BÁO CÁO THỰC HÀNH

Thực hành An toàn mạng máy tính

Lab 6: Review of Encryption Algorithms

GVTH: Nguyễn Ngọc Trưởng

Ngày báo cáo: 19/12/2025

Nhóm 7 – NT101.Q13.2

THÔNG TIN CHUNG

MSSV	Họ và tên	Nội dung báo cáo	% công việc
23521389	Nguyễn Đình Tâm	Task 1, 2, 3, 4, 5	100%

MỤC LỤC

A. TỔNG QUAN

1. Ngôn ngữ lập trình sử dụng

- Xây dựng theo mô hình Client – Server, trong đó sử dụng Python cho phía backend và ReactJS cho phía frontend.

- Python: được sử dụng để xây dựng máy chủ backend, chịu trách nhiệm xử lý logic và cung cấp API cho frontend.
- ReactJS: được sử dụng để xây dựng giao diện người dùng phía frontend.

=> Sự kết hợp giữa Python và ReactJS giúp hệ thống có kiến trúc rõ ràng, dễ mở rộng, đồng thời đáp ứng tốt các yêu cầu về hiệu năng và tính linh hoạt.

2. Các task đã hoàn thành

- Trong quá trình thực hiện lab đã hoàn thành các task sau:

- **Task 1: Caesar Cipher**
  - Cài đặt thuật toán mã hóa và giải mã Caesar, cho phép thực hiện dịch chuyển ký tự theo khóa xác định, đồng thời hỗ trợ brute-force để tìm khóa trong trường hợp không biết trước.
- **Task 2: Mono-alphabetic Substitution Cipher**
  - Xây dựng hệ mã thay thế đơn bảng chữ cái, cho phép mã hóa và giải mã dựa trên bảng ánh xạ giữa các ký tự, đồng thời hỗ trợ phân tích tần suất để hỗ trợ việc giải mã.
- **Task 3: Vigenère Cipher**
  - Cài đặt thuật toán mã hóa và giải mã Vigenère với khóa nhiều ký tự, giúp tăng mức độ bảo mật so với Caesar cipher, đồng thời hỗ trợ xử lý chuỗi văn bản có độ dài lớn.
- **Task 4: DES - Data Encryption Standard**
  - Triển khai thuật toán DES và các chế độ hoạt động ECB, CBC bao gồm xử lý khóa, vector khởi tạo (IV) và padding dữ liệu.
- **Task 5: AES - Advanced Encryption Standard**

- Triển khai thuật toán AES với các độ dài khóa khác nhau (128-bit, 192-bit, 256-bit), hỗ trợ các chế độ hoạt động ECB, CBC đảm bảo tính bảo mật và linh hoạt trong quá trình sử dụng.

B. BÁO CÁO CHI TIẾT

1. Caesar Cipher – Bruteforce

a) Chức năng chính

- Chức năng Caesar Cipher – Bruteforce được xây dựng nhằm giải mã bản mã Caesar khi không biết khóa.
- Chương trình tự động thử tất cả các khóa dịch chuyển có thể (từ 0 đến 25), đánh giá kết quả giải mã và chọn ra plaintext có khả năng đúng cao nhất dựa trên việc xuất hiện các từ tiếng Anh phổ biến.

b) Giải thuật và ý tưởng chính

- Ý tưởng chính của thuật toán gồm các bước sau:

1. **Bruteforce khóa:** Do Caesar Cipher chỉ có 26 khóa khả dĩ, hệ thống thử lần lượt tất cả các giá trị khóa từ 0 đến 25.
2. **Giải mã với từng khóa:** Với mỗi khóa, bản mã được giải mã để thu được một plaintext ứng viên.
3. **Đánh giá plaintext:** Plaintext được chấm điểm dựa trên số lượng từ tiếng Anh phổ biến xuất hiện (ví dụ: *THE, AND, IS,...*).
4. **Lựa chọn kết quả tốt nhất:** Plaintext có điểm số cao nhất được xem là kết quả giải mã hợp lý nhất.

=> Cách tiếp cận này đơn giản nhưng hiệu quả đối với Caesar Cipher do không gian khóa nhỏ và đặc trưng ngôn ngữ rõ ràng.

c) Cách hoạt động

- Hệ thống nhận ciphertext từ người dùng, thực hiện bruteforce toàn bộ khóa, sau đó:
  - Trả về khoá tìm được
  - Plaintext tương ứng

- Danh sách tất cả các kết quả giải mã để người dùng có thể so sánh và kiểm tra thủ công

d) Chi tiết chương trình

1. Tập hợp các từ tiếng Anh phổ biến

```
COMMON_WORDS = {
    "THE", "AND", "IS", "TO", "OF", "IN", "THAT", "IT", "FOR", "ARE"
}
```

Hình 1: Tập hợp các từ xuất hiện thường xuyên trong tiếng Anh.

- Tập hợp các từ xuất hiện thường xuyên trong tiếng Anh.
- Được sử dụng làm tiêu chí đánh giá mức độ "hợp lý" của plaintext sau khi giải mã.

2. Hàm chấm điểm plaintext

```
def score_text(text: str) -> int:
    words = [
        w for w in text.upper().split()
        if w.isalpha()
    ]
    return sum(1 for w in words if w in COMMON_WORDS)
```

Hình 2: Hàm score_text chấm điểm plaintext

- Giải thích:

- Chuyển toàn bộ văn bản sang chữ hoa để đồng bộ so sánh.
- Tách văn bản thành các từ và loại bỏ các chuỗi không phải chữ cái.
- Đếm số lượng từ trong plaintext trùng với tập COMMON_WORDS.
- Kết quả trả về là điểm số đánh giá chất lượng plaintext.

3. *Hàm giải mã Caesar bằng bruteforce*

```python
def solve_caesar(ciphertext: str):
    all_candidates = []
    best = None
    best_score = -1

    for k in range(26):
        pt = caesar_decrypt(ciphertext, k)
        s = score_text(pt)
        all_candidates.append({"k": k, "pt": pt})

        if s > best_score:
            best_score = s
            best = (k, pt)

    return {
        "key": best[0],
        "plaintext": best[1],
        "allCandidates": all_candidates,
        "bestScore": best_score
    }
```

Hình 3: Hàm solve_caesar giải mã Caesar bằng bruteforce

- Đầu tiên, khởi tạo biến lưu kết quả :

```
all_candidates = []

best = None

best_score = -1
```

- all_candidates: lưu tất cả các plaintext ứng với từng khóa.
- best: lưu khóa và plaintext tốt nhất.
- best_score: lưu điểm số cao nhất tìm được.

- Sau đó, lặp qua tất cả các khoá để giải mã và chấm điểm.

```
for k in range(26):
    pt = caesar_decrypt(ciphertext, k)
    s = score_text(pt)
    all_candidates.append({"k": k, "pt": pt})
```

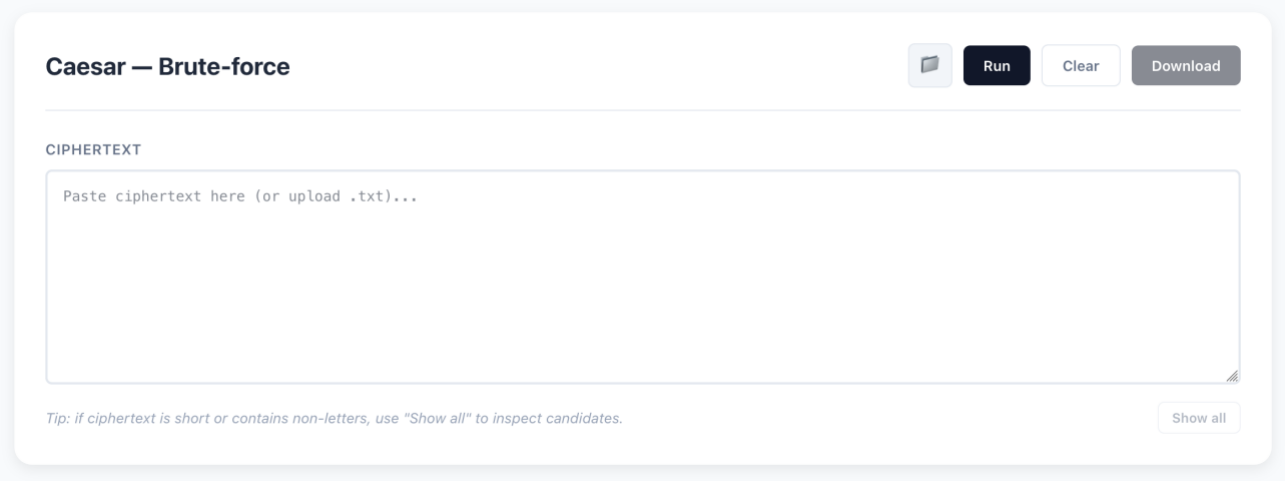
- pt: plaintext thu được sau khi giải mã với khóa k.
- s: điểm số đánh giá plaintext.
- Lưu toàn bộ kết quả vào all_candidates.

- Chọn kết quả tốt nhất và trả về kết quả :

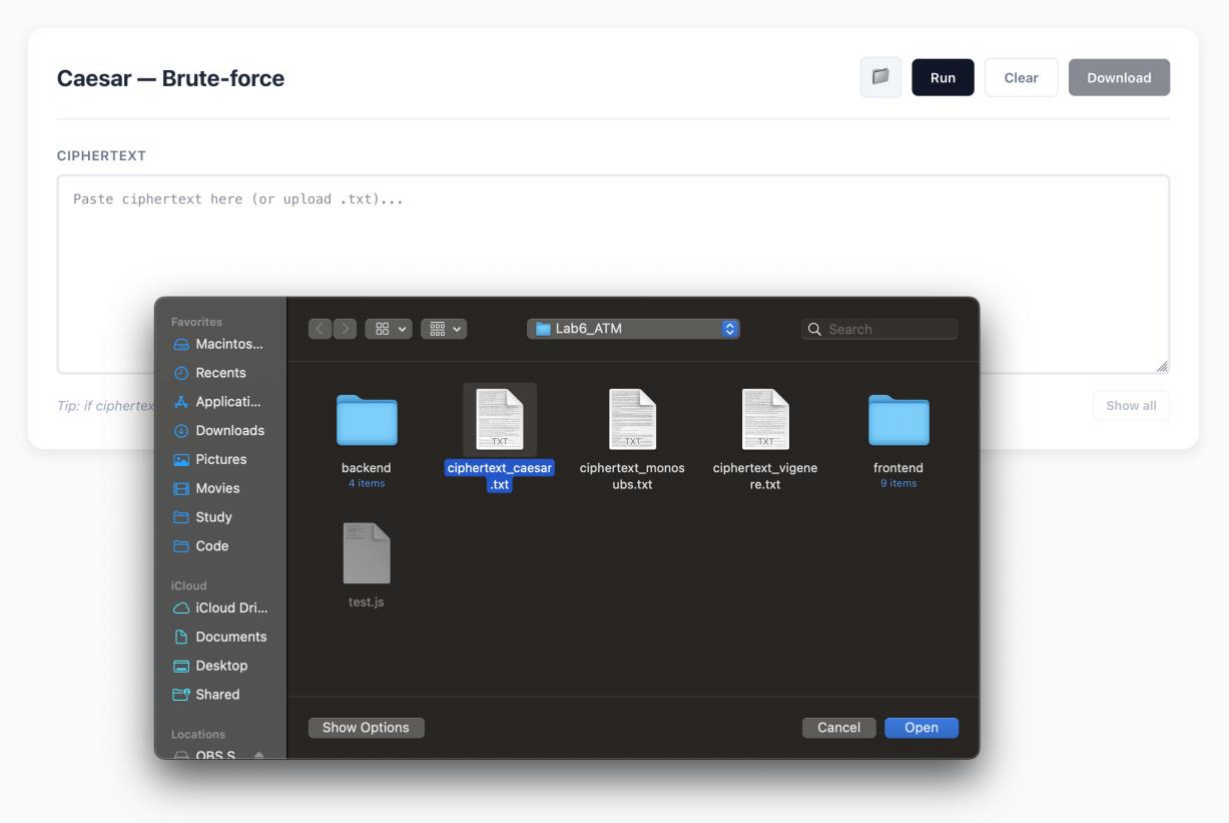
- Khoá tìm được
- Plaintext tốt nhất
- Danh sách toàn bộ kết quả bruteforce
- Điểm số đánh giá

```
if s > best_score:
        best_score = s
        best = (k, pt)


  return {
    "key": best[0],
    "plaintext": best[1],
    "allCandidates": all_candidates,
    "bestScore": best_score
  }
```

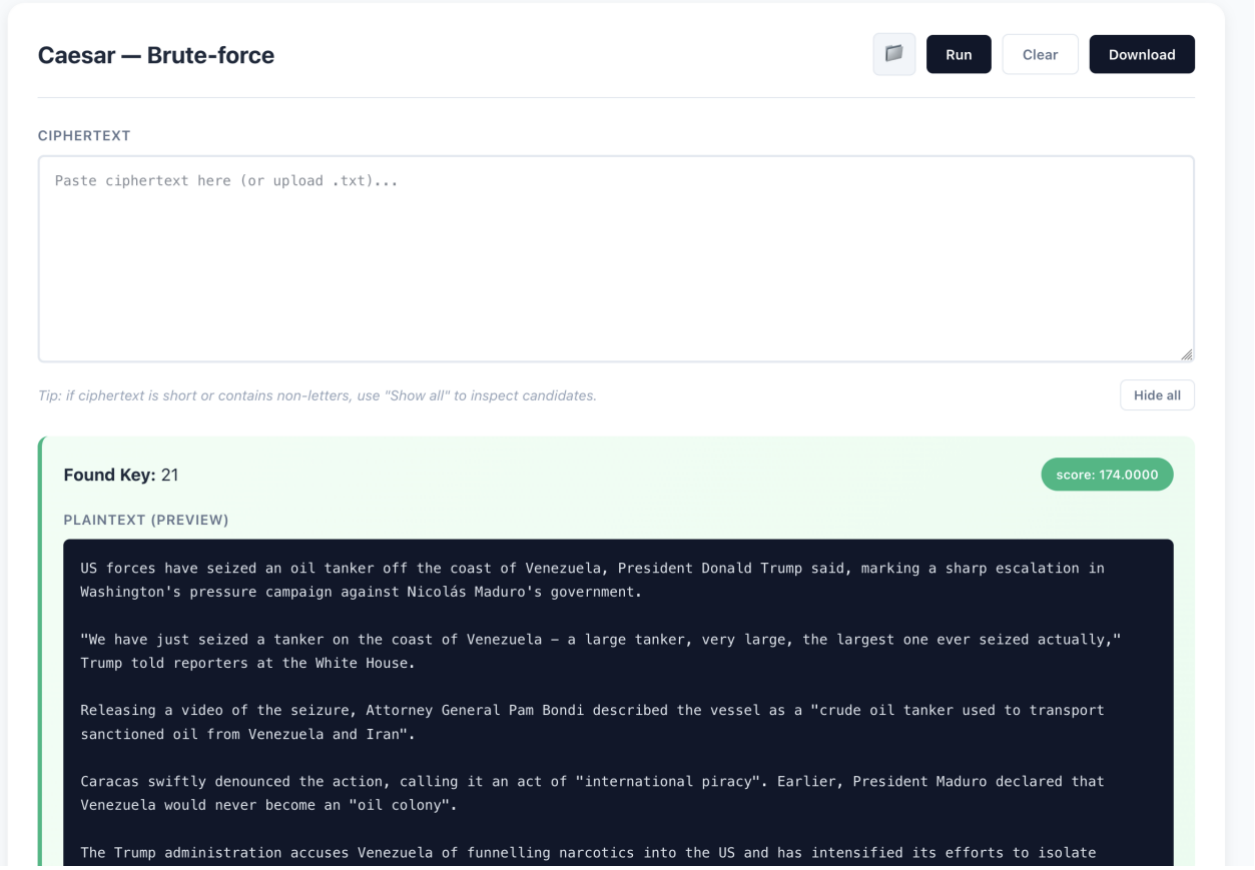
e) Kết quả



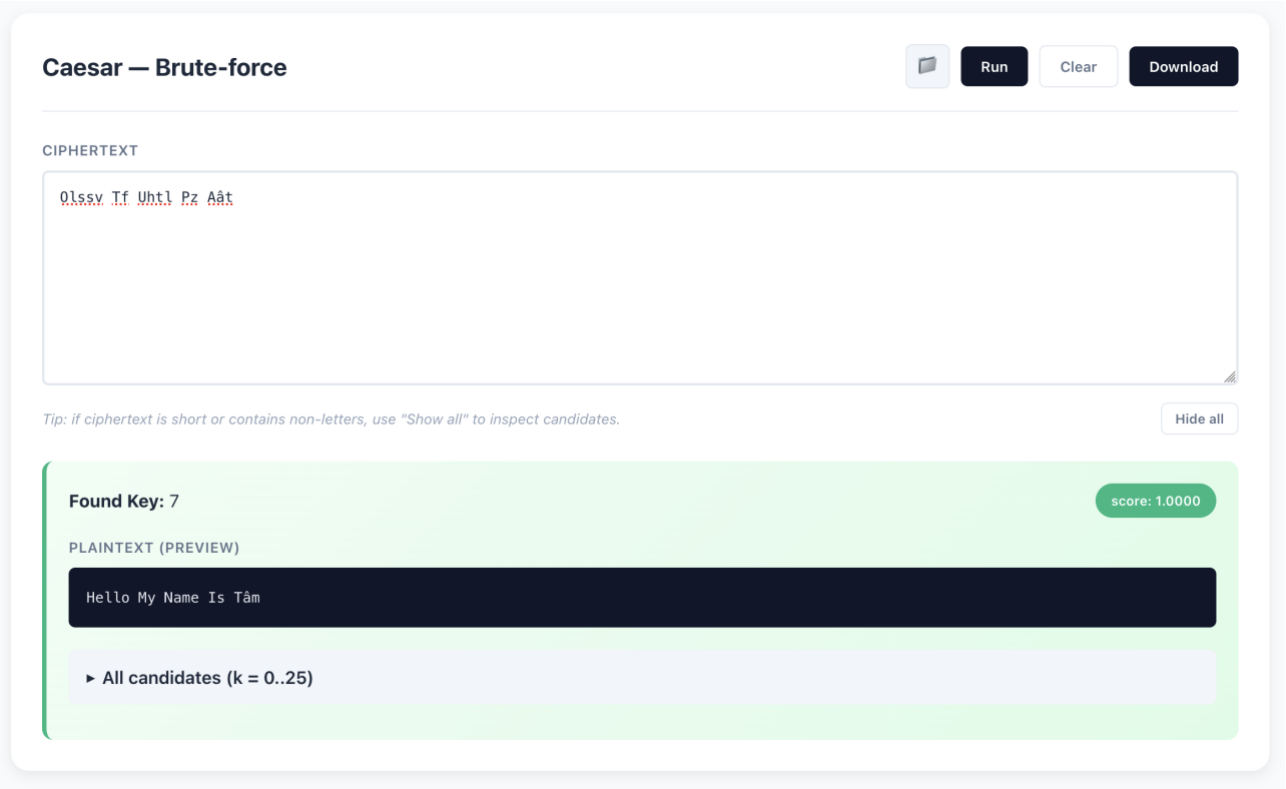
Hình 4: Giao diện của Caesar Cipher



Hình 5: Chọn file txt ciphertext của caesar



Hình 6: Xuất ra kết quả và có thể download file txt plaintext



Hình 7: Ví dụ kết quả

2. Mono-alphabetic substitution cipher

a) Chức năng chính

- Giải mã hệ mã thay thế đơn bảng chữ cái trong trường hợp không biết trước khóa.
- Sử dụng phân tích tần suất kết hợp với mô hình ngôn ngữ tiếng Anh (n-gram) để đánh giá mức độ hợp lý của plaintext.
- Cho phép:
  - ♦ Tự động tìm khóa ánh xạ tối ưu.
  - ♦ Hiển thị thống kê tần suất ký tự.
  - ♦ Trả về plaintext có xác suất đúng cao nhất.

b) Giải thuật và ý tưởng chính

- Ý tưởng chính của thuật toán gồm các bước sau:

• **Bước 1:** Phân tích tần suất ký tự
  - ♦ Thống kê tần suất xuất hiện các chữ cái trong ciphertext và so sánh với thứ tự tần suất chuẩn của tiếng Anh.

• **Bước 2:** Khởi tạo khóa ban đầu
  - ♦ Xây dựng bảng ánh xạ ban đầu giữa ký tự mã hóa và ký tự plaintext dựa trên thứ tự tần suất.

• **Bước 3:** Đánh giá bằng mô hình ngôn ngữ (n-gram)
  - ♦ Sử dụng mono-gram, bi-gram, tri-gram và quad-gram để chấm điểm plaintext thông qua xác suất logarit.

• **Bước 4:** Tối ưu khóa bằng tìm kiếm cục bộ (Hill Climbing)
  - ♦ Thực hiện hoán đổi ngẫu nhiên các ký tự trong khóa, giữ lại khóa mới nếu điểm số tốt hơn.

• **Bước 5:** Nhiều lần khởi động lại (restarts)
  - ♦ Thực hiện nhiều lần tìm kiếm để tránh rơi vào cực trị cục bộ và tìm ra kết quả tối ưu nhất.

=> Cách tiếp cận này giúp giải mã hiệu quả Mono-alphabetic Cipher ngay cả với ciphertext dài và không có thông tin khóa ban đầu.

c) Cách hoạt động

- Hệ thống nhận ciphertext từ người dùng.
- Thực hiện các bước:
  - ♦ Phân tích tần suất chữ cái
  - ♦ Sinh khoá ban đầu
  - ♦ Tối ưu khóa bằng thuật toán hill-climbing
- Sau khi hoàn tất, hệ thống:
  - ♦ Trả về khóa ánh xạ tốt nhất
  - ♦ Giải mã ciphertext thành plaintext
  - ♦
  - ♦ Cung cấp bảng thống kê tần suất ký tự cho frontend hiển thị

d) Chi tiết chương trình

1. Khởi tạo mô hình ngôn ngữ

```
@staticmethod
    def initialize_language_models(folder_path=None):
        if MonoalphabeticAnalyzer._language_model_loaded:
            return

        if folder_path is None:
            current_file_path = os.path.abspath(__file__)
            services_dir = os.path.dirname(current_file_path)

            possible_paths = [
                os.path.join(services_dir, "..", "ngrams"),
                os.path.join(services_dir, "ngrams"),
                os.path.join(os.getcwd(), "ngrams"),
                os.path.join(os.getcwd(), "app", "ngrams"),
            ]
```

```
    folder_path = None
    for path in possible_paths:
        if os.path.exists(path):
            folder_path = path
            print(f"Found ngrams folder at: {folder_path}")
            break

    if folder_path is None:
        print(f"WARNING: Could not find ngrams folder. Tried: {possible_paths}")
        MonoalphabeticAnalyzer._setup_minimal_fallback()
        return

files = {
    'mono': "english_monograms.txt",
    'bi': "english_bigrams.txt",
    'tri': "english_trigrams.txt",
    'quad': "english_quadgrams.txt"
}

for key, filename in files.items():
    full_path = os.path.join(folder_path, filename)
    if not os.path.exists(full_path):
        print(f"WARNING: Could not find {full_path}")
        continue
```

```
            if key == 'mono':
                MonoalphabeticAnalyzer._mono, MonoalphabeticAnalyzer._mono_min =
MonoalphabeticAnalyzer._load_ngram_file(full_path)
            elif key == 'bi':
                MonoalphabeticAnalyzer._bi, MonoalphabeticAnalyzer._bi_min =
MonoalphabeticAnalyzer._load_ngram_file(full_path)
            elif key == 'tri':
                MonoalphabeticAnalyzer._tri, MonoalphabeticAnalyzer._tri_min =
MonoalphabeticAnalyzer._load_ngram_file(full_path)
            elif key == 'quad':
                MonoalphabeticAnalyzer._quad, MonoalphabeticAnalyzer._quad_min =
MonoalphabeticAnalyzer._load_ngram_file(full_path)

        if MonoalphabeticAnalyzer._mono:
            sorted_mono = sorted(MonoalphabeticAnalyzer._mono.items(), key=lambda item: item[1], reverse=True)
            MonoalphabeticAnalyzer._english_frequency_order = "".join([item[0] for item in sorted_mono])

        MonoalphabeticAnalyzer._language_model_loaded = True
```

- Chức năng:
  - Tự động tìm thư mục chứa dữ liệu n-gram.
  - Đọc dữ liệu từ các file: english_monograms.txt, english_bigrams.txt, english_trigrams.txt, english_quadgrams.txt
- Cơ chế:
  - Thử nhiều đường dẫn khác nhau để tăng khả năng tương thích.
  - Nếu không tìm thấy => chuyển sang fallback tối thiểu.

```
@staticmethod
```

```
def _setup_minimal_fallback():
    """Setup minimal English frequency data as fallback"""
    print("Using minimal fallback frequency data")
    MonoalphabeticAnalyzer._english_frequency_order = "etaoinshrdlcumwfgypbvkjxqz"
    MonoalphabeticAnalyzer._language_model_loaded = True
    MonoalphabeticAnalyzer._mono_min = -10
    MonoalphabeticAnalyzer._bi_min = -10
    MonoalphabeticAnalyzer._tri_min = -10
    MonoalphabeticAnalyzer._quad_min = -10
```

- Sử dụng thứ tự tần suất chữ cái chuẩn tiếng Anh.
- Gán giá trị phạt cố định cho tất cả n-gram.
- Đảm bảo chương trình **vẫn chạy được** dù thiếu dữ liệu huấn luyện.

2. *Đọc và xử lý file n-gram*

```
@staticmethod
    def _load_ngram_file(path):
        temp = []
        total_count = 0
        try:
            with open(path, 'r', encoding='utf-8') as f:
                for line in f:
                    parts = line.strip().split()
                    if len(parts) < 2: continue
                    gram = parts[0].lower()
                    try:
                        count = float(parts[1])
                        temp.append((gram, count))
                        total_count += count
                    except ValueError: continue
        except Exception:
```

```
        return {}, 0


    ngram_dict = {}
    min_score = float('inf')
    for gram, count in temp:
        log_prob = math.log10(count / total_count)
        ngram_dict[gram] = log_prob
        if log_prob < min_score: min_score = log_prob


    return ngram_dict, min_score - 1.0
```

- Hoạt động:
  - Đọc từng dòng của file n-gram
  - Tách n-gram và số lần xuất hiện
  - Tính xác suất logarit: $\log_{10}(\text{count/total})$
  - Lưu vào dictionary
- Lý do dùng log: Tránh số quá nhỏ khi nhân nhiều xác suất và tăng độ ổn định số học.

3. *Lọc ký tự hợp lệ*

```
@staticmethod
    def filter_letters(text):
        return "".join([c.lower() for c in text if c.isalpha()])
```

- Loại bỏ:
  - Số
  - Ký tự đặc biệt
  - Khoảng trắng
- Chỉ giữ lại chữ cái thường
- Phục vụ phân tích n-gram chính xác.

4. Hàm chấm điểm plaintext

```
@staticmethod
    def compute_score(plaintext):
        s = MonoalphabeticAnalyzer.filter_letters(plaintext)
        if len(s) < 4: return -999999.0


        def score(dic, min_v, n):
            sc = 0
            cnt = 0
            for i in range(len(s) - n + 1):
                gram = s[i : i+n]
                sc += dic.get(gram, min_v)
                cnt += 1
            return sc / cnt if cnt > 0 else 0


        return (score(MonoalphabeticAnalyzer._quad, MonoalphabeticAnalyzer._quad_min, 4) * 1.0 +
score(MonoalphabeticAnalyzer._tri, MonoalphabeticAnalyzer._tri_min, 3) * 0.5 + score(MonoalphabeticAnalyzer._bi,
MonoalphabeticAnalyzer._bi_min, 2) * 0.2 + score(MonoalphabeticAnalyzer._mono,
MonoalphabeticAnalyzer._mono_min, 1) * 0.1)
```

- Ý tưởng:
  - Dựa trên mô hình ngôn ngữ tiếng Anh.
  - Sử dụng n-gram có trọng số
- Ý nghĩa:
  - N-gram càng dài => phản ánh ngôn ngữ càng chính xác.
  - Điểm càng cao => plaintext càng giống tiếng Anh.

5. Áp dụng khoá ánh xạ

```
@staticmethod
    def apply_mapping(ciphertext, key_list):
        result = []
        base_a = ord('a')
        for char in ciphertext:
            if 'a' <= char <= 'z':
                result.append(key_list[ord(char) - base_a])
            elif 'A' <= char <= 'Z':
                mapped = key_list[ord(char.lower()) - base_a]
                result.append(mapped.upper())
            else:
                result.append(char)
        return "".join(result)
```

- Thay thế từng ký tự trong ciphertext theo bảng ánh xạ.

6. Khởi tạo khóa theo tần suất

```
@staticmethod
    def build_initial_mapping_by_frequency(ciphertext):
        counts = Counter([c. lower() for c in ciphertext if c.isalpha()])
        sorted_cipher = [x[0] for x in counts.most_common()]

        all_chars = set("abcdefghijklmnopqrstuvwxyz")
        missing = list(all_chars - set(sorted_cipher))
        sorted_cipher.extend(missing)

        mapping = [''] * 26
```

```
    eng_order = MonoalphabeticAnalyzer._english_frequency_order
    used_plain = set()

    for i, c_char in enumerate(sorted_cipher):
        if not c_char.isalpha():
            continue
        c_char = c_char.lower()

        idx = ord(c_char) - ord('a')

        if idx < 0 or idx >= 26:
            continue

        if i < len(eng_order):
            p_char = eng_order[i]
        else:
            remain = list(all_chars - used_plain)
            p_char = remain[0] if remain else c_char

        mapping[idx] = p_char
        used_plain.add(p_char)

    for i in range(26):
        if mapping[i] == '':
            fallback = chr(ord('a') + i)
            if fallback not in used_plain:
```

```
                mapping[i] = fallback
            else:
                for ch in all_chars:
                    if ch not in used_plain:
                        mapping[i] = ch
                        used_plain.add(ch)
                        break
                else:
                    mapping[i] = fallback

    return mapping
```

- Các bước:
  - Đếm tần suất ký tự trong ciphertext.
  - Sắp xếp theo thứ tự giảm dần.
  - Ánh xạ:
    - Chữ xuất hiện nhiều nhất => phổ biến nhất trong tiếng Anh.
  - Hoàn thiện bảng ánh xạ để đủ 26 ký tự.
- Mục đích:
  - Tạo điểm xuất phát tốt cho thuật toán tối ưu.
  - Giảm thời gian hội tụ.

7. *Thuật toán giải mã Hill-Climbing*

```
@staticmethod
    def solve(ciphertext, restarts=20, iterations=2000):
        MonoalphabeticAnalyzer.initialize_language_models()
        best_key = None
        best_score = float('-inf')
        filtered = MonoalphabeticAnalyzer.filter_letters(ciphertext)
```

```
    for _ in range(restarts):
        key = MonoalphabeticAnalyzer.build_initial_mapping_by_frequency(ciphertext)
        for _ in range(15):
            a, b = random.randint(0, 25), random.randint(0, 25)
            key[a], key[b] = key[b], key[a]

        curr_score = MonoalphabeticAnalyzer.compute_score(MonoalphabeticAnalyzer.apply_mapping(filtered, key))

        for _ in range(iterations):
            next_key = key[:]
            a, b = random.randint(0, 25), random.randint(0, 25)
            while a == b: b = random.randint(0, 25)
            next_key[a], next_key[b] = next_key[b], next_key[a]

            next_score = MonoalphabeticAnalyzer.compute_score(MonoalphabeticAnalyzer.apply_mapping(filtered,
next_key))
            if next_score > curr_score:
                curr_score = next_score
                key = next_key

        if curr_score > best_score:
            best_score = curr_score
            best_key = key[:]

    return best_key
```

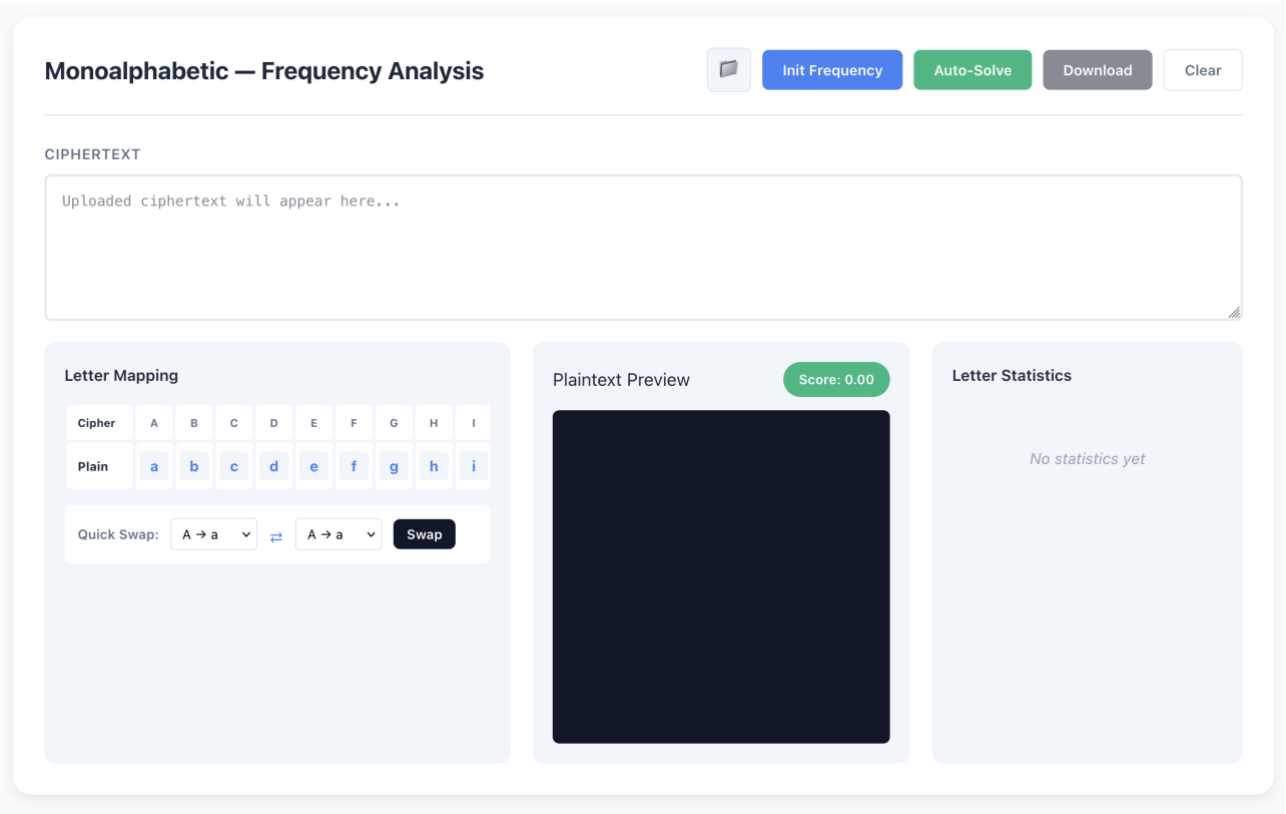
- Quy trình:

1. Khởi tạo khoá ban đầu
2. Hoán đổi ngẫu nhiên các ký tự trong khóa
3. Giải mã và chấm điểm
4. Chấp nhận khóa mới nếu điểm cao hơn
5. Lặp lại nhiều vòng và nhiều lần khởi động lại

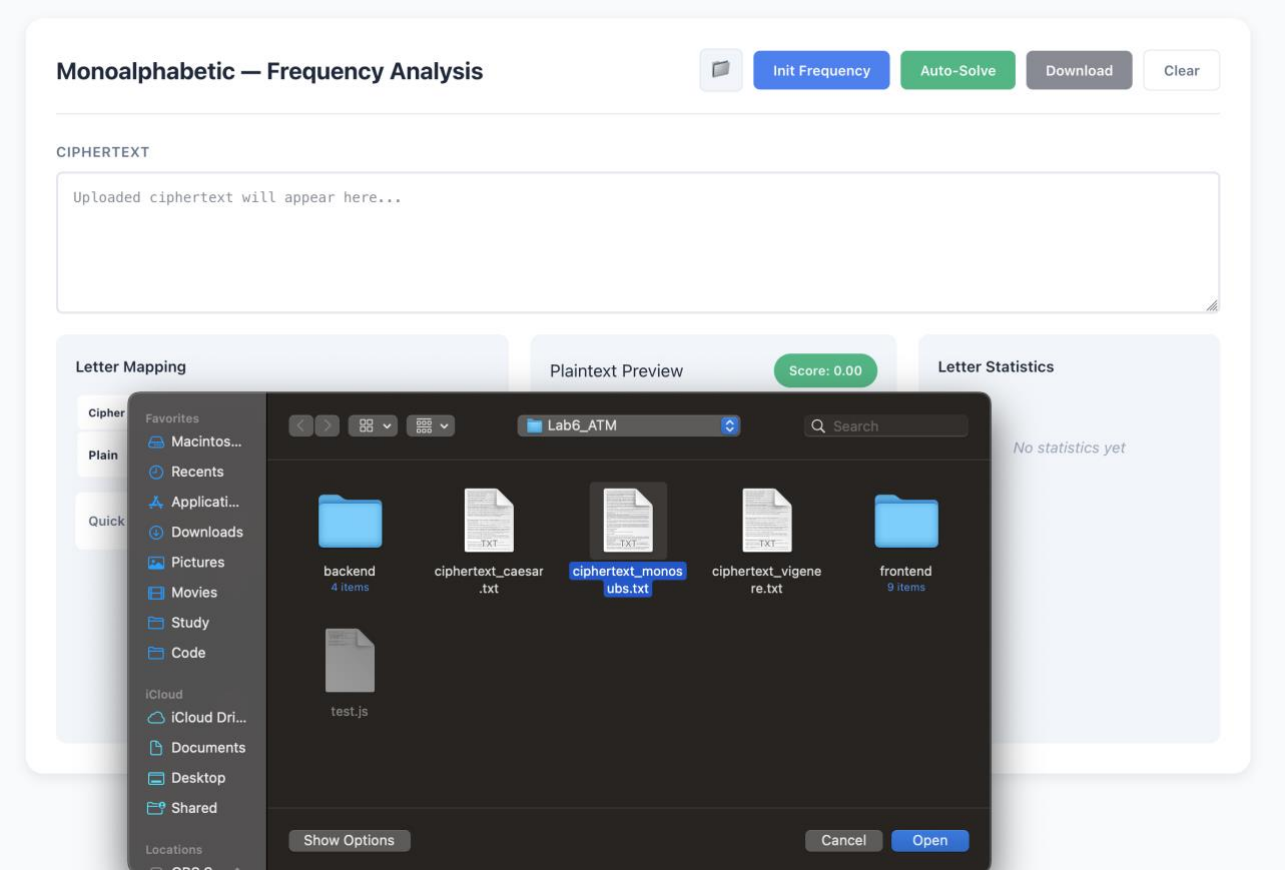
- Ý nghĩa:

- ♦ Tránh rơi vào cực trị cục bộ.
- ♦ Tăng xác suất tìm ra khóa tối ưu.

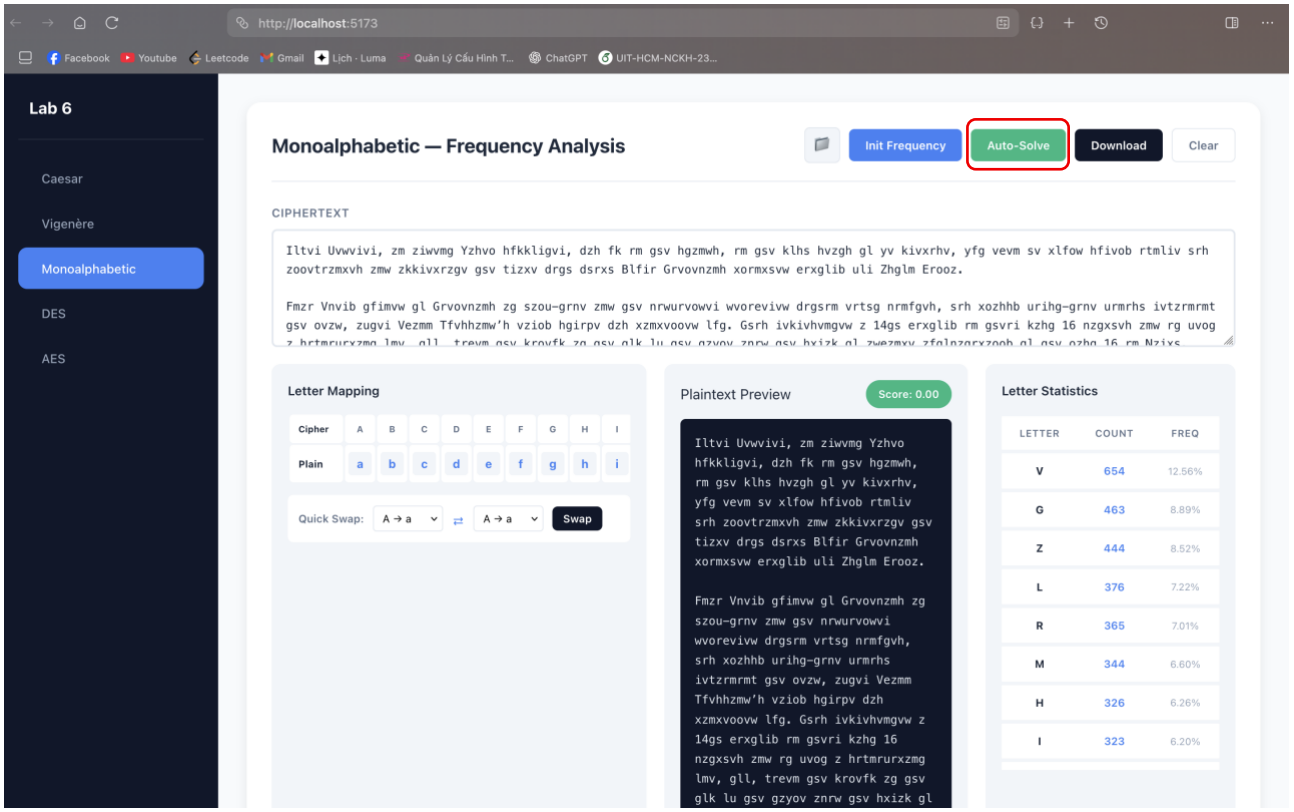
e) Kết quả



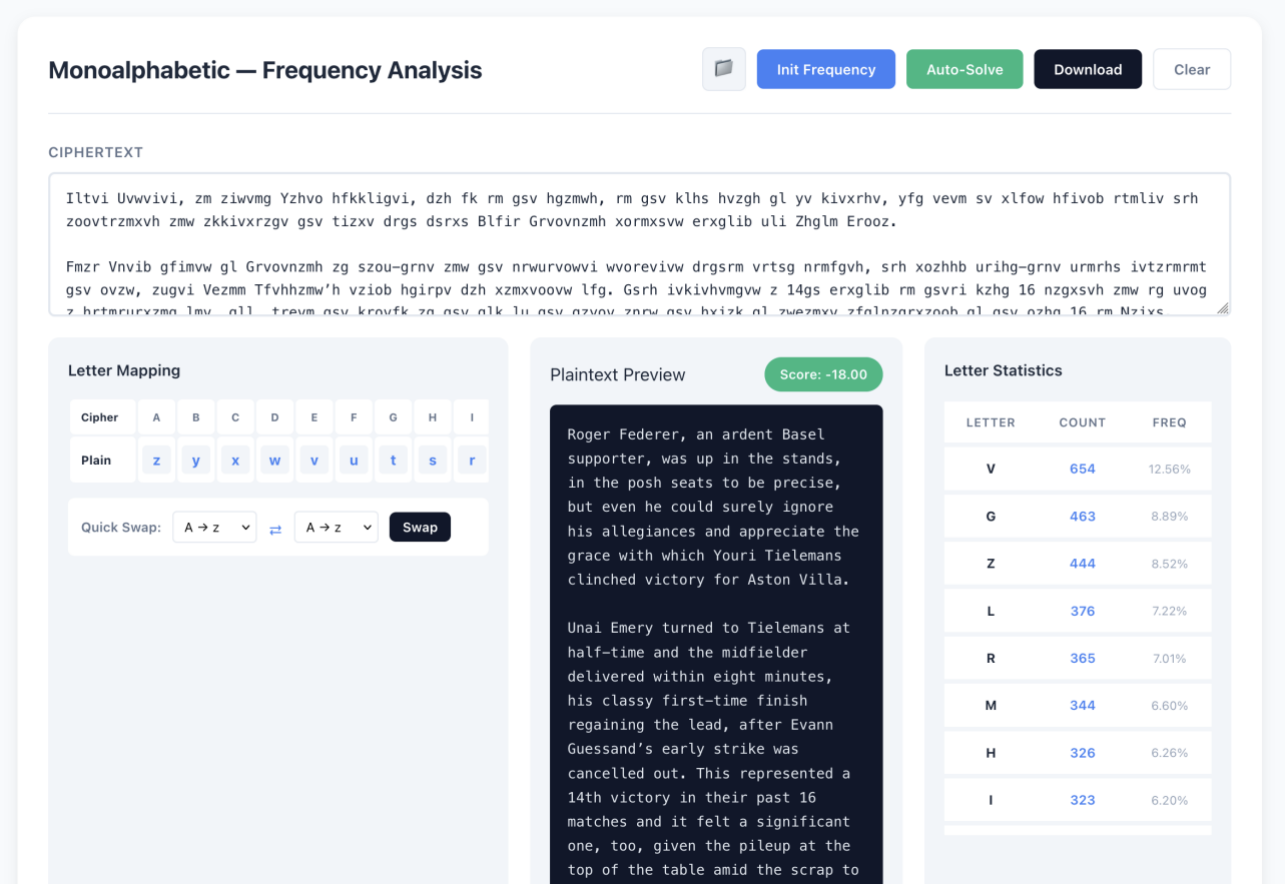
Hình 8: Giao diện của Mono-alphabetic Cipher



Hình 9: Chọn file txt ciphertext cho Mono-alphabetic



Hình 10: Nhập vào ciphertext và auto solve



Hình 11: Xuất ra kết quả và có thể download plaintext txt

3. Vigenère Cipher

a) Chức năng chính

- Chức năng Vigenère Cipher được xây dựng nhằm giải mã bản mã Vigenère khi không biết khóa.

- Hệ thống tự động:

  - Ước lượng **độ dài khóa** bằng **Index of Coincidence (IC)**
  - Tìm **giá trị khóa** bằng phân tích thống kê (chi-squared)
  - Giải mã ciphertext để thu được plaintext

- Hỗ trợ hiển thị:

  - Khoá tìm được
  - Các phép xoay vòng của khóa
  - Plaintext tương ứng

b) Giải thuật và ý tưởng chính

- Ý tưởng chính của thuật toán dựa trên các nguyên lý sau:

  1. Index of Coincidence (IC)

- ♦ IC đo mức độ trùng lặp ký tự trong văn bản.
- ♦ Văn bản tiếng Anh có IC ≈ 0.0667.
- ♦ Dùng để ước lượng độ dài khóa Vigenère.

2. Chia ciphertext thành các cột Caesar
   - ♦ Khi biết độ dài khóa k, ciphertext được chia thành k cột.
   - ♦ Mỗi cột tương đương với một bản mã Caesar.
3. Giải từng cột bằng kiểm định chi-squared
   - ♦ So sánh tần suất chữ cái của cột với tần suất tiếng Anh chuẩn.
   - ♦ Khóa Caesar có giá trị chi-squared nhỏ nhất được chọn.
4. Chuẩn hóa khóa
   - ♦ Rút gọn khóa về dạng lặp ngắn nhất.
   - ♦ Sinh các phép xoay vòng để hỗ trợ hiển thị và so sánh.

=> Cách tiếp cận này hiệu quả với ciphertext đủ dài và là phương pháp kinh điển để phá mã Vigenère.

c) Cách hoạt động

- Người dùng nhập ciphertext.
- Hệ thống thực hiện:
  - ♦ Làm sạch dữ liệu (loại bỏ ký tự không phải chữ cái).
  - ♦ Tìm độ dài khóa phù hợp nhất.
  - ♦ Tìm từng ký tự của khóa bằng phân tích thống kê.
  - ♦ Giải mã ciphertext bằng khóa tìm được.
- Kết quả trả về gồm:
  - ♦ Độ dài khóa
  - ♦ Khoá giải mã
  - ♦ Các phép xoay vòng của khóa
  - ♦ Bản rõ plaintext

d) Chi tiết chương trình

1. Tính Index of Coincidence

```
def calculate_ic(text: str) -> float:
    """Calculate Index of Coincidence for a text"""
    if not text or len(text) <= 1:
        return 0.0

    n = len(text)
    counts = Counter(text)

    numerator = sum(count * (count - 1) for count in counts.values())
    denominator = n * (n - 1)

    return numerator / denominator if denominator > 0 else 0.0
```

- Công thức:

$$IC = \frac{\sum \text{fi}(\text{fi} - 1)}{N(N-1)}$$

- Giá trị IC càng gần 0.0667 => càng giống tiếng Anh.

2. Xác định độ dài khoá

```
def find_key_length(ciphertext: str, max_len=20) -> int:
    """Find most likely key length using Index of Coincidence"""
    clean = clean_text(ciphertext)

    if len(clean) < 100:
        return 1

    best_len = 1
    best_avg_ic = 0.0

    for key_len in range(1, min(max_len + 1, len(clean) // 20)):
        ic_sum = 0.0

        for offset in range(key_len):
```

```
        column = clean[offset:: key_len]
        if len(column) > 1:
            ic_sum += calculate_ic(column)

    avg_ic = ic_sum / key_len

    if abs(avg_ic - ENGLISH_IC) < abs(best_avg_ic - ENGLISH_IC):
        best_avg_ic = avg_ic
        best_len = key_len

return best_len
```

- Thử các độ dài khóa từ 1 đến max_key_len.
- Tính IC trung bình cho từng độ dài.
- Chọn độ dài có IC gần tiếng Anh nhất.

3. *Kiểm định chi-squared*

```
def chi_squared(observed_counts: list, text_length: int) -> float:

    """Calculate chi-squared statistic against English frequency"""

    chi2 = 0.0



    for i in range(26):

        expected = ENGLISH_FREQ[i] * text_length

        observed = observed_counts[i]



        if expected > 0:

            chi2 += ((observed - expected) ** 2) / expected



    return chi2
```

- Đo độ lệch giữa tần suất quan sát và tần suất tiếng Anh chuẩn.
- Giá trị chi-squared càng nhỏ => càng phù hợp.

4. Giải Caesar cho từng cột

```python
def solve_caesar_column(column: str) -> str:
    """Find best Caesar shift for a single column using chi-squared"""
    if not column:
        return 'A'

    n = len(column)
    best_shift = 0
    min_chi2 = float('inf')

    for shift in range(26):
        observed = [0] * 26

        for char in column:
            decrypted_idx = (ord(char) - ord('A') - shift) % 26
            observed[decrypted_idx] += 1

        chi2 = chi_squared(observed, n)

        if chi2 < min_chi2:
            min_chi2 = chi2
            best_shift = shift

    return chr(ord('A') + best_shift)
```

- Thử tất cả 26 phép dịch.
- Chọn khóa Caesar có chi-squared nhỏ nhất.
- Trả về ký tự khóa tương ứng.

5. Tìm khóa Vigenère

```
def find_key(ciphertext: str, key_len: int) -> str:
    """Find the key by solving each Caesar-shifted column"""
    clean = clean_text(ciphertext)
    key_chars = []

    for offset in range(key_len):
        column = clean[offset:: key_len]
        key_char = solve_caesar_column(column)
        key_chars.append(key_char)

    return "".join(key_chars)
```

- Chia ciphertext thành key_len cột.
- Giải từng cột như Caesar.
- Ghép các ký tự để tạo khóa hoàn chỉnh.

6. Giải mã Vigenère

```
def decrypt_vigenere(ciphertext: str, key: str) -> str:
    """Decrypt Vigenere cipher, preserving non-alphabetic characters"""
    if not key:
        return ciphertext

    result = []
    key_upper = key.upper()
    key_index = 0

    for char in ciphertext:
        if char.isalpha():
```

```
        is_upper = char.isupper()

        char_upper = char.upper()



        c_idx = ord(char_upper) - ord('A')

        k_idx = ord(key_upper[key_index % len(key_upper)]) - ord('A')



        p_idx = (c_idx - k_idx) % 26



        decrypted = chr(p_idx + ord('A'))



        if not is_upper:

            decrypted = decrypted.lower()



        result.append(decrypted)

        key_index += 1

    else:

        result.append(char)



return "".join(result)
```

- Công thức giải mã:

$$P = (C - K) \ mod \ 26$$

- Giữ nguyên:

- Chữ hoa / chữ thường.
- Ký tự đặc biệt.

7. Chuẩn hoá và xoay khoá

```python
def normalize_key(key: str) -> str:
    """Reduce key to shortest repeating pattern"""
    n = len(key)

    for length in range(1, n + 1):
        if n % length == 0:
            candidate = key[: length]
            if candidate * (n // length) == key:
                return candidate

    return key

def get_canonical_key(key:  str) -> str:
    """Return the lexicographically smallest rotation of the key"""
    key = key.lower()
    n = len(key)

    rotations = [key[i:] + key[:i] for i in range(n)]

    return min(rotations)

def get_all_rotations(key:  str) -> list:
    """Get all rotations of a key"""
    key = key.lower()
    n = len(key)
    return [key[i: ] + key[:i] for i in range(n)]
```

- Rút gọn khóa về mẫu lặp ngắn nhất.
- Sinh các phép xoay vòng để so sánh và hiển thị.
- Lấy dạng khóa chuẩn (canonical key).

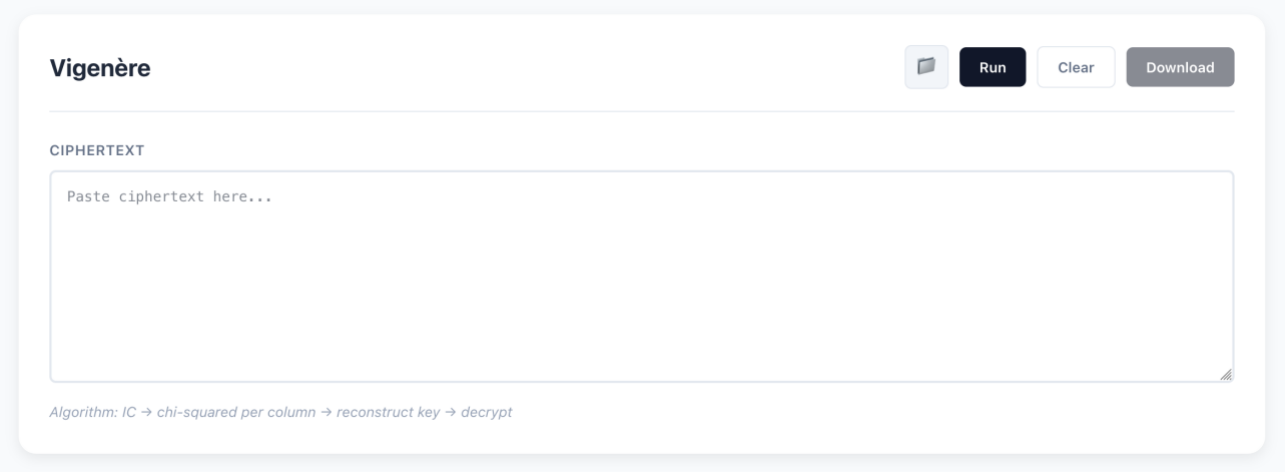
8. Hàm giải mã Vigenère

```python
def solve_vigenere(ciphertext:  str, max_key_len=20):

    """Main solver function"""

    key_len = find_key_length(ciphertext, max_key_len)

    raw_key = find_key(ciphertext, key_len)
```

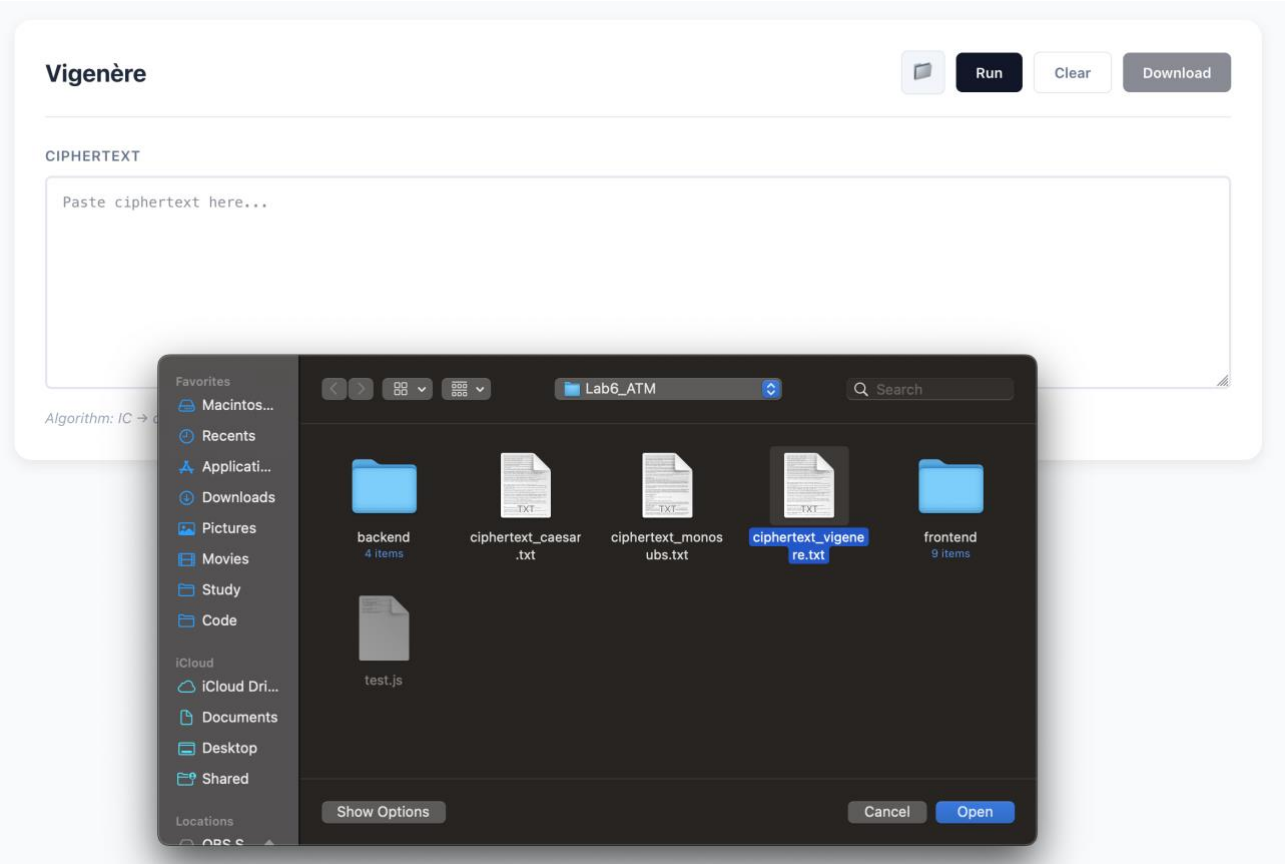
```
key = normalize_key(raw_key)



all_rotations = get_all_rotations(key)

canonical = get_canonical_key(key)



plaintext = decrypt_vigenere(ciphertext, key)



return {

    "keyLen": len(key),

    "key": key. lower(),

    "displayKey": key. lower(),

    "canonicalKey": canonical,

    "allRotations": all_rotations,

    "plaintext":  plaintext,

    "candidates": []

}
```

- Gọi toàn bộ các bước:
  - Tìm độ dài khóa
  - Tìm khoá
  - Chuẩn hóa
  - Giải mã
- Trả kết quả dưới dạng JSON cho frontend.

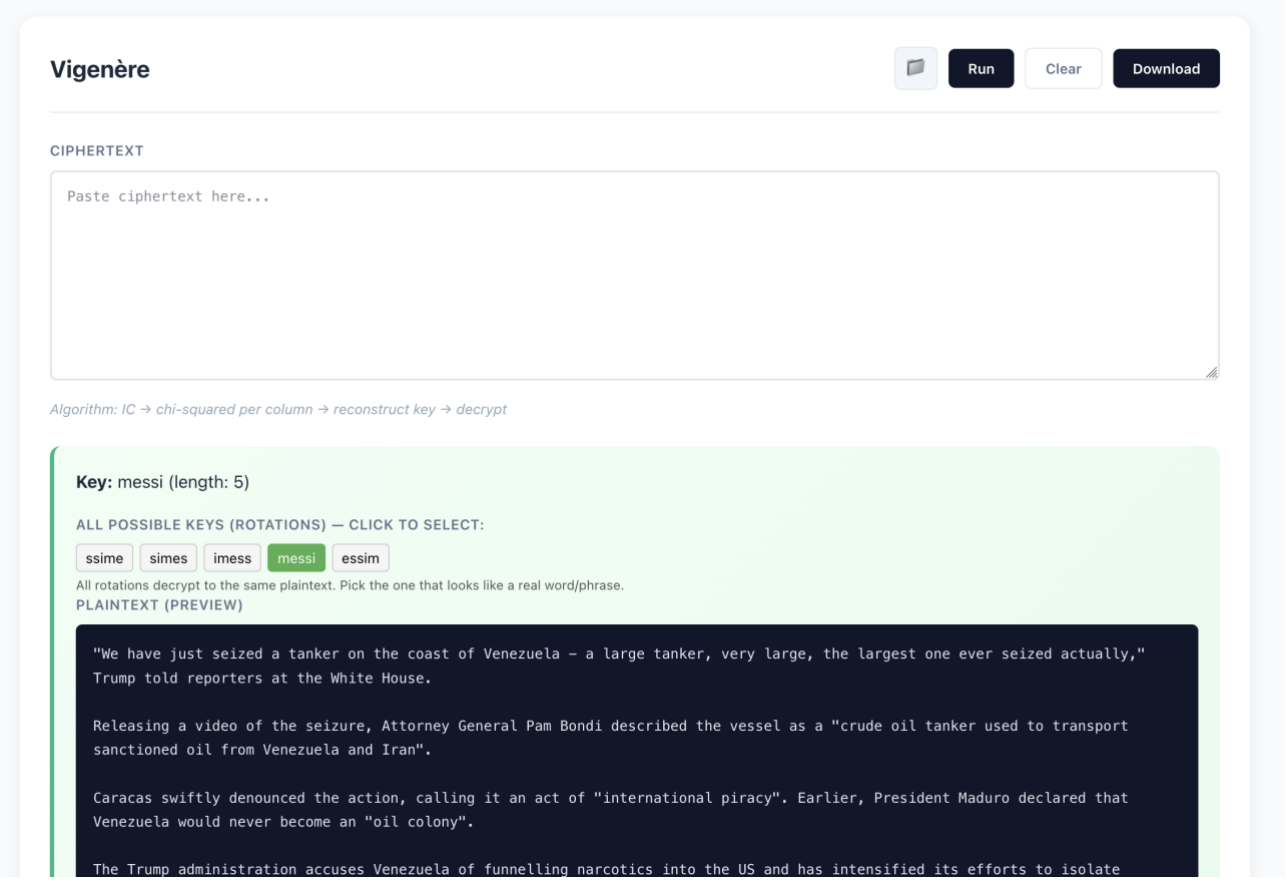
e) Kết quả



Hình 12: Giao diện của Vigenère Cipher



Hình 13: Chọn file txt ciphertext cho Vigenère



Hình 14: Xuất ra kết quả và có thể download file txt plaintext

4. DES – Data Encryption Standard

a) Chức năng chính

- Chức năng DES (Data Encryption Standard) được xây dựng nhằm:
  - ♦ Thực hiện mã hóa và giải mã dữ liệu theo chuẩn DES.
  - ♦ Hỗ trợ các chế độ hoạt động:
    - ○ ECB (Electronic Codebook)
    - ○ CBC (Cipher Block Chaining)
- Hệ thống đảm bảo:
  - ♦ Xử lý đúng kích thước khối 64-bit
  - ♦ Sinh khóa con cho từng vòng
  - ♦ Thực hiện padding và unpadding dữ liệu
- Phù hợp cho việc minh họa nguyên lý hoạt động của DES trong môn An toàn mạng máy tính.

b) Giải thuật và ý tưởng chính

- Thuật toán DES được cài đặt dựa trên các nguyên lý chuẩn sau:

1. Mã khối (Block Cipher)
   - DES xử lý dữ liệu theo từng khối 64-bit (8 byte)
2. Mạng Feistel
   - Dữ liệu được chia thành hai nửa trái (L) và phải (R).
   - Thực hiện 16 vòng Feistel, mỗi vòng sử dụng một khóa con khác nhau.
3. Sinh khóa con (Key Schedule)
   - Khóa ban đầu 64-bit được nén còn 56-bit.
   - Qua dịch vòng và hoán vị để sinh ra 16 khóa con 48-bit.
4. Hàm Feistel (f-function)
   - Mở rộng R từ 32 → 48 bit.
   - XOR với khóa con.
   - Thay thế bằng S-box.
   - Hoán vị P-box.
5. Chế độ hoạt động
   - ECB: mỗi khối mã hóa độc lập.
   - CBC: mỗi khối phụ thuộc vào khối trước thông qua IV.

c) Cách hoạt động

- Người dùng cung cấp:
  - Plaintext
  - Khóa bí mật (8 byte)
  - Chế độ hoạt động (ECB hoặc CBC)
  - IV (nếu dùng CBC)

- Hệ thống:

1. Padding dữ liệu về bội số của 8 byte
2. Sinh khóa con
3. Mã hóa hoặc giải mã từng khối
4. Ghép kết quả và trả về ciphertext hoặc plaintext

d) Chi tiết chương trình

1. Các bảng hoán vị và S-box

```
IP = [
    58,50,42,34,26,18,10,2, 60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6, 64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1, 59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5, 63,55,47,39,31,23,15,7
]

IP_INV = [
    40,8,48,16,56,24,64,32, 39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30, 37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28, 35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26, 33,1,41,9,49,17,57,25
]

E = [
    32,1,2,3,4,5, 4,5,6,7,8,9,
    8,9,10,11,12,13, 12,13,14,15,16,17,
    16,17,18,19,20,21, 20,21,22,23,24,25,
    24,25,26,27,28,29, 28,29,30,31,32,1
]

P = [
    16,7,20,21, 29,12,28,17,
    1,15,23,26, 5,18,31,10,
    2,8,24,14, 32,27,3,9,
```

```
    19,13,30,6, 22,11,4,25
]

PC1 = [
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,
    19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,
    21,13,5,28,20,12,4
]

PC2 = [
    14,17,11,24,1,5, 3,28,15,6,21,10,
    23,19,12,4,26,8, 16,7,27,20,13,2,
    41,52,31,37,47,55, 30,40,51,45,33,48,
    44,49,39,56,34,53, 46,42,50,36,29,32
]

SHIFTS = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

S_BOX = [
    [[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
     [0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
```

```
 [4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
 [15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]],
[[15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],
 [3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],
 [0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],
 [13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9]],
[[10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],
 [13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],
 [13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],
 [1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12]],
[[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],
 [13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
 [10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
 [3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]],
[[2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
 [14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
 [4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
 [11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]],
[[12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],
 [10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],
 [9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],
 [4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13]],
[[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
 [13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],
 [1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
 [6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]],
```

```
    [[13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],
     [1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],
     [7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],
     [2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]]
]
```

- Là các bảng chuẩn của DES:
  - IP / IP^{-1}: hoán vị đầu và cuối
  - E: mở rộng 32 → 48 bit
  - P: hoán vị sau S-box
  - PC1, PC2: sinh khóa con
  - S_BOX: 8 hộp thay thế phi tuyến
- Đảm bảo tính khuếch tán và nhiễu loạn (diffusion & confusion).

2. *Các hàm xử lý bit*

```
def bytes_to_bits(b): return [(b[i] >> (7-j)) & 1 for i in range(len(b)) for j in range(8)]
def bits_to_bytes(bits):
    out = bytearray(len(bits)//8)
    for i in range(len(out)):
        for j in range(8):
            out[i] |= bits[i*8+j] << (7-j)
    return bytes(out)
def permute(bits, table): return [bits[i-1] for i in table]
def xor(a,b): return [i^j for i,j in zip(a,b)]
def rotl(b,n): return b[n:]+b[:n]
```

- Chuyển đổi giữa byte và bit.
- Thực hiện hoán vị, XOR và dịch vòng.
- Là các phép toán cơ bản của DES.

3. Sinh khoá con

```
def subkeys(key):
    k = permute(bytes_to_bits(key), PC1)
    C,D = k[:28],k[28:]
    keys=[]
    for s in SHIFTS:
        C,D = rotl(C,s),rotl(D,s)
        keys.append(permute(C+D,PC2))
    return keys
```

- Thực hiện:
  - Hoán vị PC1
  - Chia khóa thành C và D (28-bit)
  - Dịch vòng theo bảng SHIFTS
  - Hoán vị PC2 để sinh 16 khóa con
- Mỗi vòng sử dụng một khóa con khác nhau.

4. Hàm Feistel

```
def feistel(R,K):
    x = xor(permute(R,E),K)
    out=[]
    for i in range(8):
        b=x[i*6:(i+1)*6]
        r=(b[0]<<1)|b[5]
        c=(b[1]<<3)|(b[2]<<2)|(b[3]<<1)|b[4]
        v=S_BOX[i][r][c]
        out += [(v>>(3-j))&1 for j in range(4)]
    return permute(out,P)
```

- Các bước:

- Mở rộng R từ 32 → 48 bit (E)
- XOR với khoá con
- Chia thành 8 khối 6-bit
- Tra cứu S-box → 32 bit
- Hoán vị P

5. Mã hoá/giải mã một khối

```
def des_block(block, keys, enc=True):
    bits=permute(bytes_to_bits(block),IP)
    L,R=bits[:32],bits[32:]
    for i in range(16):
        k=keys[i] if enc else keys[15-i]
        L,R=R,xor(L,feistel(R,k))
    return bits_to_bytes(permute(R+L,IP_INV))
```

- Áp dụng IP
- Thực hiện 16 vòng Feistel
- Đổi chỗ L–R
- Áp dụng IP^{-1}
- Dùng chung cho cả mã hóa và giải mã (đảo thứ tự khóa).

6. Padding dữ liệu

```
def pad(d):
    p=8-len(d)%8
    return d+bytes([p])*p

def unpad(d): return d[:-d[-1]]
```

- Sử dụng PKCS#5/PKCS#7 padding.
- Đảm bảo dữ liệu có độ dài bội số của 8 byte.

7. Mã hoá DES

```
def encrypt(plaintext,key,mode,iv=None):
```

```
    keys=subkeys(key)
    data=pad(plaintext)
    out=b""
    if mode=="ECB":
        for i in range(0,len(data),8):
            out+=des_block(data[i:i+8],keys,True)
        return out,None
    if mode=="CBC":
        iv=iv or os.urandom(8)
        prev=iv
        for i in range(0,len(data),8):
            blk=bytes(a^b for a,b in zip(data[i:i+8],prev))
            enc=des_block(blk,keys,True)
            out+=enc
            prev=enc
        return out,iv
    raise ValueError("Unsupported mode")
```

- ECB: Mã hóa từng block độc lập.
- CBC:
  - ♦ XOR plaintext với block trước (hoặc IV).
  - ♦ IV được sinh ngẫu nhiên nếu không cung cấp.

8. *Giải mã DES*

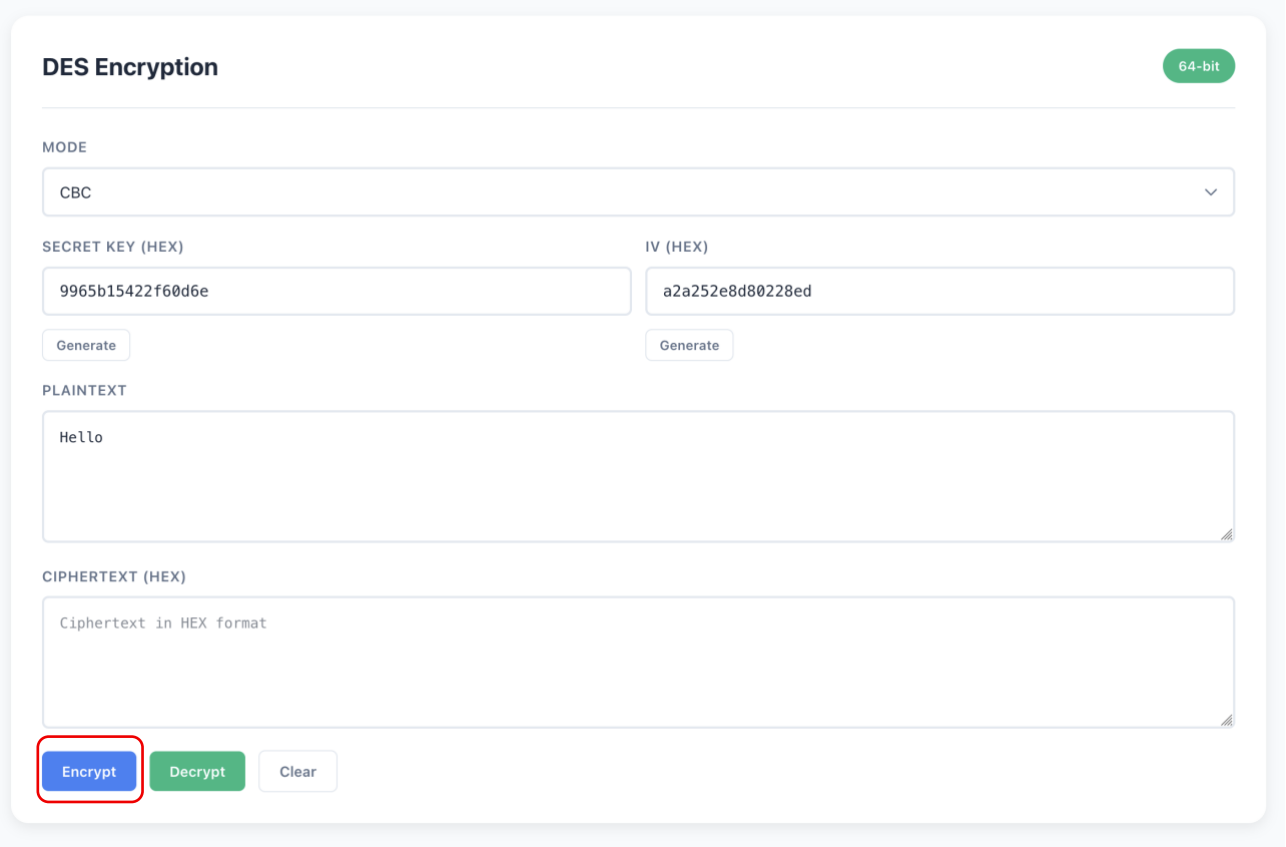
```
def decrypt(ciphertext,key,mode,iv=None):
    keys=subkeys(key)
    out=b""
    if mode=="ECB":
        for i in range(0,len(ciphertext),8):
```

```
        out+=des_block(ciphertext[i:i+8],keys,False)
    return unpad(out)
if mode=="CBC":
    if iv is None: raise ValueError("IV required")
    prev=iv
    for i in range(0,len(ciphertext),8):
        dec=des_block(ciphertext[i:i+8],keys,False)
        out+=bytes(a^b for a,b in zip(dec,prev))
        prev=ciphertext[i:i+8]
    return unpad(out)
```

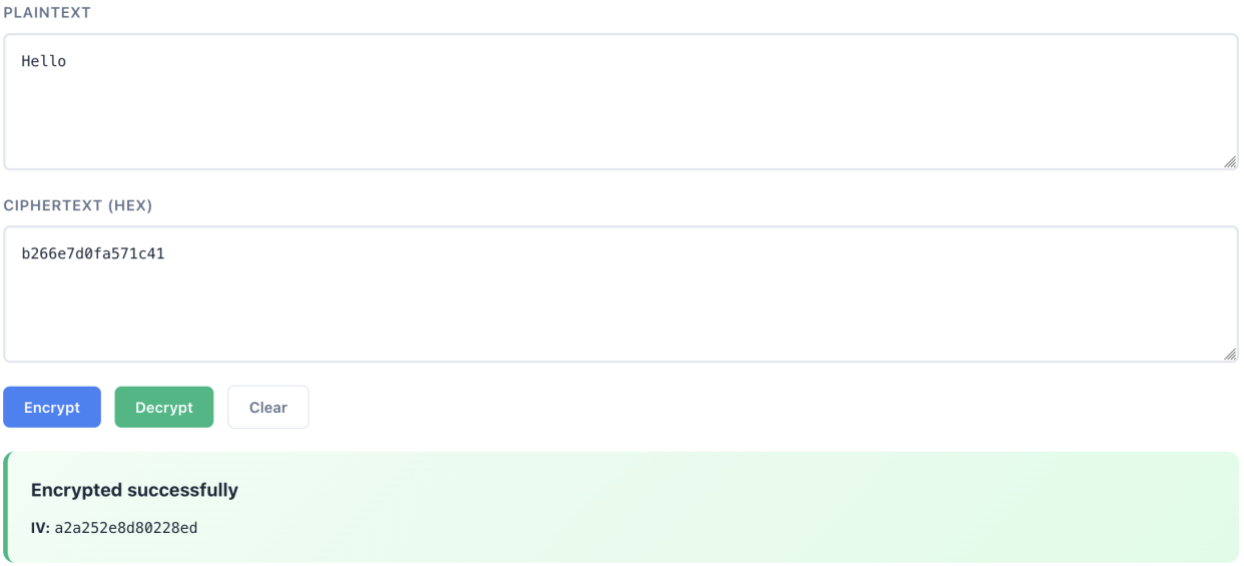
- ECB: Giải mã từng block.
- CBC: Giải mã rồi XOR với block trước.
- Sau cùng thực hiện unpadding.

e) Kết quả

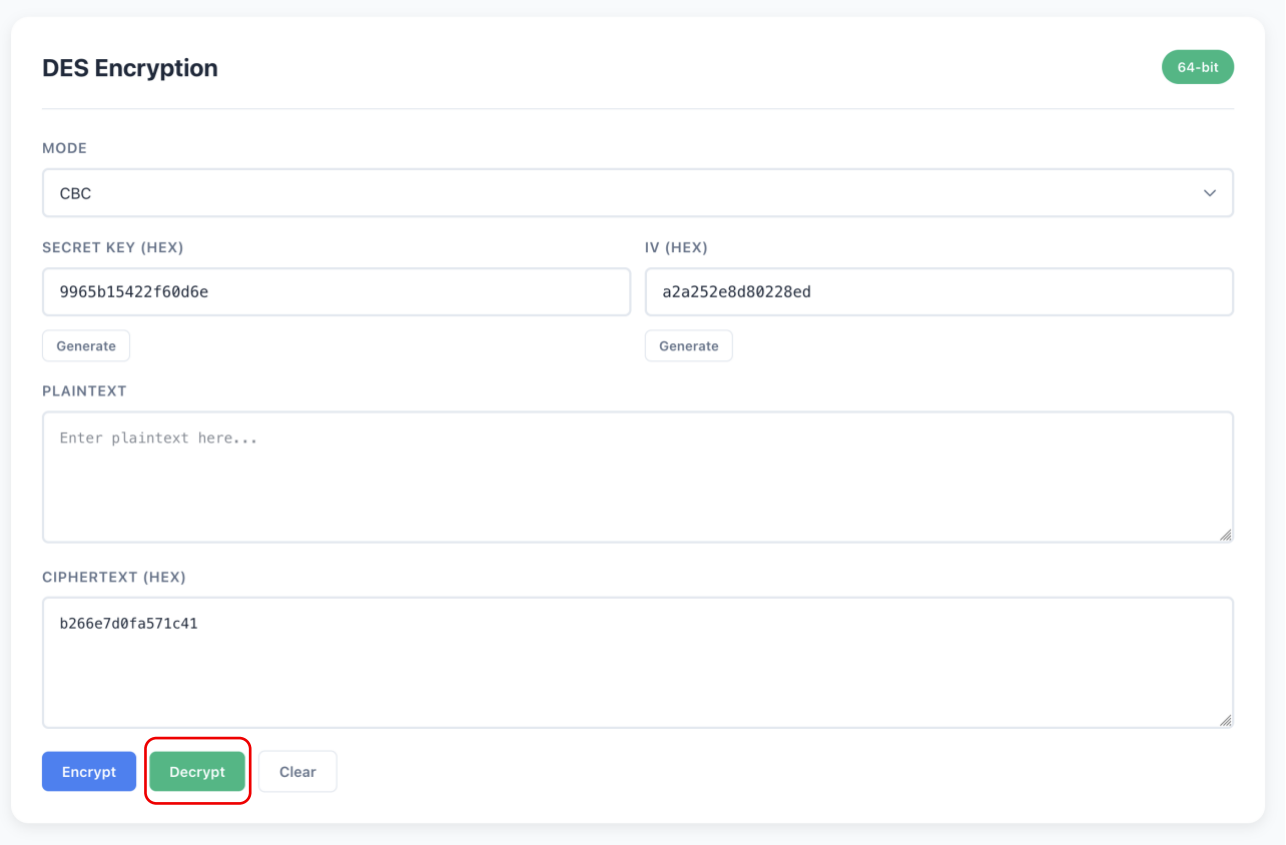
- Kết quả của chế độ CBC cho phép tải về file plaintext.txt.



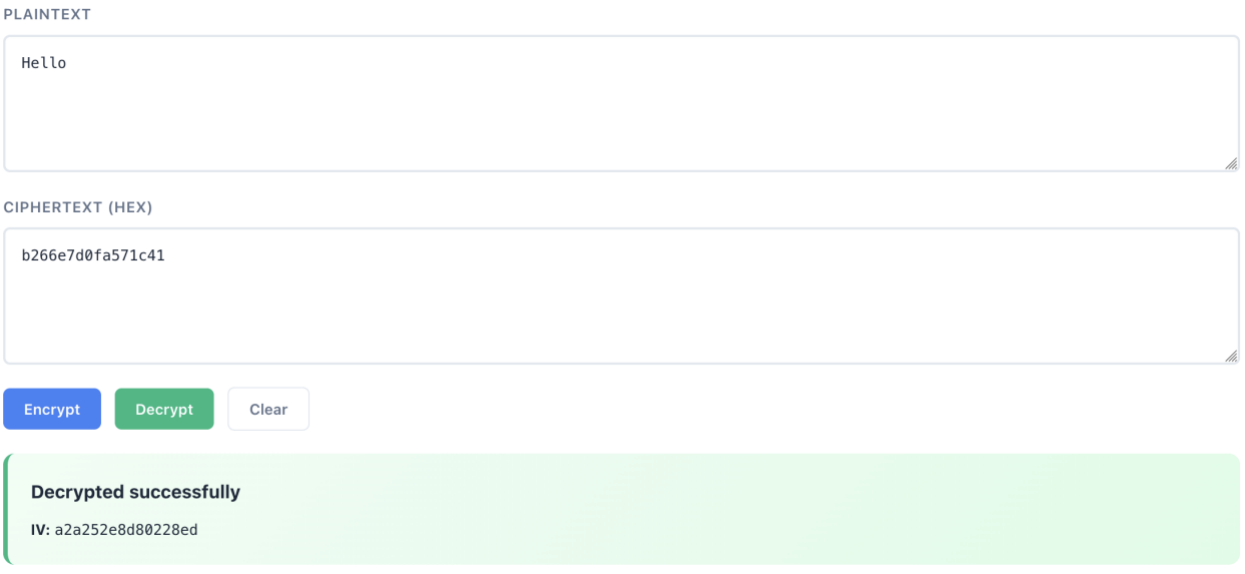
Hình 15: Giao diện chức năng DES Encryption ở chế độ CBC



Hình 16: Kết quả mã hóa DES, hiển thị ciphertext dạng HEX và IV.

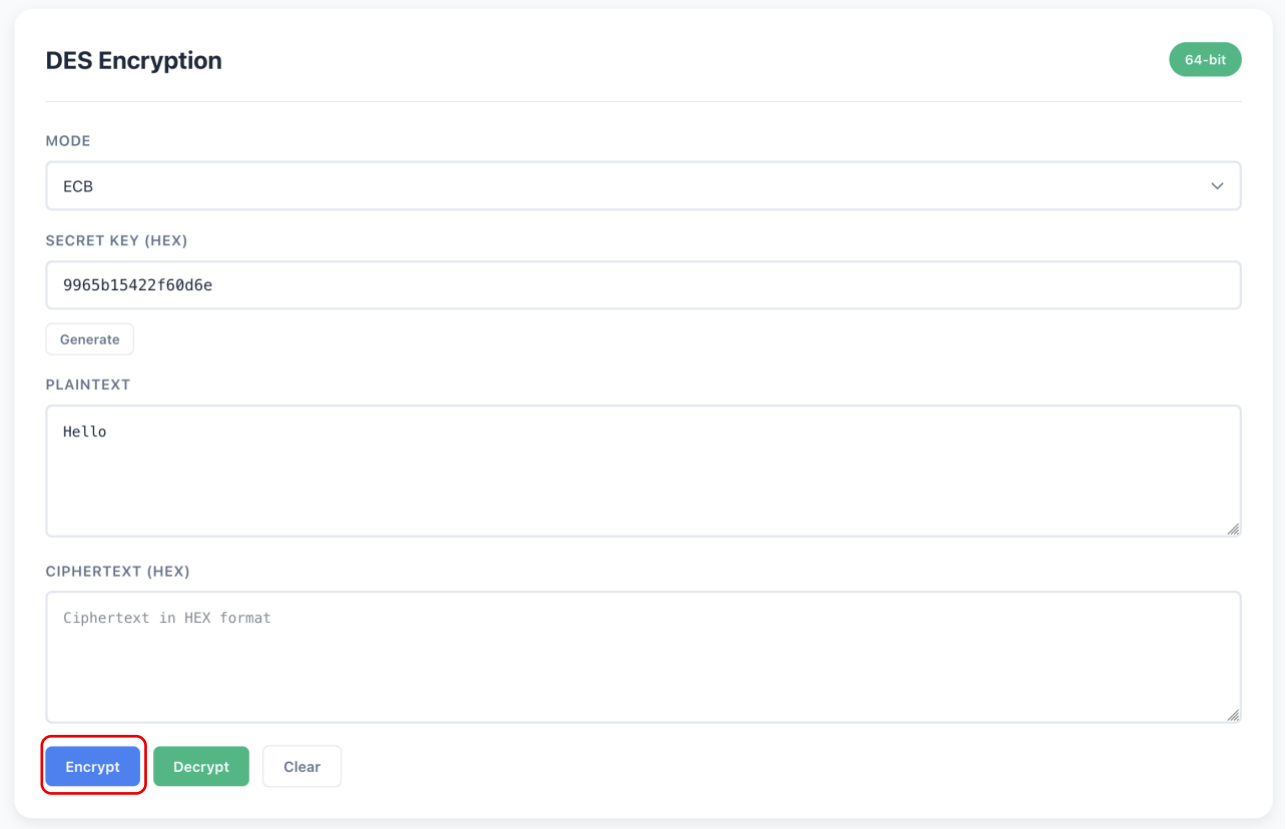


Hình 17: Ciphertext sau mã hóa, sẵn sàng cho bước giải mã

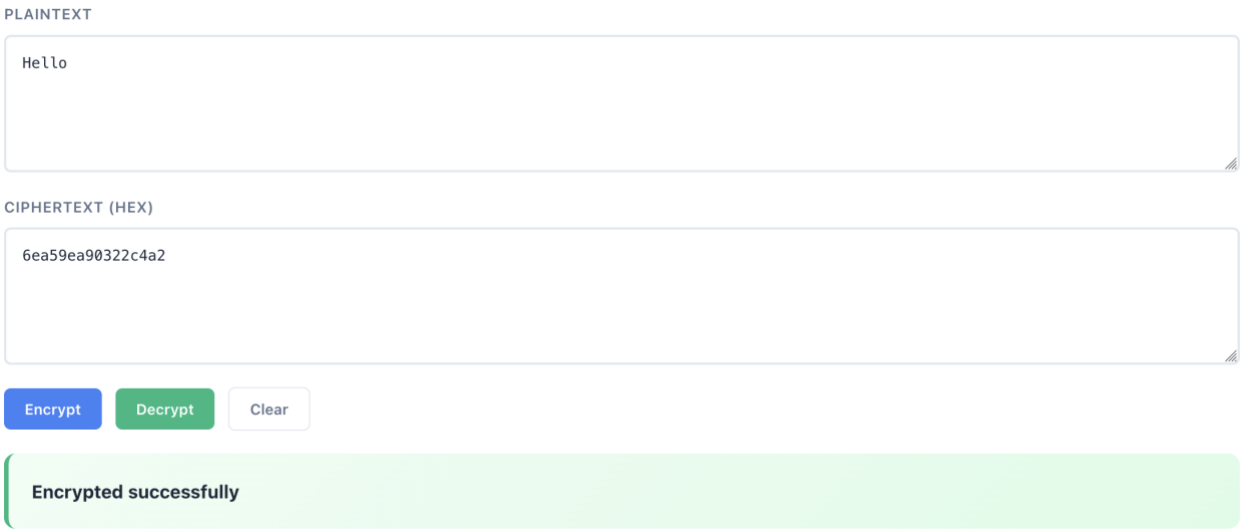


Hình 18: Kết quả giải mã DES, khôi phục đúng plaintext ban đầu.

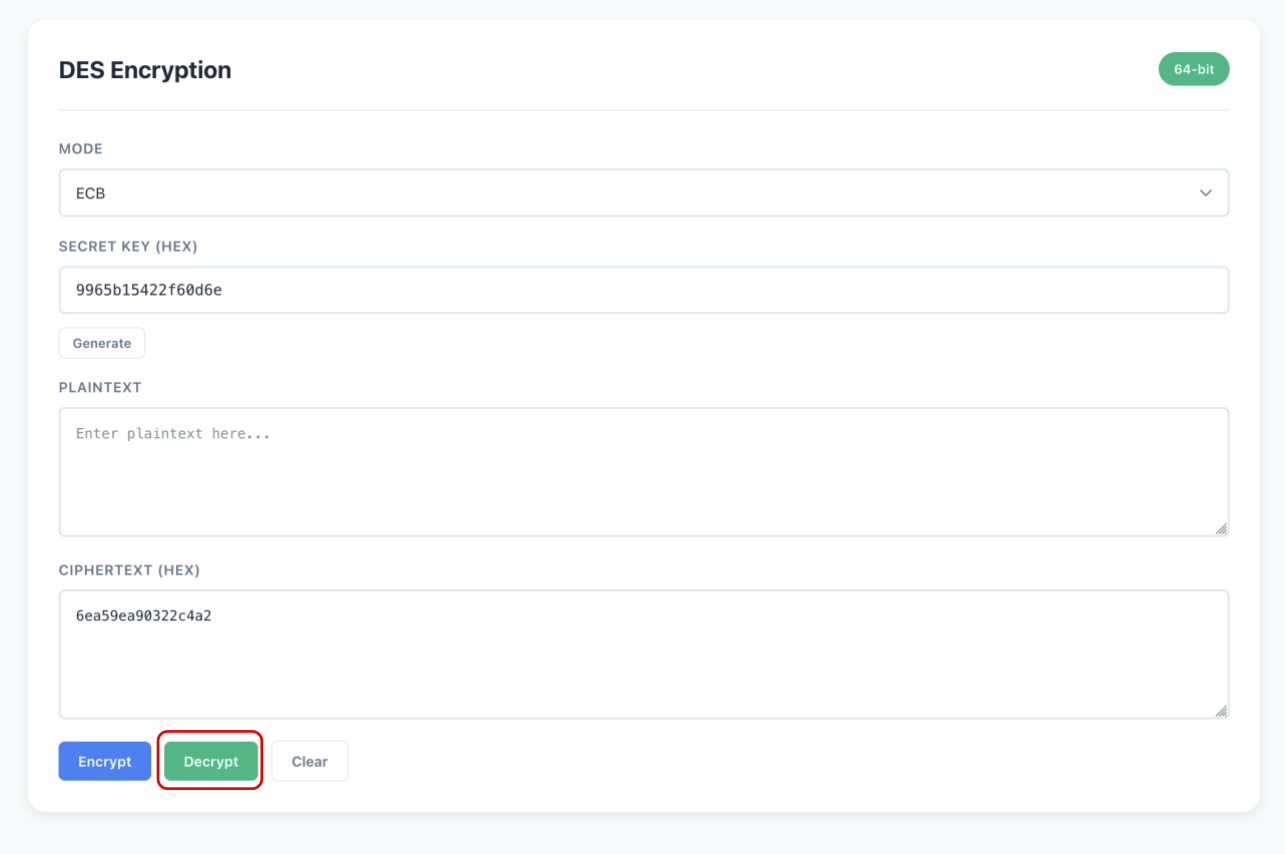
- Kết quả của chế độ ECB cho phép tải về file plaintext.txt.



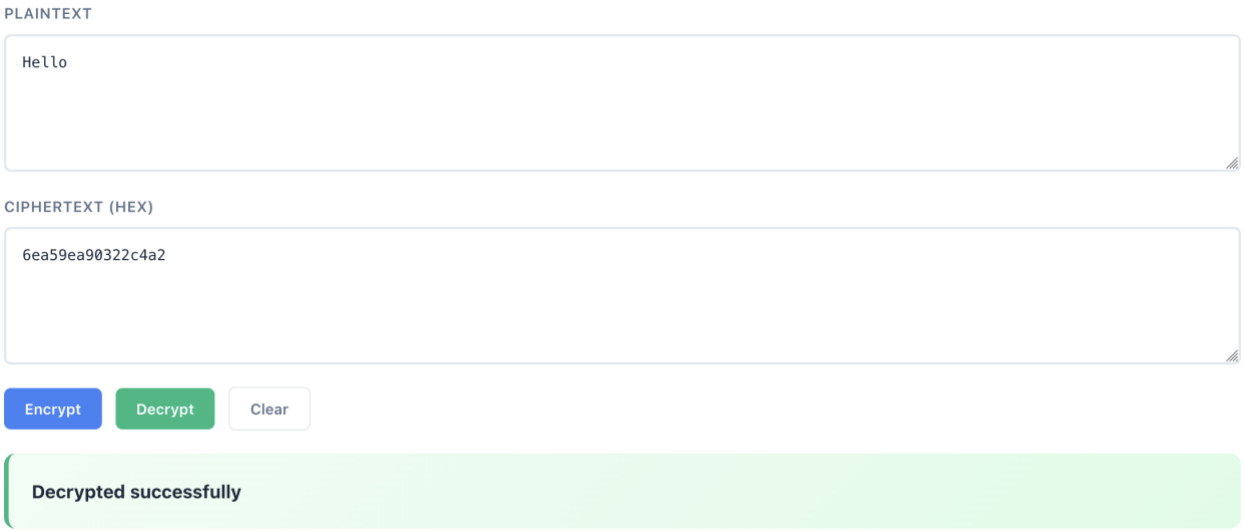
Hình 19: Giao diện chức năng DES Encryption ở chế độ ECB.



Hình 20: Kết quả mã hóa DES (ECB), hiển thị ciphertext dạng HEX.



Hình 21: Ciphertext sau mã hóa ở chế độ ECB, sẵn sàng cho bước giải mã.



Hình 22: Kết quả giải mã DES (ECB), khôi phục đúng plaintext ban đầu.

5. AES – Advanced Encryption Standard

a) Chức năng chính

- Chức năng AES (Advanced Encryption Standard) được xây dựng nhằm mã hóa và giải mã dữ liệu đối xứng theo chuẩn AES.

- Hệ thống hỗ trợ:
  - **AES-128, AES-192, AES-256** (tương ứng khóa 16, 24, 32 byte)
  - Các **chế độ hoạt động**:
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
- Thực hiện đầy đủ các bước:
  - Sinh khóa mở rộng (Key Expansion)
  - Các vòng biến đổi AES
  - Padding dữ liệu
  - Xử lý IV trong chế độ CBC

b) Giải thuật và ý tưởng chính

- AES là thuật toán mã khối đối xứng, hoạt động trên khối dữ liệu 128-bit, dựa trên mạng thay thế – hoán vị (Substitution–Permutation Network).
- Ý tưởng chính của AES gồm:
  1. Key Expansion: Từ khóa ban đầu sinh ra các **round key** cho từng vòng.
  2. Các phép biến đổi chính
     - **SubBytes:** thay thế byte thông qua S-box
     - **ShiftRows:** dịch vòng các hàng trong state
     - **MixColumns:** trộn cột trong trường hữu hạn GF(2^8)
     - **AddRoundKey:** XOR với round key
  3. Số vòng mã hóa
     - AES-128: 10 vòng
     - AES-192: 12 vòng
     - AES-256: 14 vòng
  4. Chế độ hoạt động
     - **ECB:** mã hóa từng block độc lập
     - **CBC:** mỗi block phụ thuộc block trước thông qua IV

c) Cách hoạt động

- Người dùng cung cấp:

- Plaintext / Ciphertext
- Khóa bí mật (HEX)
- Chọn chế độ hoạt động (ECB/CBC)

- Hệ thống:

- Thực hiện padding dữ liệu
- Sinh round key
- Mã hóa hoặc giải mã từng block 16 byte

- Kết quả:

- Ciphertext (kèm IV nếu dùng CBC)
- Hoặc plaintext sau khi giải mã

d) Chi tiết chương trình

1. Hằng số và S-box

```
S_BOX = [
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
```

```
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
]

INV_S_BOX = [0] * 256
for i, v in enumerate(S_BOX):
    INV_S_BOX[v] = i

RCON = [0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36]
```

- S_BOX: bảng thay thế phi tuyến dùng trong SubBytes.
- INV_S_BOX: bảng nghịch đảo dùng cho giải mã.
- RCON: hằng số vòng dùng trong mở rộng khóa.
- Đảm bảo confusion trong AES.

2. *Phép toán trong trường GF(2^8)*

```
def xtime(a):
    return ((a << 1) ^ 0x1B) & 0xFF if a & 0x80 else (a << 1)

def gmul(a, b):
    r = 0
    for _ in range(8):
        if b & 1:
            r ^= a
        a = xtime(a)
        b >>= 1
    return r & 0xFF
```

- Thực hiện nhân đa thức trong trường hữu hạn GF(2^8).
- Được sử dụng trong MixColumns và Inverse MixColumns.

3. Các phép biến đổi AES

```
def add_round_key(state, round_key):
    for i in range(16):
        state[i] ^= round_key[i]


def sub_bytes(state, inv=False):
    box = INV_S_BOX if inv else S_BOX
    for i in range(16):
        state[i] = box[state[i]]


def shift_rows(state, inv=False):
    if inv:
        state[1], state[5], state[9], state[13] = state[13], state[1], state[5], state[9]
        state[2], state[6], state[10], state[14] = state[10], state[14], state[2], state[6]
        state[3], state[7], state[11], state[15] = state[7], state[11], state[15], state[3]
    else:
        state[1], state[5], state[9], state[13] = state[5], state[9], state[13], state[1]
        state[2], state[6], state[10], state[14] = state[10], state[14], state[2], state[6]
        state[3], state[7], state[11], state[15] = state[15], state[3], state[7], state[11]


def mix_columns(state, inv=False):
    for i in range(4):
        col = [state[i * 4 + j] for j in range(4)]

        if inv:
            state[i * 4] = gmul(col[0], 14) ^ gmul(col[1], 11) ^ gmul(col[2], 13) ^ gmul(col[3], 9)
```

```
            state[i * 4 + 1] = gmul(col[0], 9) ^ gmul(col[1], 14) ^ gmul(col[2], 11) ^ gmul(col[3], 13)
            state[i * 4 + 2] = gmul(col[0], 13) ^ gmul(col[1], 9) ^ gmul(col[2], 14) ^ gmul(col[3], 11)
            state[i * 4 + 3] = gmul(col[0], 11) ^ gmul(col[1], 13) ^ gmul(col[2], 9) ^ gmul(col[3], 14)
        else:
            state[i * 4] = gmul(col[0], 2) ^ gmul(col[1], 3) ^ col[2] ^ col[3]
            state[i * 4 + 1] = col[0] ^ gmul(col[1], 2) ^ gmul(col[2], 3) ^ col[3]
            state[i * 4 + 2] = col[0] ^ col[1] ^ gmul(col[2], 2) ^ gmul(col[3], 3)
            state[i * 4 + 3] = gmul(col[0], 3) ^ col[1] ^ col[2] ^ gmul(col
```

- add_round_key: XOR state với round key.
- sub_bytes: thay thế byte bằng S-box.
- shift_rows: dịch vòng các hàng.
- mix_columns: trộn cột để khuếch tán dữ liệu.

4. *Mở rộng khoá (Key Expansion)*

```
def key_expansion(key):
    key_len = len(key)
    Nk = key_len // 4
    Nr = Nk + 6

    w = list(key)

    def sub_word(word):
        return [S_BOX[b] for b in word]

    def rot_word(word):
        return word[1:] + word[: 1]

    i = Nk
```

```
    while len(w) < 16 * (Nr + 1):
        temp = w[-4:]

        if i % Nk == 0:
            temp = sub_word(rot_word(temp))
            temp[0] ^= RCON[i // Nk]
        elif Nk > 6 and i % Nk == 4:
            temp = sub_word(temp)

        w.extend([w[-4 * Nk + j] ^ temp[j] for j in range(4)])
        i += 1

    return [w[i:i + 16] for i in range(0, len(w), 16)]
```

- Sinh các round key từ khóa ban đầu.
- Số round key phụ thuộc độ dài khóa.
- Đảm bảo mỗi vòng sử dụng khóa khác nhau.

5. *Mã hoá và giải mã một block*

```
def aes_encrypt_block(block, round_keys):
    state = list(block)
    Nr = len(round_keys) - 1

    add_round_key(state, round_keys[0])

    for r in range(1, Nr):
        sub_bytes(state)
        shift_rows(state)
        mix_columns(state)
```

```
        add_round_key(state, round_keys[r])

    sub_bytes(state)
    shift_rows(state)
    add_round_key(state, round_keys[Nr])

    return bytes(state)

def aes_decrypt_block(block, round_keys):
    state = list(block)
    Nr = len(round_keys) - 1

    add_round_key(state, round_keys[Nr])

    for r in range(Nr - 1, 0, -1):
        shift_rows(state, inv=True)
        sub_bytes(state, inv=True)
        add_round_key(state, round_keys[r])
        mix_columns(state, inv=True)

    shift_rows(state, inv=True)
    sub_bytes(state, inv=True)
    add_round_key(state, round_keys[0])

    return bytes(state)
```

- Áp dụng:

- Initial Round
- Main Rounds
- Final Round (không có MixColumns)

- Giải mã thực hiện các phép biến đổi theo thứ tự ngược.

6. *Padding dữ liệu*

```
def pad(data):
    padding_len = 16 - (len(data) % 16)
    return data + bytes([padding_len] * padding_len)

def unpad(data):
    if len(data) == 0:
        raise ValueError("Cannot unpad empty data")

    padding_len = data[-1]

    if padding_len == 0 or padding_len > 16:
        raise ValueError(f"Invalid padding length: {padding_len}")

    if len(data) < padding_len:
        raise ValueError(f"Data too short for padding:  {len(data)} < {padding_len}")

    if data[-padding_len:] != bytes([padding_len] * padding_len):
        raise ValueError("Invalid padding")

    return data[:-padding_len]
```

- Sử dụng PKCS#7 padding.
- Đảm bảo dữ liệu có độ dài bội số của 16 byte.

7. Mã hoá AES

```
def encrypt(plaintext, key, mode, iv=None):
    if len(key) not in (16, 24, 32):
        raise ValueError("Key must be 16, 24, or 32 bytes")

    round_keys = key_expansion(key)
    padded = pad(plaintext)
    ciphertext = b""

    if mode == "ECB":
        for i in range(0, len(padded), 16):
            block = padded[i:i + 16]
            ciphertext += aes_encrypt_block(block, round_keys)
        return ciphertext, None

    elif mode == "CBC":
        if iv is None:
            iv = os.urandom(16)
        if len(iv) != 16:
            raise ValueError("IV must be 16 bytes")

        prev = iv
        for i in range(0, len(padded), 16):
            block = padded[i:i + 16]
            xored = bytes(a ^ b for a, b in zip(block, prev))
            encrypted = aes_encrypt_block(xored, round_keys)
```

```
            ciphertext += encrypted
            prev = encrypted

        return ciphertext, iv

    else:
        raise ValueError(f"Unsupported mode: {mode}")
```

- **ECB:** mã hóa từng block độc lập.
- **CBC:** XOR plaintext với IV hoặc block trước khi mã hóa.
- IV được sinh ngẫu nhiên nếu không cung cấp.

8. Giải mã AES

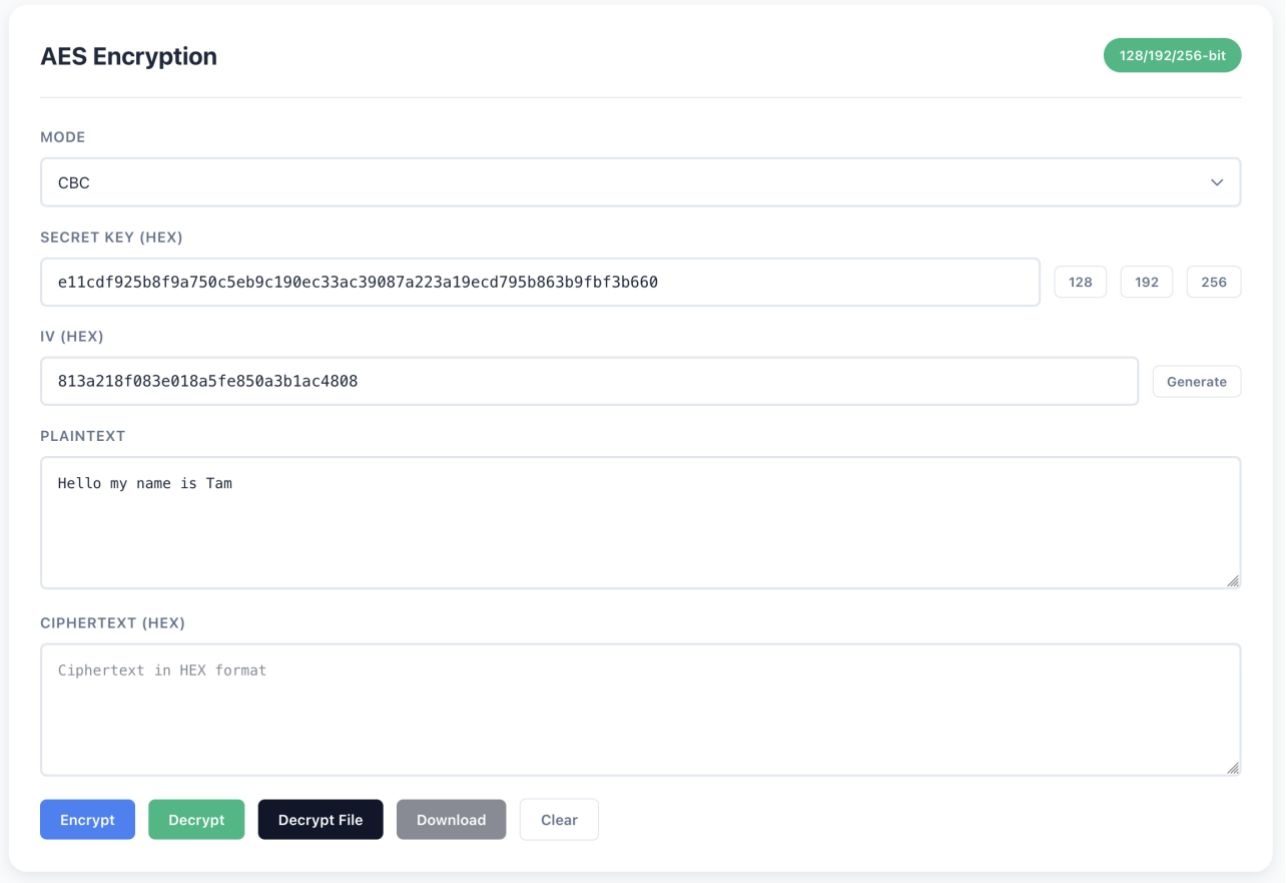
```
def decrypt(ciphertext, key, mode, iv=None):
    if len(key) not in (16, 24, 32):
        raise ValueError("Key must be 16, 24, or 32 bytes")

    if len(ciphertext) % 16 != 0:
        raise ValueError("Ciphertext length must be a multiple of 16")

    if len(ciphertext) == 0:
        raise ValueError("Ciphertext cannot be empty")

    round_keys = key_expansion(key)
    plaintext = b""

    if mode == "ECB":
        for i in range(0, len(ciphertext), 16):
```

```
        block = ciphertext[i:i + 16]
        plaintext += aes_decrypt_block(block, round_keys)
    return unpad(plaintext)

elif mode == "CBC":
    if iv is None:
        raise ValueError("IV required for CBC mode")
    if len(iv) != 16:
        raise ValueError("IV must be 16 bytes")

    prev = iv
    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i: i + 16]
        decrypted = aes_decrypt_block(block, round_keys)
        plaintext += bytes(a ^ b for a, b in zip(decrypted, prev))
        prev = block

    return unpad(plaintext)

else:
    raise ValueError(f"Unsupported mode: {mode}")
```

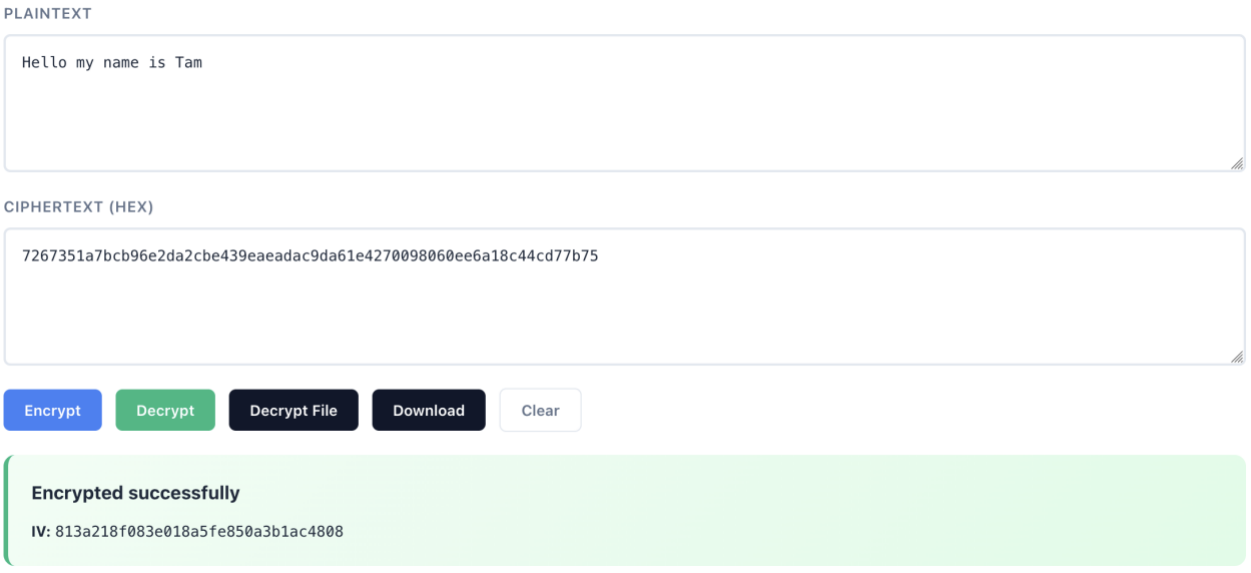
- **ECB:** giải mã từng block.
- **CBC:** giải mã rồi XOR với block trước.
- Thực hiện unpadding để thu plaintext gốc.

e) Kết quả

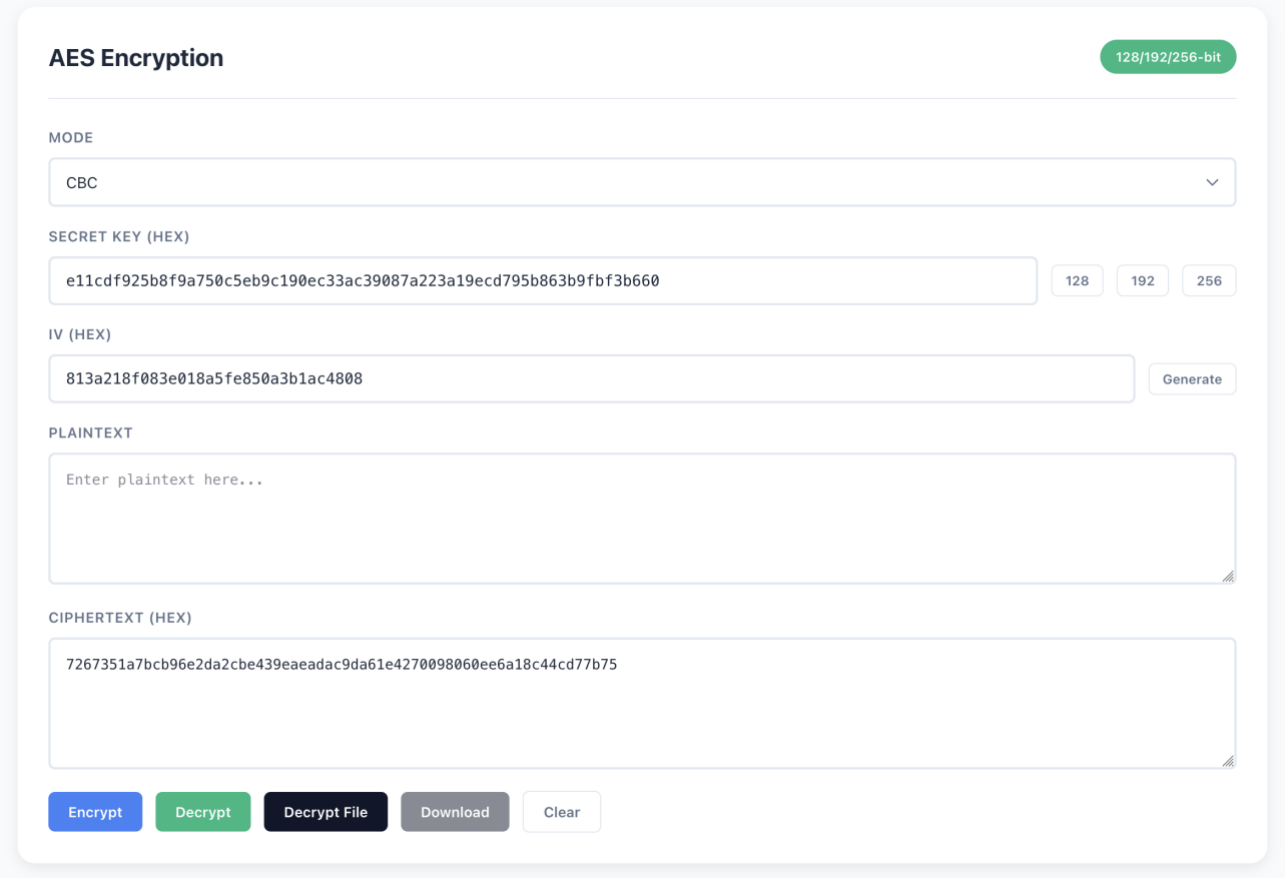
- Kết quả của chế độ CBC cho phép tải về file plaintext.txt.



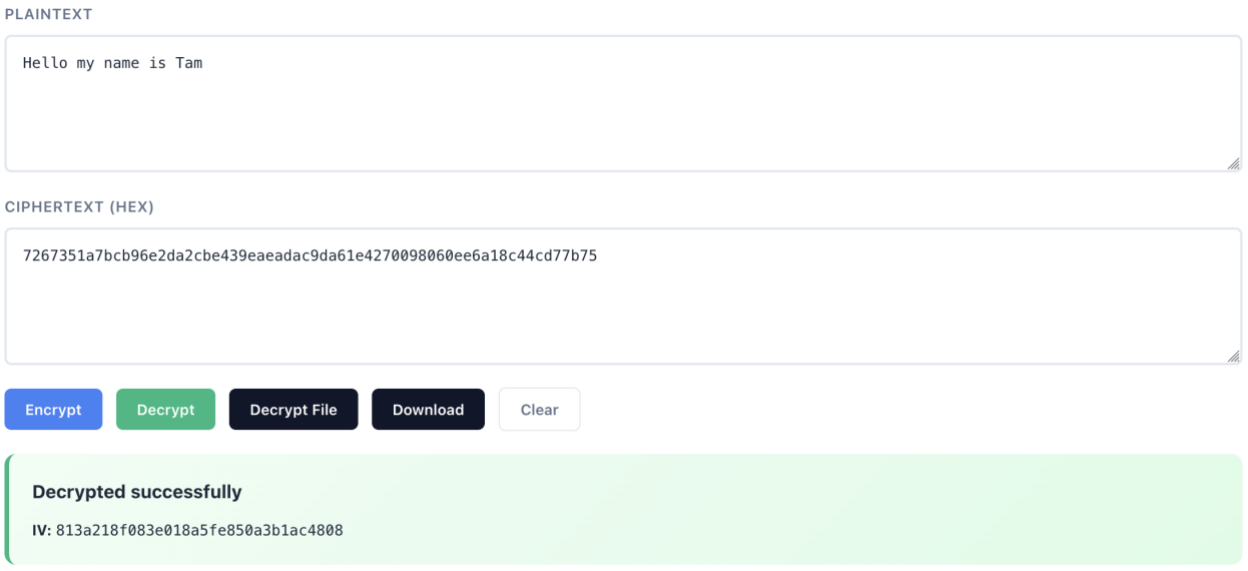
Hình 23: Giao diện chức năng AES Encryption ở chế độ CBC, hỗ trợ khóa 128/192/256-bit



Hình 24: Kết quả mã hóa AES (CBC), hiển thị ciphertext dạng HEX và IV

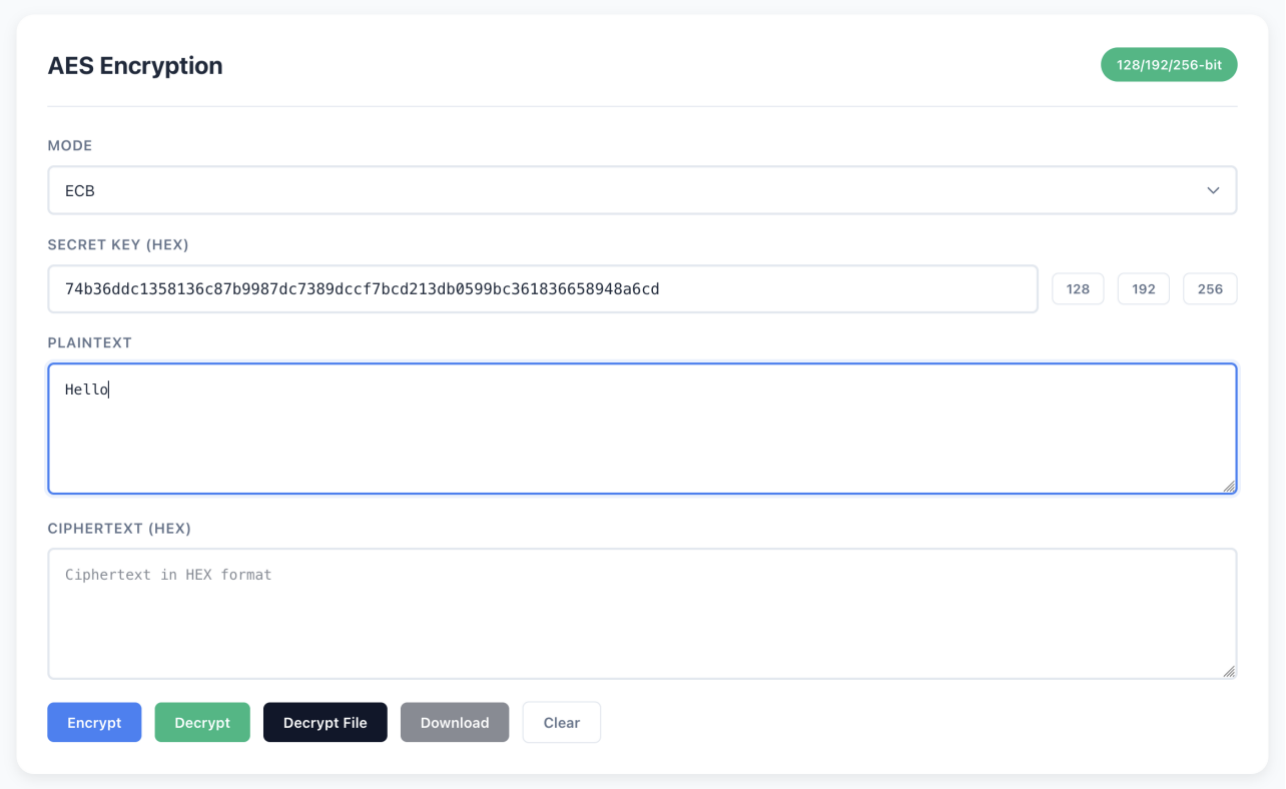


Hình 25: Ciphertext sau mã hóa AES, sẵn sàng cho bước giải mã

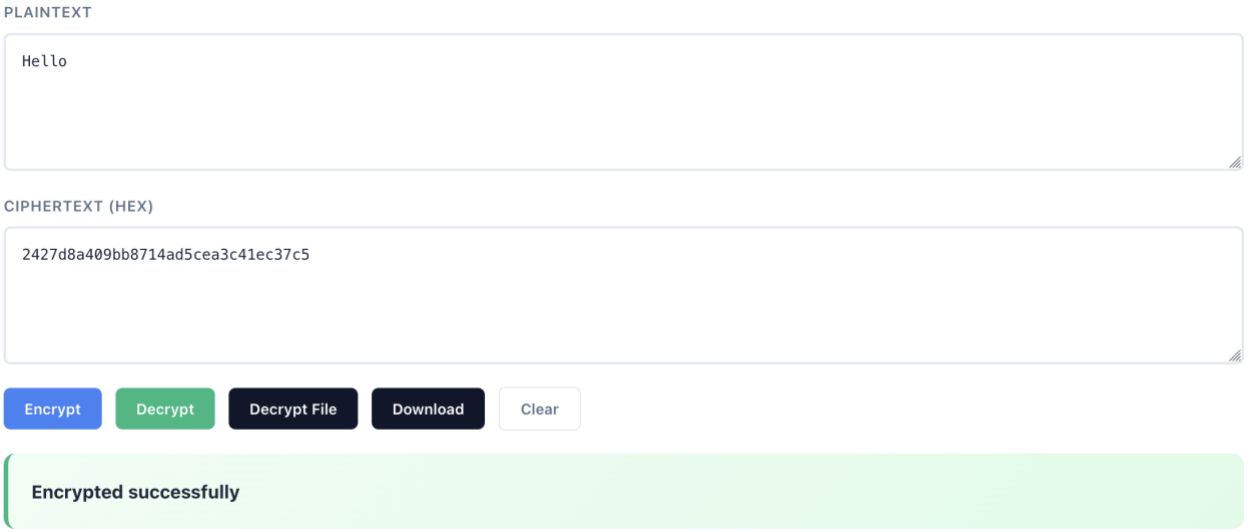


Hình 26: Kết quả giải mã AES (CBC), khôi phục đúng plaintext ban đầu

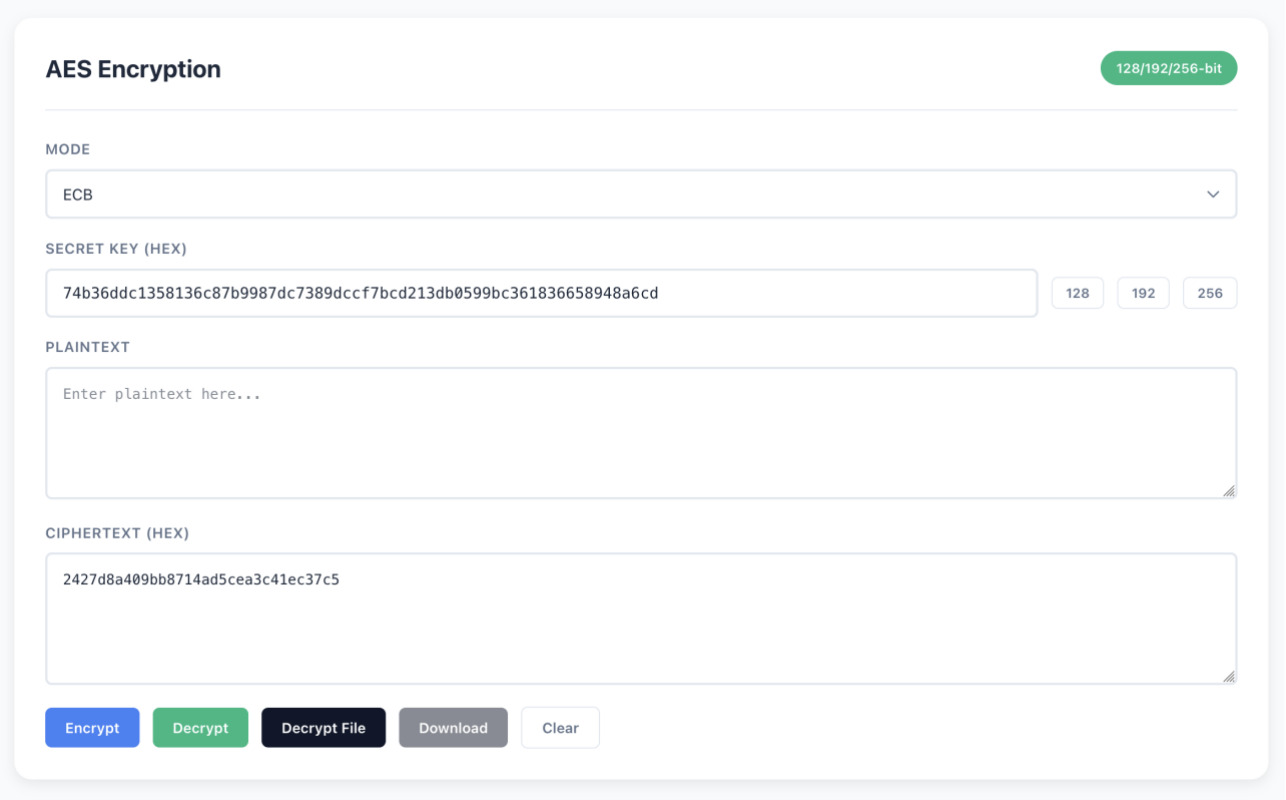
- Kết quả của chế độ ECB cho phép tải về file plaintext.txt.



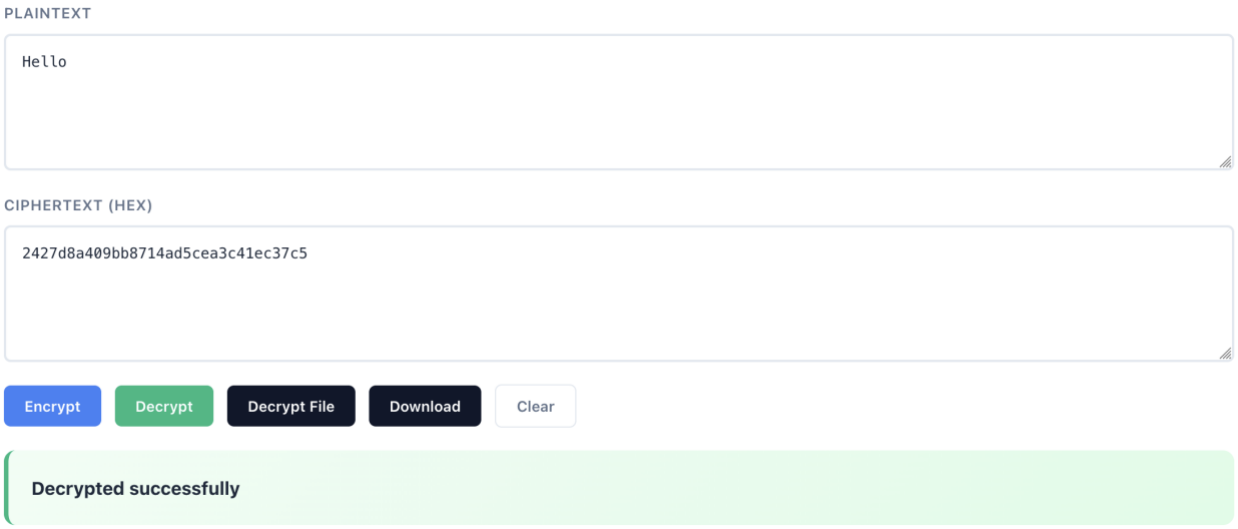
Hình 27: Giao diện chức năng AES Decryption ở chế độ ECB, hỗ trợ khóa 128/192/256-bit



Hình 28: Kết quả mã hóa AES (ECB), hiển thị ciphertext dạng HEX.



Hình 29: Ciphertext sau mã hóa AES ở chế độ ECB, sẵn sàng cho bước giải mã



Hình 30: Kết quả giải mã AES (ECB), khôi phục đúng plaintext ban đầu

HẾT