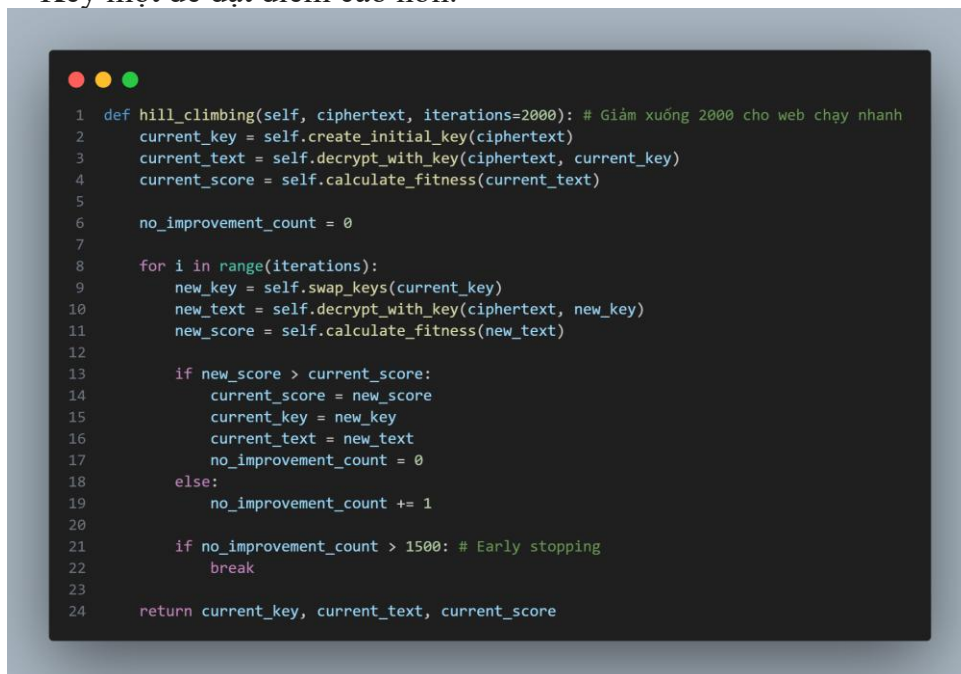
```

1 def calculate_fitness(self, text):
2     text_lower = "".join([c for c in text.lower() if c.isalpha()])
3     score = 0
4
5     if len(text_lower) < 4:
6         return -1000.0
7
8     for i in range(len(text_lower) - 3):
9         quad = text_lower[i:i+4]
10        score += self.quadgram_logs.get(quad, self.floor)
11
12    return score

```

Hình 10. Calculate_fitness.

- Đầu tiên là nó sẽ làm sạch text và mặc định score bằng 0.
- Cửa sổ trượt lấy 4 ký tự để chia ra thành từng cụm.
- Tra bảng điểm quadgram_logs đã nạp ở Bước 1 và cộng dồn điểm lại. Điểm càng lớn (gần 0), văn bản càng giống tiếng Anh.
- Bước 4: Tối ưu hóa với giải thuật Hill Climbing.
 - Đây được xem như là phần quan trọng nhất, giải thuật này cố gắng sửa từng Key một để đạt điểm cao hơn.



```

1 def hill_climbing(self, ciphertext, iterations=2000): # Giảm xuống 2000 cho web chạy nhanh
2     current_key = self.create_initial_key(ciphertext)
3     current_text = self.decrypt_with_key(ciphertext, current_key)
4     current_score = self.calculate_fitness(current_text)
5
6     no_improvement_count = 0
7
8     for i in range(iterations):
9         new_key = self.swap_keys(current_key)
10        new_text = self.decrypt_with_key(ciphertext, new_key)
11        new_score = self.calculate_fitness(new_text)
12
13        if new_score > current_score:
14            current_score = new_score
15            current_key = new_key
16            current_text = new_text
17            no_improvement_count = 0
18        else:
19            no_improvement_count += 1
20
21        if no_improvement_count > 1500: # Early stopping
22            break
23
24    return current_key, current_text, current_score

```

Hình 11. Hill Climbing.

- Tạo ra new_key bằng cách trao đổi vị trí 2 ký tự ngẫu nhiên trong Key hiện tại (swap_keys).
- self.calculate_fitness(new_text) để giải mã key mới và tính điểm.

- Nếu Điểm mới > điểm hiện tại thì sẽ giữ Key mới. Ngược lại thì giữ Key hiện tại.
- Hàm sẽ break nếu sau 1500 lần thử mà điểm không tăng. Mặc định điểm đó sẽ là đỉnh.

d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử.

Khởi động Server:

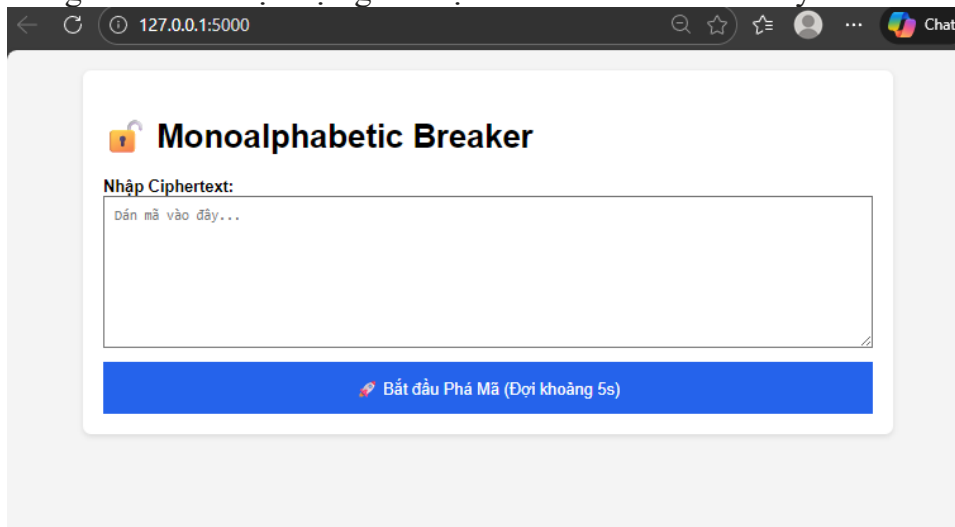
- Biên dịch và chạy: Sử dụng lệnh python <tên file> để chạy file python và Flask sẽ khởi tạo một Web Server cục bộ (Localhost).
- Trên terminal sẽ hiển thị thông báo Running on <http://127.0.0.1:5000>. Thì lúc này chương trình đã được khởi động trên đường link đó.

```
PS E:\NT101_AnToanMang\Lab6\mono_web_app> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Hình 12. Url.

Truy cập Web:

- Mở trình duyệt và truy cập <http://127.0.0.1:5000>
- Trình duyệt sẽ gửi đến server yêu cầu để truy cập web.
- Server phản hồi: Hàm index() trong code được gọi và server lấy file index.html, điền các giá trị rỗng (vì chưa có kết quả) và gửi về trình duyệt.
- Cuối cùng ta sẽ hiển thị được giao diện web như hình dưới đây.

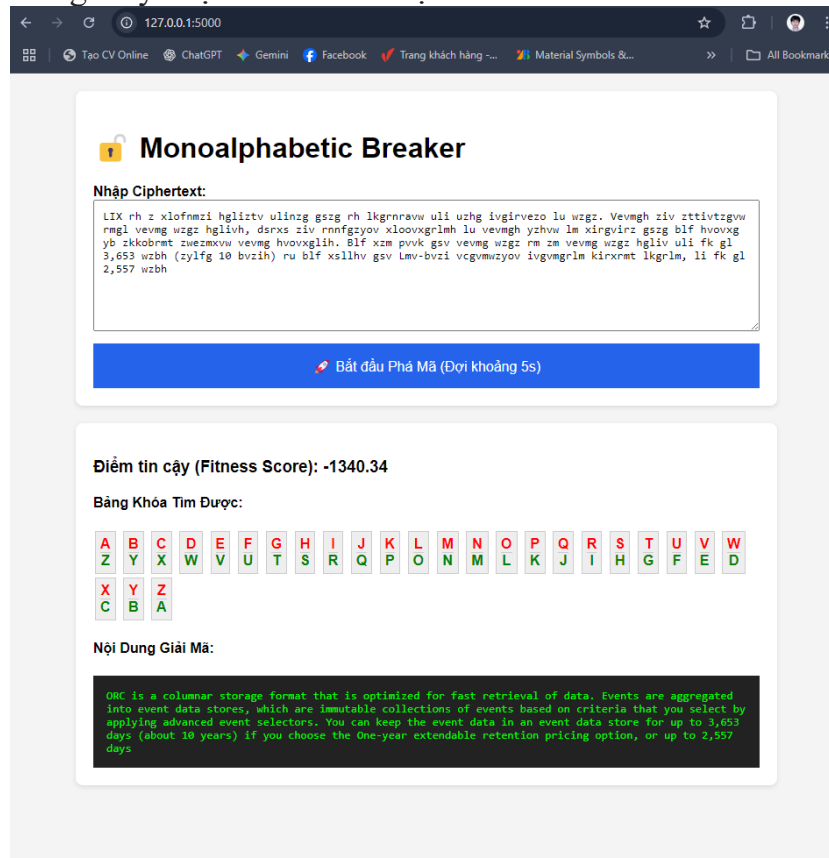


Hình 13. Giao diện Monoalphabetic

Decryption:

- Nó hoạt động như một cỗ máy dịch thuật đơn giản: nhận vào văn bản mã hóa và một cuốn từ điển (Key), sau đó dịch từng ký tự một.
1. Gửi dữ liệu:
 - Trình duyệt đóng gói chuỗi ciphertext và gửi một yêu cầu (thường là POST) về Server (Flask).
 2. Tiếp nhận:
 - Server nhận yêu cầu và gọi hàm wrapper solve_cipher_web(ciphertext).

- Code khởi tạo class MonoalphabeticCipherBreaker, nạp dữ liệu thống kê từ file english_quadgrams.txt.
3. Kích hoạt Logic Thuật toán (Trong vòng lặp Hill Climbing):
- Server bắt đầu quá trình Hill Climbing. Tại mỗi bước lặp, nó gọi hàm decrypt_with_key(ciphertext, current_key).
 - Dịch và ghép từng kí tự theo logic code và trả về kết quả.
4. Đánh giá & Lưu trữ:
- Hàm calculate_fitness chấm điểm chuỗi plaintext. Nếu điểm cao, thuật toán giữ lại Key này.
 - Sau khi chạy xong 2000 vòng lặp, Server tìm ra Key tốt nhất và Plaintext cuối cùng.
 - Ghi file: Server tự động tạo/ghi đè file plaintext.txt với nội dung Key và kết quả vừa tìm được.
5. Trả kết quả:
- Trình duyệt nhận dữ liệu JSON này và hiển thị lên màn hình giao diện web cho người dùng thấy đoạn văn bản đã dịch.



Hình 14. Output.

3. Vigenère cipher.

a) Ý tưởng và ngôn ngữ được sử dụng.

- Ý tưởng:
 - Mấu chốt của Vigenère là nó dùng nhiều bảng mã Caesar khác nhau xen kẽ. Vì vậy, ý tưởng giải mã chia làm 2 giai đoạn lớn:

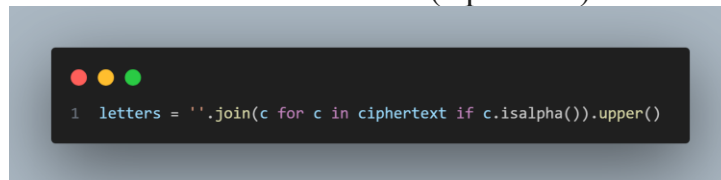
- Biến bài toán Vigenère phức tạp thành nhiều bài toán Caesar đơn giản (bằng cách tìm độ dài khóa).
- Giải từng bài toán Caesar đó bằng thống kê tần suất.
- Sử dụng phương pháp Index of Coincidence (IC) để ước lượng độ dài khóa.
- Ngôn ngữ được sử dụng:
 - Python (Core): Ngôn ngữ lập trình chính xử lý logic tính toán, chuỗi và vòng lặp.
 - Html, css: xử lý giao diện và màu sắc.
 - Flask (Web Framework): Giúp biến python từ một code logic thành 1 trang web dễ sử dụng.

b) Chức năng chính của chương trình.

- Thay vì bắt người dùng phải đoán xem mật khẩu dài bao nhiêu ký tự (ví dụ: mật khẩu 3 chữ hay 5 chữ), chương trình tự chạy thuật toán tìm kiếm.
- Chức năng chính của chương trình là dùng phương pháp phân tích thống kê để biến một đoạn mã Vigenère vô nghĩa thành một đoạn văn bản tiếng Anh có nghĩa.
- Chương trình này không chỉ giải mã nội dung mà còn tự động truy vết và tìm ra chính xác **từ khóa (Key)** gốc đã dùng để mã hóa.
- Sử dụng **Index of Coincidence** để xác định độ dài khóa, kết hợp với kiểm định **Chi-square** để so sánh tần suất. Chương trình chia nhỏ văn bản thành các nhóm và liên tục so sánh với bảng tần suất tiếng Anh chuẩn để chọn ra ký tự dịch chuyển tối ưu nhất.
- Ngoài ra, em còn sử dụng thêm HTML, CSS để tạo giao diện Web, giúp người dùng nhập liệu dễ dàng và xem kết quả trực quan hơn.

c) Giải thuật.

- Đây là chương trình sử dụng Index of Coincidence (IC) để ước lượng độ dài của khóa. Quy trình sẽ được chia ra làm 4 giai đoạn chính.
- Bước 1: Xử lý dữ liệu đầu vào.
 - Đầu vào: Chuỗi văn bản mã hóa thô (ciphertext).



Hình 15. Letter.

- Xử lý: Loại bỏ toàn bộ các ký tự đặc biệt, khoảng trắng, số... chỉ giữ lại các ký tự chữ cái (A-Z) và chuyển đổi tất cả về chữ in hoa để đồng nhất việc tính toán.
- Mục đích: Tạo ra chuỗi text clean để đảm bảo tính chính xác cho các phép toán thống kê sau này.
- Bước 2: Tìm độ dài khóa bằng thuật toán Index of Coincidence
 - Sử dụng IoC để tìm độ dài khóa m.

```

1 def ic(text):
2     text = text.upper()
3     freq = [0] * 26
4     N = 0
5     for c in text:
6         if 'A' <= c <= 'Z': # Tối ưu hóa kiểm tra
7             freq[ord(c) - ord('A')] += 1
8             N += 1
9     if N < 2:
10        return 0
11    sum_val = sum(f * (f - 1) for f in freq)
12    return sum_val / (N * (N - 1))

```

Hình 16. $Ic(text)$.

```

1 def find_key_length(letters, max_m=20):
2     best_m = 1
3     best_ic = 0
4     for m in range(1, max_m + 1):
5         subsets = ['' for _ in range(m)]
6         for i, c in enumerate(letters):
7             subsets[i % m] += c
8         avg_ic = sum(ic(s) for s in subsets) / m
9
10        if avg_ic > best_ic:
11            best_ic = avg_ic
12            best_m = m
13    return best_m

```

Hình 17. Find key.

- Hàm $ic(text)$ tính toán xác suất hai chữ cái ngẫu nhiên giống nhau.
- Hàm $find_key_length$ thử chia văn bản thành các nhóm theo độ dài m (từ 1 đến 20). Nếu m đúng, các nhóm con sẽ có chỉ số IC cao (gần 0.065).
- Đây là công thức tính IC:

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

- Kết quả: Chọn m có I.C trung bình lớn nhất làm độ dài khóa (L).
- Bước 3: Khôi phục Key.

- Sau khi có độ dài khóa, thuật toán tìm từng ký tự của khóa bằng phương pháp Chi-square.

```

1 def chi_square(freq, expected):
2     chi = 0
3     for i in range(26):
4         if expected[i] > 0:
5             chi += (freq[i] - expected[i]) ** 2 / expected[i]
6     return chi

```

Hình 18. Chi-square.

```

1 def find_key_from_subsets(subsets):
2     key = ''
3     for s in subsets:
4         freq = [0] * 26
5         for c in s:
6             freq[ord(c) - ord('A')] += 1
7         total = len(s)
8         expected = [total * english_freq[i] for i in range(26)]
9
10        best_shift = 0
11        best_chi = float('inf')
12
13        for shift in range(26):
14            new_freq = [0] * 26
15            for j in range(26):
16                new_freq[(j - shift) % 26] += freq[j]
17            chi = chi_square(new_freq, expected)
18            if chi < best_chi:
19                best_chi = chi
20                best_shift = shift
21        key += chr(best_shift + ord('A'))
22    return key

```

Hình 19. Finde_key_from_subnet.

- Duyệt từng nhóm: Xét từng nhóm con thứ i (tương ứng với ký tự thứ i của khóa).
- Thử dịch chuyển: Với mỗi nhóm, thử giải mã bằng tất cả 26 phép dịch chuyển Caesar (Shift 0 -> 25).
- So khớp tần suất: Tại mỗi phép dịch, tính tần suất ký tự của nhóm đó và so sánh với Bảng tần suất tiếng Anh chuẩn (english_freq) bằng công thức Chi-square:

$$\chi^2 = \sum \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}}$$

- Chọn ký tự khóa: Phép dịch nào cho ra giá trị χ^2 nhỏ nhất (tức là giống tiếng Anh nhất) sẽ được chọn làm ký tự khóa tại vị trí đó.
- Ghép khóa: Lặp lại cho tất cả các nhóm để ghép thành chuỗi khóa hoàn chỉnh (biến key).

d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử.

Khởi động Server:

- Biên dịch và chạy: Sử dụng lệnh python <tên file> để chạy file python và Flask sẽ khởi tạo một Web Server cục bộ (Localhost).
- Trên terminal sẽ hiển thị thông báo Running on <http://127.0.0.1:5000>. Thì lúc này chương trình đã được khởi động trên đường link đó.

```

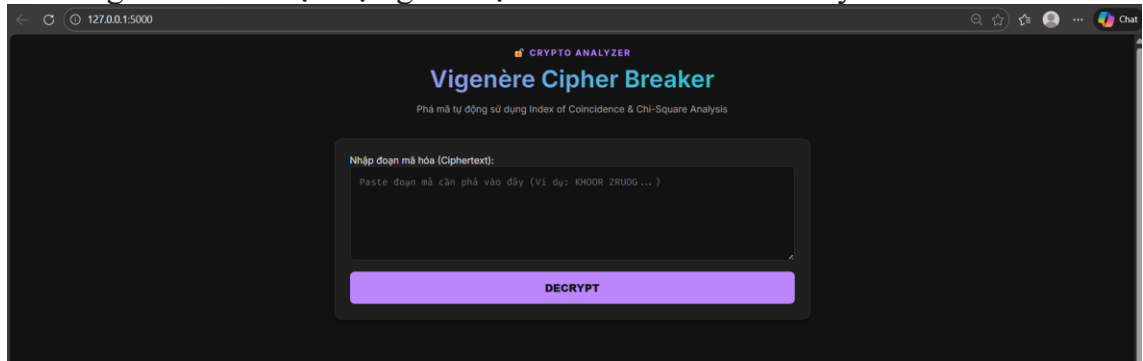
PS E:\NT101_AnToanMang\Lab6\vigenere_web> python .\app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 518-049-137

```

Hình 20. Url.

Truy cập Web:

- Mở trình duyệt và truy cập <http://127.0.0.1:5000>
- Trình duyệt sẽ gửi đến server yêu cầu để truy cập web.
- Server phản hồi: Hàm index() trong code được gọi và server lấy file index.html, điền các giá trị rỗng (vì chưa có kết quả) và gửi về trình duyệt.
- Cuối cùng ta sẽ hiển thị được giao diện web như hình dưới đây.

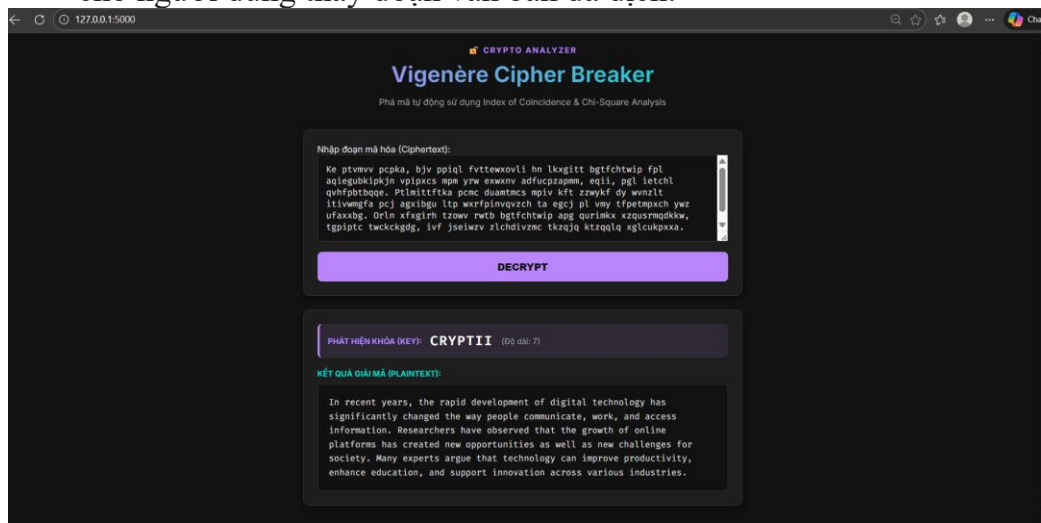


Hình 21. Giao diện Vigenere.

Decryption:

- Nó hoạt động như một máy phân tích ngược: nhận vào văn bản mã hóa, tự động tính toán để tái tạo lại chiếc chìa khóa (Key) đã mất, sau đó mở khóa toàn bộ văn bản.
1. Gửi dữ liệu:
 - Trình duyệt đóng gói chuỗi ciphertext và gửi một yêu cầu (thường là POST) về Server (Flask).
 2. Tiếp nhận:
 - Server nhận yêu cầu và gọi hàm wrapper solve(ciphertext)
 - Code nạp bảng tần suất chuẩn english_freq (dữ liệu cứng trong code chứa tỷ lệ xuất hiện của A-Z) để làm mốc so sánh.
 3. Kích hoạt Logic Thuật toán (Phân tích Thống kê IC & Chi-square):
 - Tìm độ dài khóa: Server chạy hàm find_key_length, thử chia văn bản thành các nhóm con. Nó tính chỉ số IC cho từng nhóm và chọn ra độ dài khóa (\$m\$) mà tại đó các nhóm có chỉ số IC gần với tiếng Anh nhất (0.065).

- Tìm từ khóa: Server chạy hàm `find_key_from_subsets`. Với mỗi nhóm con (tương ứng 1 ký tự khóa), nó thử dịch chuyển 26 lần và dùng kiểm định Chi-square để so sánh với `english_freq`.
4. Đánh giá & Lưu trữ:
- Tại mỗi vị trí của khóa, thuật toán chọn ký tự có độ sai lệch Chi-square thấp nhất (tức là giống tiếng Anh nhất).
 - Sau khi ghép được trọn vẹn Key (ví dụ: "SECRET"), Server gọi hàm `decrypt_vigenere` để thực hiện phép trừ đại số một lần duy nhất, biến toàn bộ bản mã thành Plaintext tiếng Anh có nghĩa và trả về kết quả.
 - Ghi file: Server tự động tạo/ghi đè file `plaintext.txt` với nội dung Key và kết quả vừa tìm được.
5. Trả kết quả:
- Trình duyệt nhận dữ liệu JSON này và hiển thị lên màn hình giao diện web cho người dùng thấy đoạn văn bản đã dịch.



Hình 22. Output.

4. DES — Cài đặt các chế độ hoạt động.

a) Ý tưởng và ngôn ngữ được sử dụng.

- Ý tưởng:
 - Mấu chốt của DES là sử dụng cấu trúc mạng Feistel kết hợp với các phép toán trên bit (thay vì ký tự) để đạt được hai tính chất mật mã: Confusion và Diffusion.
 - Quy trình thực hiện chia làm 3 cơ chế chính:
 - Sinh khóa (Key Schedule): Từ 1 khóa chính (64-bit), sử dụng phép dịch bit và hoán vị để tạo ra 16 khóa con riêng biệt cho 16 vòng lặp.
 - Hàm F (Feistel Function): Sử dụng các bảng tra cứu cố định (S-Boxes) để thay thế dữ liệu phi tuyến tính và các bảng hoán vị (P-Box) để xáo trộn vị trí bit.

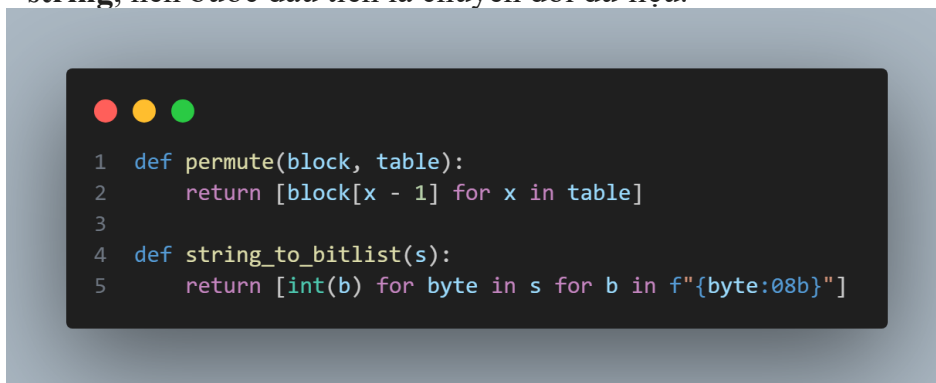
- Chế độ vận hành (Modes of Operation): Ngoài việc mã hóa khối cơ bản (ECB), triển khai thêm chế độ CBC (sử dụng Vector khởi tạo - IV và phép XOR chuỗi) để khắc phục điểm yếu lặp lại của ECB
- Ngôn ngữ được sử dụng:
 - Python: Ngôn ngữ chính để cài đặt thuật toán từ con số 0. Sử dụng List và String để mô phỏng thao tác xử lý trên từng bit và phép XOR, hoàn toàn không sử dụng thư viện mật mã có sẵn.
 - HTML, CSS: Xử lý giao diện người dùng, tạo form nhập Key (bắt buộc 8 ký tự), chọn Mode (ECB/CBC) và hiển thị kết quả dạng Hex/String.
 - Flask (Web Framework): Đóng vai trò Backend, nhận yêu cầu từ giao diện, kích hoạt module DES để mã hóa/giải mã và trả kết quả về trình duyệt.

b) Chức năng chính của chương trình.

- Chức năng chính là mã hóa và giải mã dữ liệu bằng cách thực hiện thuật toán mã hóa đối xứng DES chuẩn trên từng khối dữ liệu 64-bit.
- Hỗ trợ đa chế độ vận hành: ECB và CBC
- Tự động xử lý đệm (PKCS #7): Tự động thêm các byte đệm vào cuối dữ liệu đầu vào nếu độ dài không chia hết cho 8 bytes (64 bits), đảm bảo thuật toán hoạt động đúng với mọi độ dài văn bản.
- Chuyển đổi định dạng dữ liệu: Hỗ trợ chuyển đổi kết quả mã hóa (Ciphertext) sang dạng chuỗi Hex để thuận tiện cho việc hiển thị trên giao diện web và lưu trữ/truyền tải.

c) Giải thuật.

- Ứng dụng có chức năng mã hóa và giải mã DES với 2 chế độ hoạt động cơ bản là ECB, CBC và áp dụng thuật toán đệm PKCS #7.
- Bước 1: Xử lý dữ liệu và Bit (Helper function)
 - Vì DES thao tác trên từng **bit** (nhị phân), nhưng Python làm việc với **byte** và **string**, nên bước đầu tiên là chuyển đổi dữ liệu.



Hình 23. Helper.

- Chuyển đổi chuỗi ký tự sang danh sách các bit (0 và 1) và ngược lại. Các bảng hoán vị (Permutation Tables) được dùng để xáo trộn vị trí các bit.
- Bước 2: Sinh khóa (Key schedule).
 - DES sử dụng khóa chính 64-bit để tạo ra 16 khóa con (Subkeys) 48-bit cho 16 vòng lặp.

```

1  def generate_keys(key):
2      key_bits = string_to_bitlist(key)
3      key_pc1 = permute(key_bits, PC1)
4
5      left, right = key_pc1[:28], key_pc1[28:]
6
7      shifts = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,1]
8
9      keys = []
10     for shift in shifts:
11         left = left[shift:] + left[:shift]
12         right = right[shift:] + right[:shift]
13         keys.append(permute(left + right, PC2))
14
15     return keys

```

Hình 24. Generate key.

- Logic:
 - PC-1: Khóa 64-bit đi qua hoán vị PC1, loại bỏ 8 bit kiểm tra chẵn lẻ, còn lại 56 bit.
 - Chia đôi: Tách thành 2 nửa left (28 bit) và right (28 bit).
 - Dịch vòng (Shift): Trong mỗi vòng, cả hai nửa được dịch trái 1 hoặc 2 bit (theo mảng shifts).
 - PC-2: Gộp hai nửa lại và đi qua hoán vị PC2 để chọn ra 48 bit làm khóa con (Ki).
- Bước 3: Hàm Feistel.
 - Đây là thành phần quan trọng nhất, thực hiện các phép biến đổi phi tuyến tính để tạo sự "hỗn loạn" (Confusion).

```

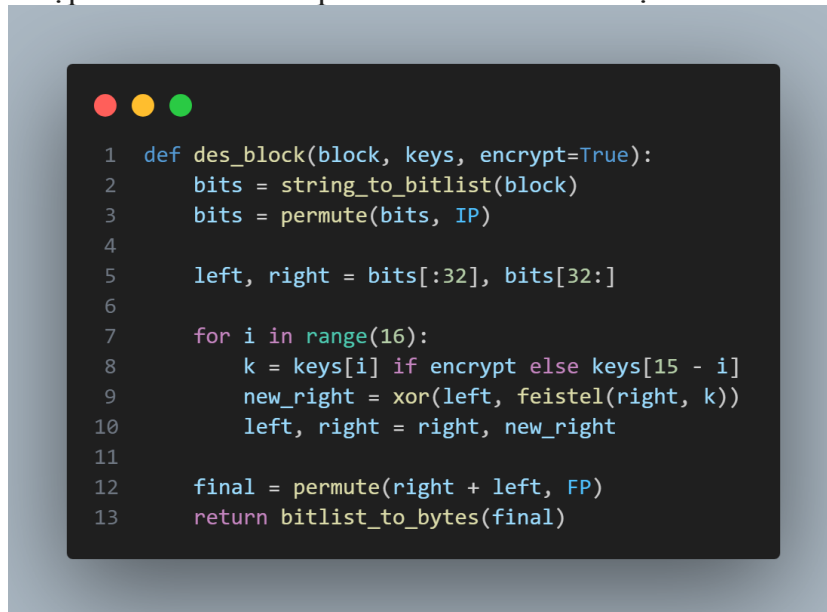
1  def feistel(right, subkey):
2      expanded = permute(right, E)
3      tmp = xor(expanded, subkey)
4
5      out = []
6      for i in range(8):
7          block = tmp[i*6:(i+1)*6]
8          row = (block[0] << 1) | block[5]
9          col = int("".join(str(b) for b in block[1:5]), 2)
10         out += [int(b) for b in f"{S_BOX[i][row][col]:04b}"]
11
12     return permute(out, P)

```


Hình 25. Feistel.

- Logic:

- Mở rộng (Expansion - E): Đầu vào 32-bit (nửa phải \$R\$) được mở rộng thành 48-bit bằng bảng E.
 - Trộn khóa (Key Mixing): Thực hiện phép XOR giữa kết quả trên với khóa con (\$K_i\$).
 - Thay thế (Substitution - S-Boxes): Chia 48 bit thành 8 nhóm (mỗi nhóm 6 bit). Mỗi nhóm đi qua một hộp S_BOX tương ứng để nén lại còn 4 bit. Tra S-Box bằng cách dùng bit đầu và bit cuối tạo thành chỉ số hàng, 4 bit giữa tạo thành chỉ số cột.
 - Hoán vị (Permutation - P): Kết quả 32-bit cuối cùng được xáo trộn qua bảng P.
- Bước 4: Mã hóa Khối DES:
- Kết hợp tất cả các thành phần trên để mã hóa một khối 64-bit.

Hình 26. *Des_block*.

- Logic:
 - Hoán vị khởi tạo (IP): Xáo trộn dữ liệu đầu vào.
 - Chia đôi: Tách thành L0 và R0.
 - Vòng lặp (16 Rounds):
 - Lưu lại Rcũ.
 - Lmới = Rcũ
 - R mới = L cũ XOR với Feistel
 - Hoán vị kết thúc (FP): Gộp \$L\$ và \$R\$ (đảo ngược thứ tự) và qua bảng FP.
- Chế độ hoạt động ECB:
 - Hàm mã hóa ECB:




```

1 def des_ecb_encrypt(data, key):
2     keys = generate_keys(key)
3     data = pad(data)
4     ct = b''
5
6     for i in range(0, len(data), 8):
7         ct += des_block(data[i:i+8], keys, True)
8
9     return ct

```

Hình 27. Des

- Logic:
 - Đầu tiên sẽ sinh ra khóa con với generate_keys.
 - Hàm pad sẽ thêm các byte giả (theo chuẩn PKCS#7) vào cuối chuỗi nếu dữ liệu gốc không đủ độ dài.
 - Khởi tạo biến chứa kết quả ct = 'b'
 - Vòng lặp mã hóa từng khối:
 - for i in range(0, len(data), 8): Vòng lặp duyệt qua dữ liệu, mỗi bước nhảy 8 bytes.
 - des_block(data[i:i+8], keys, True): Cắt ra một khối 8 bytes: data[i:i+8]. Gọi hàm lõi des_block với tham số encrypt=True để thực hiện mã hóa khối đó.
 - ct += ...: Kết quả mã hóa của khối này được nối đuôi ngay vào biến ct.
 - Hàm trả về chuỗi nhị phân ct chứa toàn bộ nội dung đã mã hóa.
- Hàm giải mã ECB:



```

1 def des_ecb_decrypt(data, key):
2     keys = generate_keys(key)
3     pt = b''
4
5     for i in range(0, len(data), 8):
6         pt += des_block(data[i:i+8], keys, False)
7
8     return unpad(pt)

```

Hình 28. Des_ecb_devrypt.

- Logic:
 - Đầu tiên sẽ sinh ra khóa con với generate_keys.
 - pt = b'': Biến này dùng để tích lũy các khối plaintext sau khi giải mã.
 - Vòng lặp giải mã từng khối:

- for i in range(0, len(data), 8): Duyệt qua chuỗi ciphertext (lúc này chắc chắn độ dài đã là bội số của 8 nên không cần padding).
 - des_block(data[i:i+8], keys, False): Cắt ra một khối 8 bytes ciphertext. Gọi hàm lỗi des_block với tham số encrypt=False (chế độ giải mã). Lúc này, các khóa con sẽ được sử dụng theo thứ tự ngược lại (từ K16 về K1).
 - pt += ...: Kết quả giải mã (plaintext) được nối vào biến pt.
- Chế độ hoạt động CBC:
- o Hàm mã hóa CBC:



```

1 def des_cbc_encrypt(data, key, iv=None):
2     keys = generate_keys(key)
3     data = pad(data)
4
5     if iv is None:
6         iv = os.urandom(8)
7
8     ct = b''
9     prev = iv
10
11    for i in range(0, len(data), 8):
12        block = xor(string_to_bitlist(data[i:i+8]), string_to_bitlist(prev))
13        block = bitlist_to_bytes(block)
14        encrypted = des_block(block, keys, True)
15        ct += encrypted
16        prev = encrypted
17
18    return ct, iv

```

Hình 29. Des_cbc_encrypt

- o Logic:
 - Đầu tiên sẽ sinh ra khóa con với generate_keys.
 - Hàm pad sẽ thêm các byte giả (theo chuẩn PKCS#7) vào cuối chuỗi nếu dữ liệu gốc không đủ độ dài.
 - if iv is None: iv = os.urandom(8): Nếu người dùng không cung cấp IV, chương trình sẽ tự sinh ngẫu nhiên 8 bytes. IV là yếu tố bắt buộc để mã hóa khối đầu tiên (vì khối đầu tiên không có "khối trước nó").
 - Khởi tạo biến chứa kết quả và biến prev để lưu giữ giá trị của khối mã hóa trước đó. Ở vòng lặp đầu tiên, nó chính là IV.
 - Vòng lặp mã hóa chuỗi (Chaining Loop):
 - for i in range(0, len(data), 8): Duyệt từng khối 8 bytes.
 - Bước XOR (Quan trọng): block = xor(..., prev): Khối dữ liệu rõ (Plaintext) hiện tại được XOR với khối prev (Ciphertext trước đó hoặc IV). Đây là bước giúp che giấu mẫu dữ liệu.
 - Bước Mã hóa: encrypted = des_block(..., keys, True): Kết quả sau khi XOR mới được đưa vào hàm DES để mã hóa.
 - Cập nhật trạng thái: ct += encrypted dùng để lưu kết quả và prev = encrypted là khối vừa mã hóa xong trở thành prev cho vòng lặp tiếp theo.
 - Hàm trả về cả ct (Ciphertext) và iv (vì bên nhận cần có IV này mới giải mã được).

- Hàm giải mã CBC:



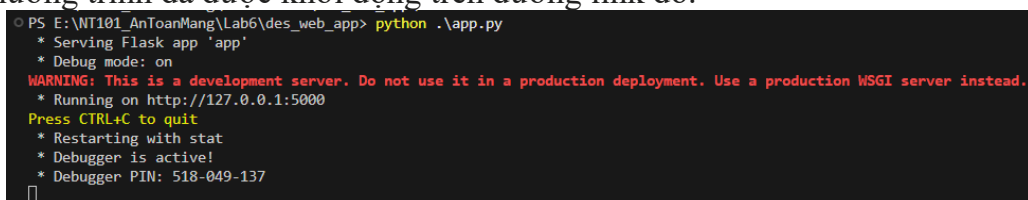
Hình 30. *Des_cbc_decrypt*

- Logic:
 - Đầu tiên sẽ sinh ra khóa con với generate_keys.
 - prev = iv: Để giải mã khối đầu tiên, ta cần IV giống hệt lúc mã hóa.
 - Vòng lặp giải mã:
 - for i in range(0, len(ct), 8): Duyệt từng khối ciphertext.
 - Bước Giải mã trước: Khối ciphertext hiện tại được đưa vào hàm DES (chế độ decrypt) trước tiên.
 - Bước XOR sau: Kết quả sau khi giải mã DES được XOR với khối ciphertext trước đó (prev) để hoàn nguyên về Plaintext gốc.
 - Cập nhật trạng thái: lưu plaintext và lưu ý quan trọng: prev dùng cho vòng sau chính là khối ciphertext hiện tại (chưa qua giải mã), chứ không phải kết quả plaintext.
 - Gỡ bỏ đệm (unpad): Loại bỏ các byte đệm để lấy dữ liệu gốc.

d) Cách hoạt động của chương trình sau khi biên dịch/chạy thử.

Khởi động Server:

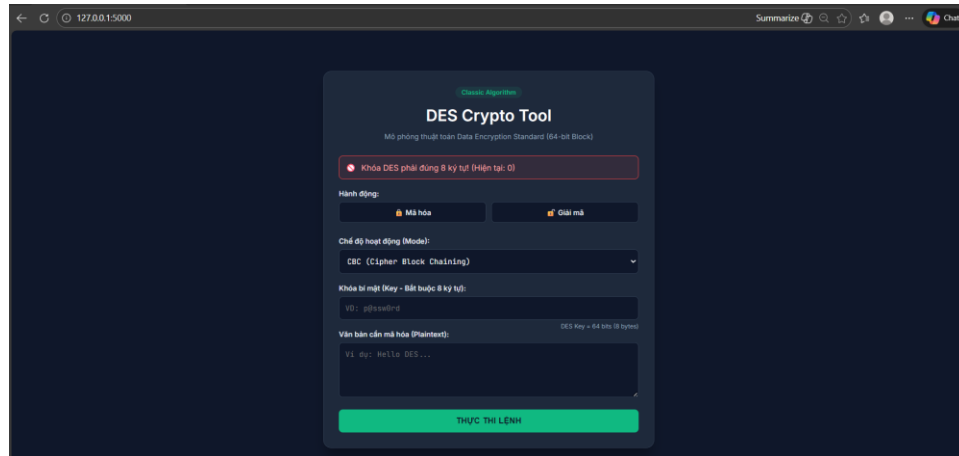
- Biên dịch và chạy: Sử dụng lệnh python <tên file> để chạy file python và Flask sẽ khởi tạo một Web Server cục bộ (Localhost).
- Trên terminal sẽ hiển thị thông báo Running on <http://127.0.0.1:5000>. Thì lúc này chương trình đã được khởi động trên đường link đó.



Hình 31. *Url.*

Truy cập Web:

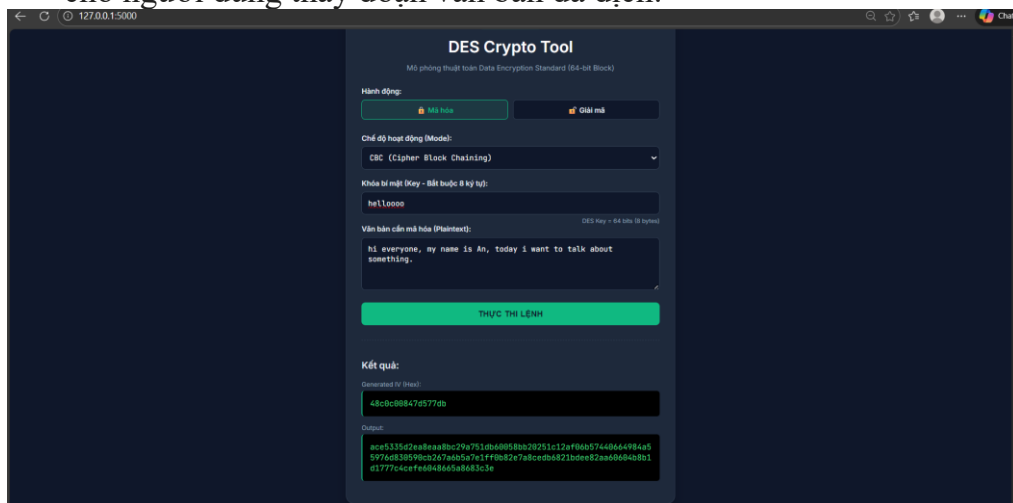
- Mở trình duyệt và truy cập <http://127.0.0.1:5000>
- Trình duyệt sẽ gửi đến server yêu cầu để truy cập web.
- Server phản hồi: Hàm index() trong code được gọi và server lấy file index.html, điền các giá trị rỗng (vì chưa có kết quả) và gửi về trình duyệt.
- Cuối cùng ta sẽ hiển thị được giao diện web như hình dưới đây.



Hình 32. Giao diện Des.

Encryption (Mã hóa):

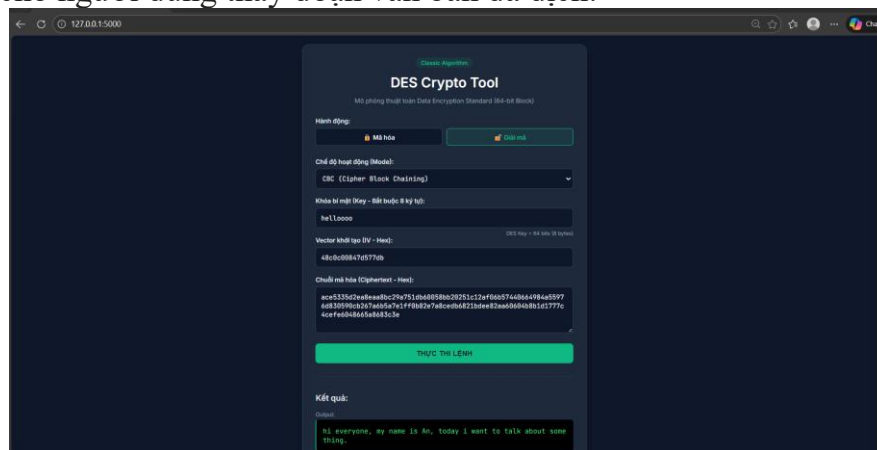
- Nó hoạt động như một máy khóa kỹ thuật số: nhận vào văn bản rõ (Plaintext) và khóa bí mật (Key), sau đó xáo trộn các bit dựa trên cấu trúc mạng Feistel để tạo ra chuỗi mã hóa không thể đọc được.
1. Gửi dữ liệu:
 - Trình duyệt đóng gói chuỗi plaintext và gửi một yêu cầu (thường là POST) về Server (Flask).
 2. Tiếp nhận:
 - Server nhận yêu cầu và gọi hàm wrapper encrypt(plaintext, key, mode).
 - Code chuyển đổi chuỗi Plaintext và Key từ dạng ký tự sang dạng bytes để chuẩn bị xử lý cấp độ bit.
 3. Kích hoạt Logic Thuật toán (Feistel và Padding):
 - Xử lý đệm (Padding): Server chạy hàm pad(data). Vì DES xử lý theo khối 64-bit (8 bytes), nếu văn bản không chia hết cho 8, thuật toán tự động thêm các byte giả theo chuẩn PKCS#7 để đủ độ dài khối.
 - Mã hóa với 2 chế độ tự chọn EBC và CBC được phân tích ở phần giải thuật.
 4. Trả kết quả:
 - Trình duyệt nhận dữ liệu JSON này và hiển thị lên màn hình giao diện web cho người dùng thấy đoạn văn bản đã dịch.



Hình 33. Output.

Encryption (Mã hóa):

- Nó hoạt động như quy trình đảo ngược (Reverse Engineering) có kiểm soát: nhận vào chuỗi Hex (Ciphertext) và Khóa, sử dụng chính các khóa con đã tạo nhưng theo thứ tự ngược lại để khôi phục văn bản gốc.
1. Gửi dữ liệu:
 - Trình duyệt đóng gói chuỗi ciphertext và gửi một yêu cầu (thường là POST) về Server (Flask).
 2. Tiếp nhận:
 - Server nhận yêu cầu và gọi hàm wrapper `decrypt(ciphertext_hex, key, mode)`.
 - Code thực hiện giải mã chuỗi Hex thành bytes để đưa vào thuật toán.
 3. Kích hoạt Logic Thuật toán (Feistel ngược và Unpadding):
 - Gỡ bỏ đệm (Unpadding): Sau khi có được chuỗi plaintext thô, server chạy hàm `unpad(pt)`. Hàm này đọc giá trị của byte cuối cùng để biết cần cắt bỏ bao nhiêu byte đệm PKCS#7, trả về văn bản gốc sạch sẽ cho người dùng.
 - Giải mã với 2 chế độ tự chọn ECB và CBC được phân tích ở phần giải thuật.
 4. Trả kết quả:
 - Trình duyệt nhận dữ liệu JSON này và hiển thị lên màn hình giao diện web cho người dùng thấy đoạn văn bản đã dịch.



Hình 34. Output.

5. AES — Cài đặt các chế độ hoạt động.**a) Ý tưởng và ngôn ngữ được sử dụng.**

- Ý tưởng:
 - Khác với DES (dùng Feistel), AES sử dụng cấu trúc SPN - Substitution-Permutation Network kết hợp với các phép toán đại số trên trường hữu hạn $GF(2^8)$. Điều này giúp thuật toán đạt được tốc độ cao và độ an toàn mạnh mẽ thông qua hai tính chất: Confusion và Diffusion.
 - Quy trình thực hiện chia làm 3 cơ chế chính:
 - Mở rộng khóa (Key Expansion): Từ 1 khóa chính 128-bit (16 bytes), sử dụng bảng S-Box và các hằng số vòng (Rcon) để tạo ra một chuỗi khóa mở rộng gồm 176 bytes, chia thành 11 bộ khóa con (Round Keys) phục vụ cho 10 vòng lặp.

- Các phép biến đổi trạng thái (State Transformations): Thay vì chia đôi khối dữ liệu như DES, AES xử lý toàn bộ ma trận 4x4 (State).
- Chế độ vận hành (Modes of Operation):
 - CBC: Sử dụng Vector khởi tạo (IV), phép XOR chuỗi và đệm PKCS#7 để đảm bảo tính ngẫu nhiên và an toàn cho dữ liệu khối.
 - CFB: Biến mã hóa khối thành mã hóa dòng (stream cipher), cho phép xử lý dữ liệu liên tục bằng cách mã hóa lại ciphertext trước đó.
- Ngôn ngữ được sử dụng:
 - Python: Ngôn ngữ chính để cài đặt thuật toán từ con số 0. Sử dụng các phép toán trên bit và tính toán đại số trên trường hữu hạn $GF(2^8)$ để thực hiện các phép biến đổi ma trận phức tạp (như MixColumns), hoàn toàn không sử dụng thư viện mật mã có sẵn.
 - HTML, CSS: Xử lý giao diện người dùng, tạo form nhập Key (bắt buộc 16 ký tự cho AES-128), chọn Mode (CBC/CFB) và hiển thị kết quả dạng Hex/String.
 - Flask (Web Framework): Đóng vai trò Backend, nhận yêu cầu từ giao diện, kích hoạt module DES để mã hóa/giải mã và trả kết quả về trình duyệt.

b) Chức năng chính của chương trình.

- Chức năng chính là mã hóa và giải mã dữ liệu bằng cách thực hiện thuật toán mã hóa đối xứng AES với độ dài khóa 128-bit.
- Hỗ trợ đa chế độ vận hành: CBC và CFB
- Tự động xử lý đệm (PKCS #7): Đối với chế độ CBC, chương trình tự động tính toán và thêm các byte đệm vào cuối dữ liệu nếu độ dài không chia hết cho 16 bytes (128 bits), đảm bảo tính toàn vẹn của cấu trúc khối.
- Hỗ trợ chuyển đổi kết quả mã hóa và IV sang dạng chuỗi Hexadecimal (Hệ 16) để dễ dàng hiển thị trên giao diện web hoặc lưu trữ.

c) Giải thuật.

- Ứng dụng có chức năng mã hóa và giải mã DES với 2 chế độ hoạt động cơ bản là CBC, CFB và áp dụng thuật toán đệm PKCS #7.
- Bước 1: Xử lý dữ liệu và Bit (Helper function)
 - Khác với các thuật toán cổ điển, AES thực hiện tính toán trên trường hữu hạn $GF(2^8)$. Các phép toán cộng là XOR, còn phép nhân là nhân đa thức module.

```

1 def xtime(x: Byte) -> Byte:
2     x &= 0xFF
3     return ((x << 1) ^ 0x1B) & 0xFF if (x & 0x80) else (x << 1) & 0xFF
4
5 def mul(a: Byte, b: Byte) -> Byte:
6     res = 0
7     a &= 0xFF
8     b &= 0xFF
9     while b:
10        if b & 1:
11            res ^= a
12            a = xtime(a)
13            b >>= 1
14    return res & 0xFF

```

Hình 35. Helper function.

- Hàm xtime: Thực hiện nhân một số với x (tương đương nhân 2) trong GF(2⁸). Nếu kết quả tràn 8 bit, nó sẽ XOR với đa thức bất quy tắc 0x1B.
- Hàm mul: Thực hiện phép nhân hai số bất kỳ bằng cách kết hợp xtime và phép cộng (XOR)
- Bước 2: Mở rộng khóa (Key expansion):
 - AES-128 cần 11 bộ khóa con (Round Keys) cho 10 vòng lặp + 1 vòng khởi tạo.

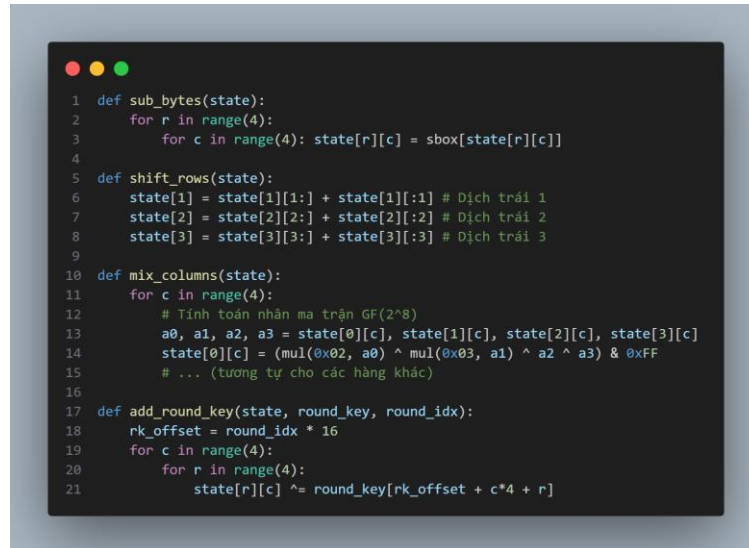
```

1 def key_expansion(key: bytes) -> List[Byte]:
2     if len(key) != 16:
3         raise ValueError("Key must be 16 bytes for AES-128")
4     w = list(key)[:] # initial 16 bytes
5     bytes_produced = 16
6     rcon_iter = 1
7     while bytes_produced < 176:
8         temp = w[bytes_produced-4:bytes_produced]
9         if bytes_produced % 16 == 0:
10            # rotate
11            temp = temp[1:] + temp[:1]
12            # sub
13            temp = [sbox[t] for t in temp]
14            # Rcon
15            temp[0] ^= Rcon[rcon_iter]
16            rcon_iter += 1
17        for t in temp:
18            w.append(w[bytes_produced - 16] ^ t)
19            bytes_produced += 1
20    return w

```

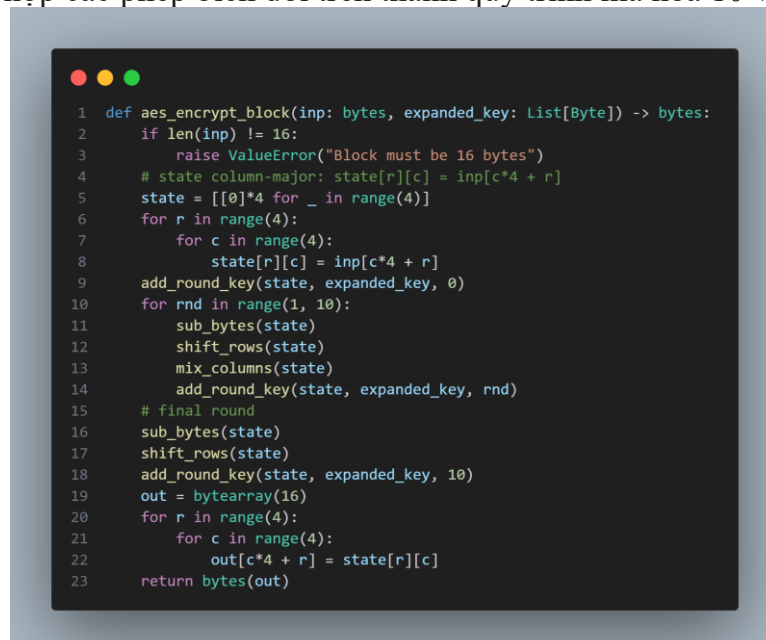
Hình 36. Key_expansion.

- Khóa ban đầu 16 bytes được sao chép vào 16 byte đầu của mảng mở rộng w.
- Các từ (word - 4 bytes) tiếp theo được tạo ra dựa trên từ trước đó và từ cách đó 4 vị trí.
- Mỗi 4 từ (16 bytes), thuật toán áp dụng hàm RotWord (dịch vòng), SubWord (tra bảng S-Box) và XOR với hằng số vòng Rcon để tạo tính phi tuyến tính.
- Bước 3: Các phép biến đổi trạng thái (State Transformations)
 - AES xử lý dữ liệu dưới dạng ma trận trạng thái (State) 4x4 bytes. Mỗi vòng lặp bao gồm 4 bước biến đổi



Hình 37. State.

- SubBytes (Thay thế): Thay thế từng byte trong ma trận bằng giá trị tương ứng trong bảng sbox. Đây là bước phi tuyến tính chính.
- ShiftRows (Dịch hàng): Hàng 0 giữ nguyên, Hàng 1 dịch trái 1, Hàng 2 dịch trái 2, Hàng 3 dịch trái 3.
- MixColumns (Trộn cột): Nhân ma trận trạng thái với một ma trận hằng số trong $GF(2^8)$. Bước này giúp khuếch tán dữ liệu mạnh mẽ.
- AddRoundKey (Cộng khóa): XOR ma trận trạng thái với khóa con của vòng hiện tại
- Bước 4: Mã hóa khối AES
 - Kết hợp các phép biến đổi trên thành quy trình mã hóa 10 vòng.



Hình 38. AES block.

- Vòng khởi tạo: Chỉ thực hiện AddRoundKey.
- 9 Vòng lặp chính: Thực hiện đủ 4 bước: SubBytes -> ShiftRows -> MixColumns -> AddRoundKey.

- Vòng cuối (Vòng 10): Giống vòng chính nhưng bỏ qua MixColumns.
- Bước 5: Chế độ hoạt động.
 - Chế độ CBC:
 - CBC (Liên kết Khối Mật mã) là chế độ phổ biến nhất để bảo mật dữ liệu khối. Đặc điểm cốt lõi là việc mã hóa của mỗi khối phụ thuộc vào kết quả của khối trước đó.

```

1 def pkcs7_pad(data: bytes) -> bytes:
2     block = 16
3     pad = block - (len(data) % block)
4     if pad == 0:
5         pad = block
6     return data + bytes([pad])*pad

```

Hình 39. Pkcs7_pad

- Xử lý Đệm (Padding PKCS#7):
 - Do AES làm việc trên các khối cố định 16-byte, CBC yêu cầu tổng độ dài dữ liệu phải là bội số của 16. Hàm pkcs7_pad được gọi đầu tiên để thêm các byte giả vào cuối chuỗi (Ví dụ: thiếu 4 byte thì thêm 04 04 04 04).
- Vector Khởi tạo (IV): Hàm sử dụng một chuỗi ngẫu nhiên 16-byte (gen_iv) làm đầu vào cho khối đầu tiên.
- Quy trình mã hóa dữ liệu

```

1 def aes_cbc_encrypt(plaintext: bytes, key: bytes, iv_out: bytearray=None) -> bytes:
2     expanded = key_expansion(key)
3     data = pkcs7_pad(plaintext)
4     if iv_out is None:
5         iv = gen_iv()
6     else:
7         if len(iv_out) == 0:
8             iv = gen_iv()
9             iv_out.extend(iv)
10        else:
11            if len(iv_out) != 16:
12                raise ValueError("IV must be 16 bytes")
13            iv = bytes(iv_out)
14    prev = bytearray(iv)
15    out = bytearray()
16    for i in range(0, len(data), 16):
17        block = bytes([data[i+j] ^ prev[j] for j in range(16)])
18        enc = aes_encrypt_block(block, expanded)
19        out.extend(enc)
20        prev = bytearray(enc)
21    return bytes(out)

```

Hình 40. Cbc encrypt

- Logic:

- Dữ liệu được chia thành các khối 16-byte (P1, P2, ...).
- Trước khi đi vào hàm mã hóa AES, khối Plaintext hiện tại được XOR với khối Ciphertext trước đó (C_prev).

- Công thức:

$$Block_{input} = P_i \oplus C_{i-1} \text{ (Với } i = 0 \text{ thì } C_{-1} \text{ là IV).}$$

- Sau đó :

$$C_i = AES_Encrypt(Block_{input}).$$

- Biến prev trong code được cập nhật liên tục: prev = bytearray(enc)

- Quy trình giải mã dữ liệu:

```

1 def aes_cbc_decrypt(ciphertext: bytes, key: bytes, iv_in: bytes) -> bytes:
2     if len(ciphertext) % 16 != 0:
3         raise ValueError("Ciphertext not multiple of block size")
4     if len(iv_in) != 16:
5         raise ValueError("IV (16 bytes) required for CBC decrypt")
6     expanded = key_expansion(key)
7     prev = bytearray(iv_in)
8     out = bytearray()
9     for i in range(0, len(ciphertext), 16):
10        block = ciphertext[i:i+16]
11        dec = aes_decrypt_block(block, expanded)
12        plain = bytes([dec[j] ^ prev[j] for j in range(16)])
13        out.extend(plain)
14        prev = bytearray(block)
15    return pkcs7_unpad(bytes(out))

```

Hình 41. Cbc decrypt

- Logic:

- Quá trình ngược lại: Khối Ciphertext được đưa vào hàm giải mã AES trước.
 - Kết quả sau giải mã được XOR với khối Ciphertext trước đó để hoàn nguyên Plaintext.
 - Công thức:
- $$P_i = AES_Decrypt(C_i) \oplus C_{i-1}.$$
- Cuối cùng, hàm pkcs7_unpad loại bỏ các byte đệm.

- Chế độ CFB:

- CFB (Phản hồi Mật mã) là chế độ biến thuật toán mã hóa khối (AES) thành thuật toán mã hóa dòng (Stream Cipher). Đặc điểm nổi bật là nó không cần Padding và có thể xử lý dữ liệu có độ dài bất kỳ.
- Không cần Padding: Vì CFB hoạt động bằng cách XOR dữ liệu với một dòng khóa (keystream), độ dài đầu ra bằng đúng độ dài đầu vào.
- Thanh ghi dịch (Shift Register): Code sử dụng biến shiftreg (khởi tạo bằng IV) đóng vai trò như một bộ nhớ đệm đầu vào cho AES.
- Quy trình mã hóa:

```

1 def aes_cfb_encrypt(plaintext: bytes, key: bytes, iv_out: bytearray=None) -> bytes:
2     expanded = key_expansion(key)
3     if iv_out is None:
4         iv = gen_iv()
5     else:
6         if len(iv_out) == 0:
7             iv = gen_iv()
8             iv_out.extend(iv)
9         else:
10            if len(iv_out) != 16:
11                raise ValueError("IV must be 16 bytes")
12            iv = bytes(iv_out)
13    shiftreg = bytearray(iv)
14    out = bytearray()
15    stream = bytearray(16)
16    si = 16 # force generation on first byte
17    for i in range(len(plaintext)):
18        if si == 16:
19            stream = bytearray(aes_encrypt_block(bytes(shiftreg), expanded))
20            si = 0
21        c = plaintext[i] ^ stream[si]
22        out.append(c)
23        # shift left one byte and append ciphertext byte
24        shiftreg = shiftreg[1:] + bytes([c])
25        si += 1
26    return bytes(out)

```

Hình 42. CFB encrypt.

- Logic:
 - Hàm AES Encrypt mã hóa shiftreg để tạo ra một khối 16-byte dòng khóa (stream).
 - Từng byte của Plaintext (P) được XOR với từng byte của stream để tạo ra Ciphertext (C).
 - Cơ chế Phản hồi: Sau khi mã hóa một byte, byte Ciphertext đó (C) được đẩy vào cuối shiftreg, và shiftreg dịch sang trái 1 byte (loại bỏ byte cũ nhất).
 - Logic này lặp lại liên tục, tạo ra hiệu ứng dòng chảy dữ liệu.
- Quy trình giải mã:

```

1 def aes_cfb_decrypt(ciphertext: bytes, key: bytes, iv_in: bytes) -> bytes:
2     if len(iv_in) != 16:
3         raise ValueError("IV (16 bytes) required for CFB decrypt")
4     expanded = key_expansion(key)
5     shiftreg = bytearray(iv_in)
6     out = bytearray()
7     stream = bytearray(16)
8     si = 16
9     for i in range(len(ciphertext)):
10        if si == 16:
11            stream = bytearray(aes_encrypt_block(bytes(shiftreg), expanded))
12            si = 0
13        p = ciphertext[i] ^ stream[si]
14        out.append(p)
15        shiftreg = shiftreg[1:] + bytes([ciphertext[i]])
16        si += 1
17    return bytes(out)

```

Hình 43. CFB decrypt.

- Điểm đặc trưng của chế độ CFB là quy trình giải mã **sử dụng hàm mã hóa khối (aes_encrypt_block)**, thay vì sử dụng hàm giải mã (aes_decrypt_block) như các chế độ ECB hay CBC.

- **Nguyên lý hoạt động:** Cơ chế này xuất phát từ bản chất của CFB là một kỹ thuật biến đổi mã hóa khối thành **mã hóa dòng (Stream Cipher)**. Trong mô hình này:
 - Khối AES đóng vai trò là một **bộ tạo dòng khóa (Keystream Generator)** chứ không trực tiếp biến đổi văn bản rõ.
 - Phép toán cốt lõi để ẩn/hiện dữ liệu là phép **XOR** giữa Dữ liệu và Dòng khóa.
- Công thức:

$$P_i = C_i \oplus \text{Stream_Byte.}$$

d) *Cách hoạt động của chương trình sau khi biên dịch/chạy thử.*

Khởi động Server:

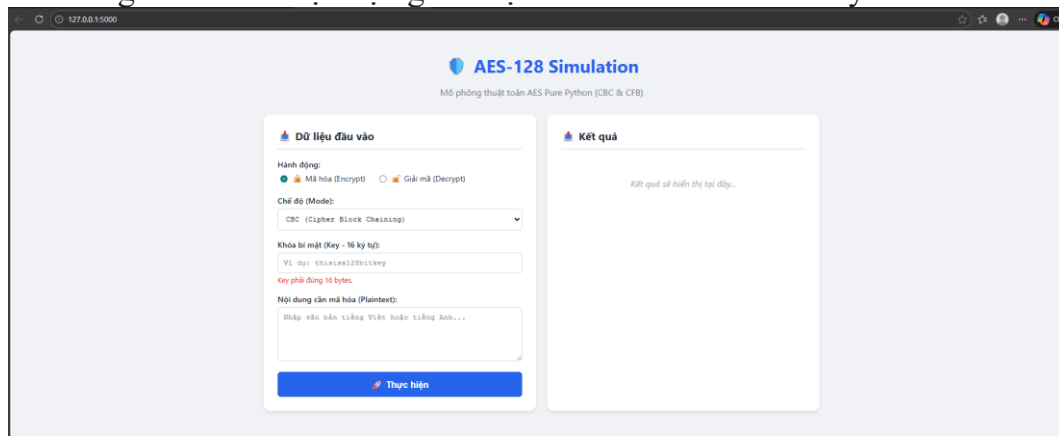
- Biên dịch và chạy: Sử dụng lệnh python <tên file> để chạy file python và Flask sẽ khởi tạo một Web Server cục bộ (Localhost).
- Trên terminal sẽ hiển thị thông báo Running on <http://127.0.0.1:5000>. Thì lúc này chương trình đã được khởi động trên đường link đó.

```
PS E:\NT101_AnToanMang\Lab6\aes_web_app> python .\app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 518-049-137
```

Hình 44. Url.

Truy cập Web:

- Mở trình duyệt và truy cập <http://127.0.0.1:5000>
- Trình duyệt sẽ gửi đến server yêu cầu để truy cập web.
- Server phản hồi: Hàm index() trong code được gọi và server lấy file index.html, điền các giá trị rỗng (vì chưa có kết quả) và gửi về trình duyệt.
- Cuối cùng ta sẽ hiển thị được giao diện web như hình dưới đây.



Hình 45. Giao diện AES.

Encryption (Mã hóa):

- Nó hoạt động như một hệ thống biến đổi ma trận số học: nhận vào văn bản rõ (Plaintext) và khóa bí mật (Key), sau đó biến đổi dữ liệu thông qua cấu trúc mạng thay thế - hoán vị (SPN) trên trường hữu hạn $GF(2^8)$ để tạo ra chuỗi mã hóa an toàn.
1. Gửi dữ liệu:

- Người dùng nhập Plaintext, Key (16 ký tự) và chọn chế độ (CBC/CFB).
 - Trình duyệt đóng gói dữ liệu này và gửi một yêu cầu (POST) về Server (Flask).
2. Tiếp nhận:
- Server nhận yêu cầu và xử lý dữ liệu đầu vào.
 - Code chuyển đổi chuỗi Plaintext sang dạng bytes và kiểm tra độ dài Key (phải đúng 16 bytes/128-bit). Nếu thiếu IV (đối với CBC/CFB), server sẽ tự động sinh ngẫu nhiên.
3. Kích hoạt Logic Thuật toán (SPN và Modes):
- Mở rộng khóa (Key Expansion): Từ 1 khóa chính, thuật toán tạo ra 11 bộ khóa con (Round Keys) để dùng cho 10 vòng lặp.
 - Xử lý chế độ vận hành:
 - Với CBC: Server chạy hàm pkcs7_pad để thêm đệm cho đủ khối 16-byte, sau đó XOR dữ liệu với khối trước đó rồi mới mã hóa.
 - Với CFB: Server sử dụng thanh ghi dịch (Shift Register) để tạo dòng khóa, sau đó XOR với Plaintext (biến mã hóa khối thành mã hóa dòng).
4. Trả kết quả:
- Trình duyệt nhận dữ liệu JSON này và hiển thị lên màn hình giao diện web cho người dùng thấy đoạn văn bản đã dịch.

The screenshot shows the 'AES-128 Simulation' web interface. It has two main panels: 'Dữ liệu đầu vào' (Input Data) and 'Kết quả' (Result).

Dữ liệu đầu vào:

- Hành động:** Radio buttons for 'Mã hóa (Encrypt)' (selected) and 'Giải mã (Decrypt)'.
- Chế độ (Mode):** A dropdown menu set to 'CBC (Cipher Block Chaining)'.
- Khóa bí mật (Key - 16 ký tự):** A text input field containing 'thisis128bitkey'.
- Nội dung cần mã hóa (Plaintext):** A text input field containing 'i love you, my name is An, University of Information Technology, i love volleyball.'
- A red error message below the key field says 'Key phải đúng 16 bytes.'
- A blue 'Thực hiện' (Execute) button at the bottom.

Kết quả:

- IV (Hex) được sinh ra:** A dark box displaying 'a2b396ecd5ef5e3dc235c70e12073292'.
- Output:** A dark box displaying the encrypted ciphertext in hex: '61ae386c89ca878157930aadedea80a4559b91c70e814b4e4c5330ef2c8924f30fe805501a6a9d38983eed567be3cc0747241ad6b34d93adde7522ed4ef325d6563cffe61c77a399ef72cc6c7c74f9b5b9b1131e80532249084cdf96d7793d6'.

Hình 46. Encrypt.

Encryption (Mã hóa):

- Nó hoạt động như một quy trình khôi phục đảo ngược: nhận vào chuỗi Hex (Ciphertext), Khóa và IV, sau đó sử dụng các phép toán ngược (hoặc tái tạo dòng khóa) để hoàn nguyên văn bản gốc.
1. Gửi dữ liệu:
- Người dùng nhập Ciphertext (dạng Hex), Key và IV tương ứng.
 - Trình duyệt đóng gói các tham số này và gửi yêu cầu (POST) về Server.
2. Tiếp nhận:
- Server nhận yêu cầu, kiểm tra tính hợp lệ của chuỗi Hex và chuyển đổi chúng ngược lại thành dữ liệu bytes thô để máy tính xử lý.
3. Kích hoạt Logic Thuật toán (Inverse AES & Stream Generation):

- Tái tạo khóa: Hệ thống vẫn thực hiện mở rộng khóa để có đủ 11 bộ khóa con.
- Xử lý chế độ vận hành:
 - Với CBC: Dữ liệu đi qua hàm giải mã khối (aes_decrypt_block) để đảo ngược các bước biến đổi (InvSubBytes, InvShiftRows...), sau đó XOR với khối mã hóa trước đó và gỡ bỏ đệm (Unpadding).
 - Với CFB (Điểm đặc biệt): Hệ thống sử dụng hàm mã hóa (aes_encrypt_block) lên thanh ghi dịch (chứa IV hoặc Ciphertext cũ) để tái tạo lại dòng khóa (Keystream), sau đó XOR với Ciphertext để ra Plaintext.

4. Trả kết quả:

- Server chuyển đổi dữ liệu bytes đã giải mã sang chuỗi ký tự (String UTF-8).
- Trình duyệt nhận JSON và hiển thị văn bản gốc (Plaintext) lên màn hình cho người dùng đọc.

The screenshot shows the 'AES-128 Simulation' web application. The 'Dữ liệu đầu vào' (Input Data) section on the left contains the following fields:

- Hành động:** Radio buttons for 'Mã hóa (Encrypt)' and 'Giải mã (Decrypt)'. 'Giải mã (Decrypt)' is selected.
- Chế độ (Mode):** A dropdown menu set to 'CBC (Cipher Block Chaining)'.
- Khóa bí mật (Key - 16 ký tự):** A text input field containing 'thisis128bitkey'.
- Initialization Vector (IV - Hex):** A text input field containing 'a2b396cc0d5ef5e3dc235c70e12073292'.
- Chuỗi mã hóa (Ciphertext - Hex):** A text input field containing a long hexadecimal string: '61ae386c89ca878157930eadea80a4559b91c70c814b4e4c5330ef2c8924f30fe805501a6a9d38983eed567be3cc0747241ad6b34d93e1dde75328ed4ef525d6565cfff61c77a389ef72cc6c7c74f9b5b9b1131e80532243084cdf36d7793d6'.
- A blue button labeled 'Thực hiện' (Execute) is at the bottom of the input section.

The 'Kết quả' (Result) section on the right shows the 'Output:' as a dark box with the text: 'i love you, my name is An, University of Information Technology, i love volleyball.'

Hình 47. Decrypt.

6. Tài liệu hướng dẫn.

1. <https://medium.com/@Operaho/make-a-caesars-cipher-with-python-8958ffa1e90d>
2. <https://www.youtube.com/watch?v=sxFObRNriUg>
3. <https://www.dcode.fr/monoalphabetic-substitution>
4. <https://viblo.asia/p/encryption-des-Qpmleq27lrd>
5. <https://www.youtube.com/watch?v=S-vLA7d1ORI>
6. <https://www.geeksforgeeks.org/computer-networks/ecb-mode-vs-cbc-mode-in-cryptography/>
7. <https://www.devglan.com/online-tools/aes-encryption-decryption>