BÁO CÁO THỰC HÀNH

Thực hành An toàn mạng máy tính

Lab 6: REVIEW OF ENCRYPTION ALGORITHMS

GVTH: Nguyễn Ngọc Trưởng

Ngày báo cáo: 22/11/2025

Nhóm 7 – NT101.Q13.2

THÔNG TIN CHUNG

MSSV	Họ và tên	Nội dung báo cáo	% công việc
23521422	Huỳnh Lê Đại Thắng	Tất cả	100%

1. Giới thiệu chung

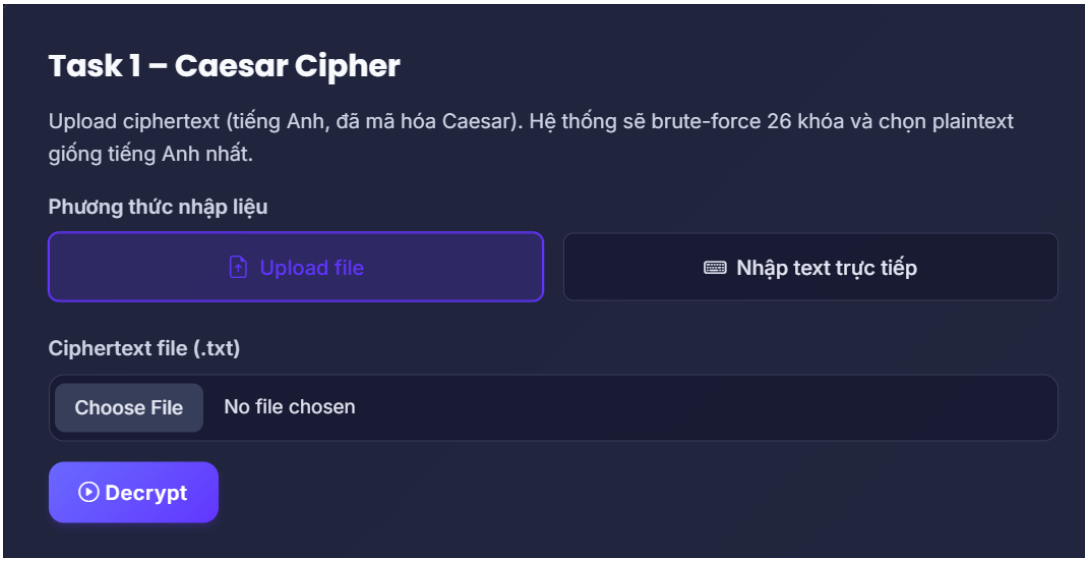
Chương trình ***Lab06 – Review of Encryption Algorithms*** là một ứng dụng Web full-stack được xây dựng chủ yếu bằng Python (Flask) ở backend, kết hợp HTML/CSS/JavaScript ở frontend. Ứng dụng tập trung vào phân tích – phá mã các thuật toán cổ điển và mã hóa hiện đại DES/AES. Hệ thống cho phép:

- Phá mã tự động 3 mật mã cổ điển: Caesar, Monoalphabetic Substitution, Vigenère.
- Mã hóa/giải mã với 2 thuật toán hiện đại DES và AES, hỗ trợ các mode ECB và CBC.
- Tự cài đặt toàn bộ phần core cryptography (DES/AES, cryptanalysis) từ đầu, không dùng thư viện mã hóa ngoài; các thư viện như pycryptodome chỉ dùng cho tiện ích phụ trợ (hex, test).
- Giao diện Web một trang (single page), hỗ trợ Dark/Light Mode, responsive, hiệu ứng glass morphism, kèm một AI Chatbot về lý thuyết mật mã (offline + tích hợp Google Gemini khi có API key).

2. Chức năng chính của chương trình

a) 2.1. Nhóm Cryptanalysis (phân tích và phá mã)

Task 1 – Caesar Cipher Breaker

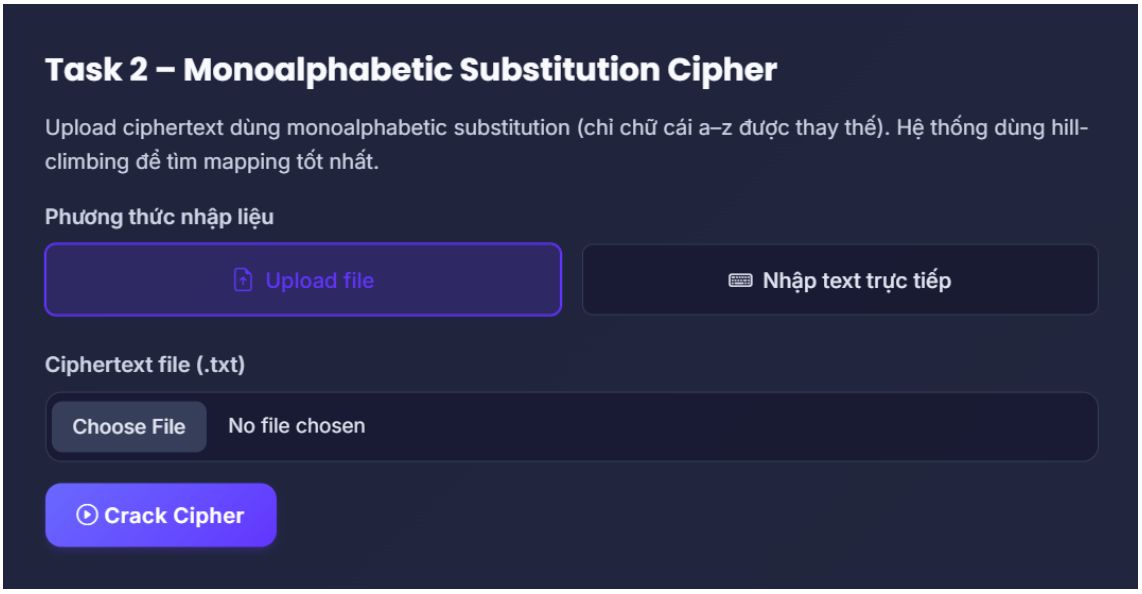


Hình 1. *Giao diện chương trình Caesar Breaker*

- Chương trình thực hiện vét cạn toàn bộ 26 khóa có thể của Caesar.

- Với mỗi khóa, hệ thống tiến hành giải mã bản mã rồi tính giá trị thống kê *Chi-Square* so với tần suất xuất hiện chữ cái chuẩn trong tiếng Anh. Dòng plaintext nào có phân bố gần với tiếng Anh nhất sẽ cho giá trị *Chi-Square* nhỏ nhất.
- Kết quả cuối cùng là cặp (*best_key*, *best_plaintext*) tương ứng với giá trị *Chi-Square* nhỏ nhất.

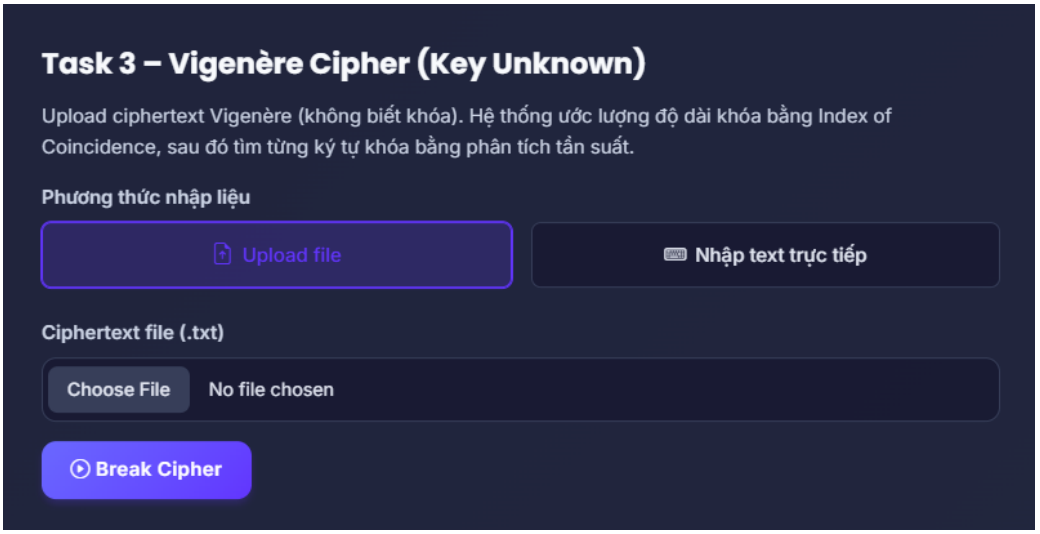
Task 2 – Substitution Cipher Breaker



Hình 2. *Giao diện chương trình Substitution Cipher Breaker*

- Task này dùng để phá monoalphabetic substitution (A–Z bị thay thế theo một hoán vị 26 chữ).
- Do không thể brute-force (không gian khóa 26!), chương trình tìm khóa tốt bằng *hill-climbing* kết hợp *random restart* (thỉnh thoảng có *annealing* để thoát kẹt).
- Khóa được chọn dựa trên điểm ngôn ngữ: thống kê n-gram (đặc biệt *quadgram*) và thưởng theo từ điển tiếng Anh, plaintext càng giống tiếng Anh thì điểm càng cao.

Task 3 – Vigenère Cipher Breaker (không biết khóa trước)

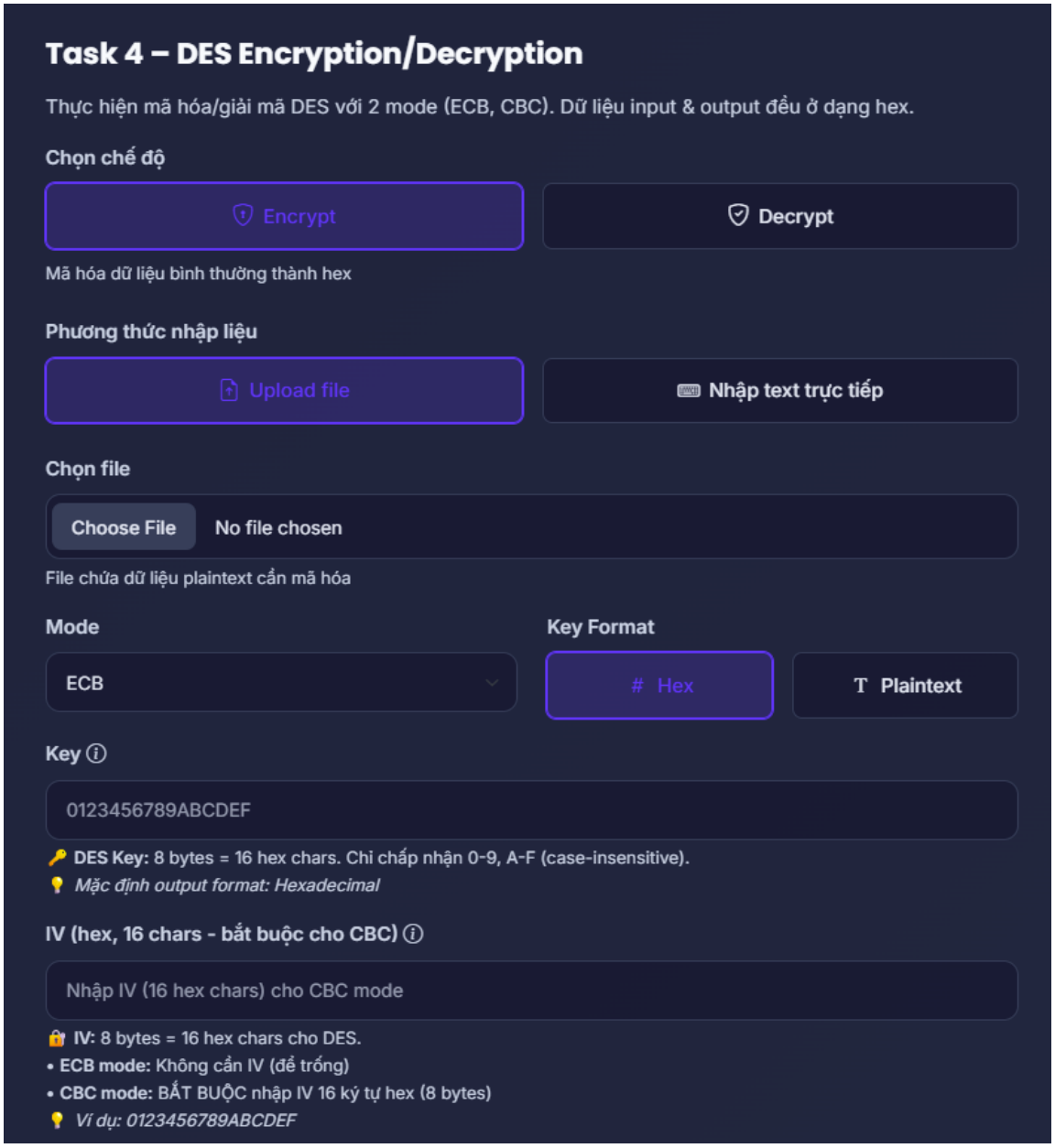


Hình 3. *Giao diện chương trình Vigenère Cipher Breaker*

- Trong bài toán này không biết trước độ dài khóa Vigenère, nên trước hết chương trình sử dụng *Index of Coincidence (IC)* để ước lượng các độ dài khóa có khả năng đúng.
- Với mỗi giá trị *key_len* ứng viên, chương trình tách ciphertext thành *key_len* chuỗi con. Mỗi chuỗi con được xem như một Caesar cipher riêng.
- Đối với từng chuỗi con, chương trình áp dụng phân tích tần suất kết hợp *Chi-Square* để tìm ra ký tự khóa phù hợp nhất. Ghép các ký tự đó lại sẽ thu được khóa hoàn chỉnh cho độ dài đang xét.
- Sau đó, chương trình dùng khóa này để giải toàn bộ ciphertext, tính lại *Chi-Square* cho bản rõ thu được và chọn cặp (key, plaintext) có chất lượng tốt nhất.

b) 2.2. Nhóm mã hóa hiện đại – Modern Encryption

Task 4 – DES Encryption/Decryption (ECB, CBC)



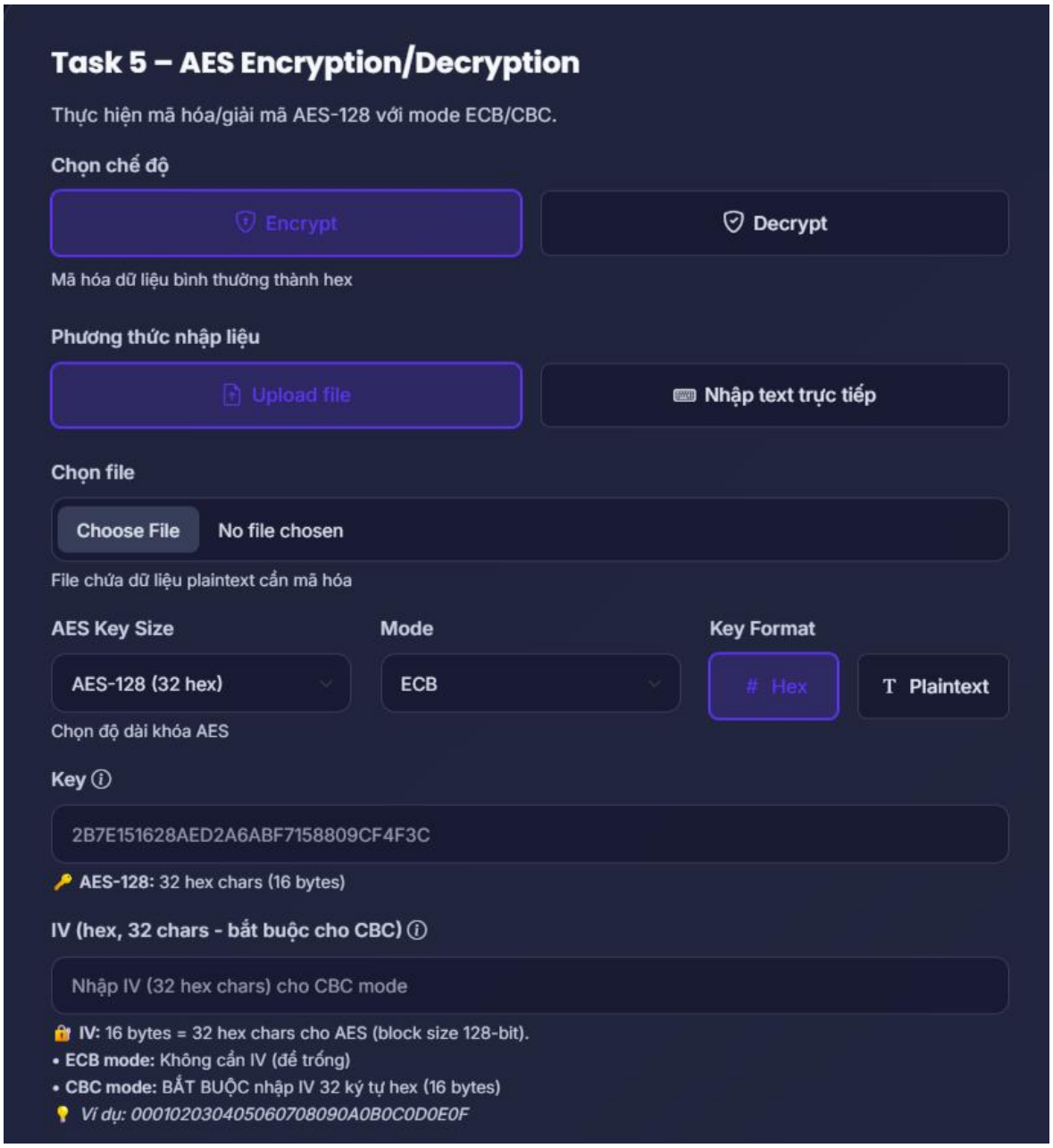
Hình 4. *Giao diện chương trình DES Encryption/Decryption (ECB, CBC)*

– Chương trình tự cài đặt nhân DES (DES core) với các đặc điểm:
  - Kích thước khối: 64 bit (8 byte).
  - Khóa hiệu dụng dài 56 bit (nhận đầu vào 8 byte, 64 bit, có các bit parity).
  - Thực hiện đầy đủ 16 vòng *Feistel* cùng với các bước *IP, FP, E-box, P-box, S-box, PC1, PC2* và key schedule để sinh subkey cho từng vòng.
– Hai chế độ hoạt động được hỗ trợ là:
  - **ECB**: mỗi block plaintext được mã hóa độc lập với nhau.
  - **CBC**: mỗi block thứ *i* được tính theo công thức

$$C_i = E(P_i \oplus C_{i-1}) \text{với } C_0 = IV.$$

- Cả ECB và CBC đều sử dụng padding ***PKCS#7***.
- Khi mã hóa ở chế độ CBC, nếu người dùng không truyền *IV* thì chương trình sẽ tự sinh một *IV* ngẫu nhiên và trả về *IV* này cùng với ciphertext.

Task 5 – AES Encryption/Decryption (ECB, CBC)



Hình 5. *Giao diện chương trình AES Encryption/Decryption (ECB, CBC)*

- Nhân AES được cài đặt cho block kích thước 128 bit (16 byte). Chương trình hỗ trợ ba độ dài khóa:
  - AES-128: khóa dài 16 byte, tương ứng 10 vòng lặp.
  - AES-192: khóa dài 24 byte, tương ứng 12 vòng lặp.
  - AES-256: khóa dài 32 byte, tương ứng 14 vòng lặp.
- Các thành phần lõi của AES đều được tự triển khai, bao gồm:
  - SubBytes với S-box chuẩn.
  - ShiftRows.
  - MixColumns.
  - AddRoundKey.
  - Key Expansion để sinh toàn bộ round key từ khóa ban đầu.
- Hai mode hoạt động:
  - **ECB**: mỗi block được mã hóa độc lập.
  - **CBC**: XOR block plaintext với block ciphertext trước đó rồi mới đưa vào vòng AES, cần IV dài 16 byte cho block đầu tiên.
- Padding ***PKCS#7*** được sử dụng tương tự như ở DES.

c) 2.3. Các tính năng bổ sung (Validation, UI, Chatbot)

Character set validation & filtering

- Chương trình quy định một tập ký tự cho phép bao gồm:
  - Chữ cái: a–z, A–Z
  - Chữ số: 0–9
  - Khoảng trắng
  - Các dấu câu: . , ; : ? ! ' " - ( )
- Mọi ký tự nằm ngoài tập cho phép sẽ tự động được thay thế bằng khoảng trắng. Đồng thời hệ thống sinh ra cảnh báo (warning) để hiển thị trên giao diện người dùng.

Web interface – Single Page App

– Ứng dụng Web được thiết kế dạng Single Page App, sử dụng một file *templates/index.html* duy nhất để chứa giao diện cho cả 5 task: Caesar, Substitution, Vigenère, DES và AES.

Hình 6. *Giao diện chính của chương trình*

– Các tính năng chính của giao diện:

- Hỗ trợ chế độ Dark/Light, có nút chuyển đổi và lưu trạng thái theme bằng CSS/JS.
- Cho phép người dùng upload file hoặc nhập dữ liệu trực tiếp cho từng task.
- Cung cấp nút "Copy to clipboard" cho plaintext, ciphertext và IV.
- Hiển thị thông báo dạng toast, overlay loading khi xử lý, kiểm tra dung lượng và phần mở rộng file ở phía client.

AI Chatbot – Crypto Assistant

– Hệ thống tích hợp một chatbot chuyên về lý thuyết mật mã và nội dung project *Lab06*, có thể giải thích các khái niệm như Caesar, Substitution, Vigenère, DES, AES, sự khác nhau giữa các mode ECB/CBC và chỉ ra file code tương ứng trong project.

- Chatbot sử dụng một knowledge base offline trong *crypto/chatbot_knowledge.py* để trả lời nhanh các câu hỏi phổ biến, gần như không có độ trễ.
- Nếu cấu hình biến môi trường GEMINI_API_KEY trong file .env, chatbot có thể kết hợp với Google Gemini: ưu tiên dùng dữ liệu offline, khi không đủ thì mới gọi Gemini để trả lời chi tiết hơn.

3. Cơ sở lý thuyết, ý tưởng và thuật toán

d) 3.1. Task 1 – Phá mã Caesar (Caesar Cipher Breaker)

3.1.1. Ý tưởng

- Mật mã Caesar chỉ dịch chuyển vòng quanh bảng chữ cái với 26 khóa (0–25) do đó có thể *brute-force toàn bộ* không gian khóa.
- Với mỗi khóa, ta giải mã ciphertext rồi so sánh phân bố tần suất chữ cái thu được với phân bố chuẩn tiếng Anh bằng *Chi-Square statistic.*
- Khóa cho *giá trị Chi-Square nhỏ nhất* được chọn là khóa đúng (plaintext "giống tiếng Anh nhất").

3.1.2. Thuật toán chi tiết

1. Chuẩn hóa ciphertext:
   - Chuyển về chữ hoa, loại bỏ ký tự không phải A–Z (hoặc thay bằng khoảng trắng).
2. Với mỗi khóa k từ 0 đến 25:
   - Giải mã: dịch lùi mỗi chữ cái 1 khoảng k (mod 26).
   - Đếm tần suất 26 chữ cái trong plaintext.
   - Áp dụng công thức *Chi-Square*:

$$\chi^2 = \sum_{i=0}^{25} \frac{(O_i - E_i)^2}{E_i}$$

     - O_i: số lần xuất hiện của chữ cái thứ i trong plaintext.
     - E_i: kỳ vọng (tần suất chuẩn tiếng Anh × độ dài văn bản).
3. Ghi nhận *(best_key, best_score, best_plaintext)* với χ^2 nhỏ nhất.
4. Trả về kết quả.

3.1.3. Code minh họa (rút gọn)

(1) Hàm dịch 1 ký tự

```python
def shift_char(c: str, k: int) -> str:
    """
    Dịch 1 ký tự theo khóa k (0-25). Giữ nguyên ký tự không phải chữ cái.
    Optimized: reduced branching, direct calculation.
    """
    if not c.isalpha():
        return c

    base = ord("A") if c.isupper() else ord("a")
    return chr((ord(c) - base - k) % 26 + base)
```

Hình 7. *Hàm dịch 1 ký tự*

- Nếu c không phải chữ cái thì giữ nguyên (giữ khoảng trắng, số, dấu câu…).
- Nếu là chữ cái:
  - Quy về chỉ số 0–25.
  - Trừ đi khóa k (giải mã Caesar).
  - Lấy modulo 26 rồi cộng lại về mã ASCII.

(2) Hàm giải mã với 1 khóa

```python
def decrypt_caesar_with_key(ciphertext: str, k: int) -> str:
    """
    Trả về plaintext khi giải mã ciphertext bằng key k.
    KHÔNG bao giờ trả về None.
    """
    return "".join(shift_char(c, k) for c in ciphertext)
```

Hình 8. *Hàm giải mã với 1 khóa*

- Duyệt từng ký tự trong ciphertext, áp dụng *shift_char* với khóa k.
- Trả về *plaintext ứng với khóa k.*

(3) Hàm tính điểm Chi-Square

```python
def chi_square_score(text: str) -> float:      You, 2 weeks ago · update 2
    """
    Tính chi-square statistic giữa phân bố chữ cái của text và phân bố
    tiếng Anh chuẩn. Chi-square càng nhỏ -> càng giống tiếng Anh.

    Optimized: faster counting, reduced operations.
    """
    if not text:
        return float("inf")

    # Count letters efficiently using list (faster than dict for 26 items)
    counts = [0] * 26
    total = 0

    for c in text.upper():
        if c in LETTERS_SET:
            counts[ord(c) - 65] += 1
            total += 1

    if total == 0:
        return float("inf")

    # Calculate chi-square with pre-computed frequencies
    chi_sq = 0.0
    for i, letter in enumerate(LETTERS):
        observed = counts[i]
        expected = ENGLISH_FREQ[letter] * total
        if expected > 0:  # Avoid division by zero
            chi_sq += (observed - expected) ** 2 / expected

    return chi_sq
```

Hình 9. *Hàm tính điểm Chi-Square*

- Đếm số lần xuất hiện của mỗi chữ cái A–Z.
- total = tổng số chữ cái (bỏ qua số, dấu…).
- Với mỗi chữ letter:
    - observed = số lần xuất hiện trong text.
    - expected = tần suất chuẩn tiếng Anh (ENGLISH_FREQ) * total.
- Tính Chi-Square

- Giá trị càng nhỏ → text càng giống tiếng Anh.

(4) Hàm giải mã caesar

```python
def break_caesar(ciphertext: str):
    """
    Bruteforce 26 khóa, chấm điểm từng plaintext bằng chi-square.
    Trả về (best_key, best_plaintext).
    """
    best_key = 0
    best_plain = ""
    best_score = float("inf")  # chi-square càng nhỏ càng tốt

    # Đảm bảo ciphertext là string
    if ciphertext is None:
        ciphertext = ""

    for k in range(26):
        plain = decrypt_caesar_with_key(ciphertext, k)
        score = chi_square_score(plain)

        if score < best_score:
            best_score = score
            best_key = k
            best_plain = plain

    return best_key, best_plain
```

Hình 10. *Hàm giải mã Caesar*

- Khởi tạo *best_score* = +∞, chưa biết khóa tốt nhất.
- Lặp k từ 0 → 25:
    - Giải mã ciphertext với khóa k.
    - Tính score = chi_square_score(plain).
    - Nếu score tốt hơn (nhỏ hơn) → cập nhật best_key, best_plain.

- Kết quả:
  - Khóa Caesar tốt nhất.
  - Plaintext tương ứng.

e) 3.2. Task 2 – Phá mã Substitution (Monoalphabetic)

3.2.1. Ý tưởng

- Không gian khóa: **26!** rất lớn nên không thể brute-force.
- Dùng *hill-climbing + random restart* trên space khóa:
  - Khởi tạo một key (mapping) ngẫu nhiên.
  - Mỗi bước, thực hiện hoán đổi 2 ký tự trong key → key mới.
  - Giải mã và tính *score* dựa trên *quadgram statistics* + bonus dictionary.
  - Nếu score tốt hơn chấp nhận hoán đổi; nếu không bỏ qua (hoặc *annealing* nhẹ).
  - Lặp đi lặp lại, kết hợp nhiều lần khởi tạo lại *(random restart)* để tránh kẹt *local optimum.*

3.2.2. Hàm đánh giá (score)

- Dùng bảng tần suất *quadgram tiếng Anh* (file *english_quadgrams.txt* trong project).
- Tính:

$$\text{score(plaintext)} = \sum \log P(\text{quadgram}) + \text{word\_bonus}$$

- *word_bonus*: cộng thêm điểm nếu trong plaintext có nhiều từ nằm trong *wordlist.txt*.

3.2.3. Thuật toán hill-climbing

1. Bắt đầu từ một bảng thay thế (key) ban đầu, dùng nó giải thử ciphertext và chấm điểm "giống tiếng Anh".
2. Tạo các key "hàng xóm" bằng cách đổi chỗ 2 chữ trong key (swap 2 vị trí).
3. Với mỗi key hàng xóm, giải thử và chấm điểm, nếu điểm tốt hơn, nhận key mới ngay và tiếp tục lặp (cứ cải thiện dần).
4. Khi thử hoán đổi mà không còn key nào tốt hơn, thuật toán dừng (vì đã kẹt ở một đỉnh cục bộ).

5. (Annealing) thỉnh thoảng vẫn chấp nhận một bước tệ hơn một chút để thoát kẹt, rồi tiếp tục leo lên.

3.2.4. Code minh họa (rút gọn)

(1) Hàm score ngôn ngữ tổng hợp

```
def _language_score(text: str) -> float:
    """
    Score tổng hợp - TỐI ƯU CHO ACCURACY CAO NHẤT:
        Sử dụng full n-gram suite: bigram + trigram + quadgram + word bonus
    """
    # Load tất cả n-grams
    _load_bigrams()
    _load_trigrams()
    _load_quadgrams()

    text_lower = text.lower()
    s = "".join(c for c in text_lower if c in ALPHABET)
    s_len = len(s)

    if s_len < 2:
        return float("-inf")

    score = 0.0

    # Bigram (10% weight - pattern cơ bản nhưng quan trọng)
    if s_len >= 2 and _BI_LOG:
        bi_score = sum(_BI_LOG.get(s[i : i + 2], _BI_DEFAULT) for i in range(s_len - 1))
        score += 0.10 * bi_score

    # Trigram (20% weight - context ngắn)
    if s_len >= 3 and _TRI_LOG:
        tri_score = sum(
            _TRI_LOG.get(s[i : i + 3], _TRI_DEFAULT) for i in range(s_len - 2)
        )
        score += 0.20 * tri_score

    # Quadgram (60% weight - chính nhưng không áp đảo)
    if s_len >= 4 and _QUAD_LOG:
        quad_score = sum(
            _QUAD_LOG.get(s[i : i + 4], _QUAD_DEFAULT) for i in range(s_len - 3)
        )
        score += 0.60 * quad_score

    # Word bonus (10% nhưng rất quan trọng cho accuracy)
    word_score = _word_bonus(text)
```

Hình 11. *Hàm score ngôn ngữ tổng hợp*

- Trước hết, lọc chuỗi chỉ còn chữ cái a–z.
- Tính log-probability cho:
  - Bigram: s[i:i+2].
  - Trigram: s[i:i+3].
  - Quadgram: s[i:i+4].
- Mỗi n-gram có trọng số khác nhau (10/20/60%).
- *word_bonus* tăng điểm cho plaintext chứa nhiều từ có nghĩa, giúp chọn kết quả "đọc được tiếng Anh".

(2) Hàm break_substitution(ciphertext)

```python
def break_substitution(ciphertext: str, rounds: int = 80, consolidate: int = 6):
    """
    Hàm dùng trong Flask - TỐI ƯU CHO ĐỘ CHÍNH XÁC CAO NHẤT.

    Trả về:
        score (float),
        mapping_str: "cipher: abcdef... | plain : <key>",
        plaintext: ciphertext đã giải với key tốt nhất.

    Args:
        ciphertext: văn bản mã hóa cần giải
        rounds: số vòng hill-climb tối đa (80 - cao để đảm bảo accuracy)
        consolidate: số lần cần đạt cùng kết quả để xác nhận (6 - chắc chắn)
    """
    random.seed()

    if not ciphertext:
        mapping_str = "cipher: " + ALPHABET + " | plain : " + ALPHABET
        return 0.0, mapping_str, ""

    # Sử dụng sample size lớn để đảm bảo accuracy cao
    letters = [c for c in ciphertext if c.isalpha()]
    # Ưu tiên accuracy hơn speed
    if len(letters) > 8000:
        sample_size = 6000  # File rất lớn -> vẫn sample nhiều
    elif len(letters) > 4000:
        sample_size = 7000  # File lớn -> sample rất nhiều
    else:
        sample_size = len(letters)  # File nhỏ -> dùng hết

    score, key = _break_with_hillclimb(
        ciphertext, rounds=rounds, sample_letters=sample_size, consolidate=consolidate
    )

    plaintext = _apply_key(ciphertext, key)

    # Format mapping rõ ràng hơn (theo đề bài)
    cipher_line = "CIPHER: " + ALPHABET.upper()
    plain_line = "PLAIN : " + key.upper()
    mapping_str = cipher_line + " | " + plain_line

    return score, mapping_str, plaintext
```

Hình 12. *Hàm break_substitution(ciphertext)*

- Chọn một *sample* (hoặc toàn bộ ciphertext).
- Chạy *nhiều vòng hill-climbing (random restart)* với:

- Một số key khởi tạo random.
- Một số key khởi tạo từ *_frequency_seed*.

- Mỗi vòng hill-climb trả về (score, key).
- Chọn key có score cao nhất toàn cục.
- Dùng *_apply_key* để biến toàn bộ ciphertext thành plaintext.
- Trả về *score, mapping_str, plaintext.*

f) 3.3. Task 3 – Phá mã Vigenère (key unknown)

3.3.1. Ý tưởng

- Vigenère là *Caesar theo chu kỳ* nên mỗi vị trí trong khóa tương ứng một Caesar khác nhau.
- Không biết độ dài khóa nên sẽ dùng *Index of Coincidence* để đoán.
- Khi đoán được *key_len*, ta:
  - Chia ciphertext thành *key_len* dãy con: mỗi dãy là các ký tự cách nhau *key_len*.
  - Xem mỗi dãy là một ciphertext của Caesar, phá bằng *chi-square* như Task 1.
  - Ghép các shift lại thành full key.

3.3.2. IC – Index of Coincidence

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

- f_i: số lần xuất hiện chữ cái i.
- N: tổng số chữ cái.
- IC của tiếng Anh ≈ 0.065; IC chuỗi random ≈ 0.038.

3.3.3. Thuật toán chi tiết

1. Lọc ciphertext chỉ còn A–Z.
2. Với mỗi *key_len* từ 2–20:
   - Chia ciphertext thành *key_len* dãy con:
     - dãy 0: vị trí 0, key_len, 2×key_len, …
     - dãy 1: vị trí 1, 1+key_len, …

- Tính IC cho từng dãy, lấy trung bình.

3. Chọn một số *key_len* có IC cao nhất làm ứng viên.
4. Với mỗi *key_len* ứng viên:
   - Với từng vị trí i (0..key_len-1):
     - Lấy dãy con i.
     - Thử tất cả shift 0..25, làm *chi-square* và chọn shift tốt nhất.
   - Ghép các shift thành chuỗi key.
   - Giải mã ciphertext bằng key (Vigenère decrypt) và chấm điểm *(chi-square).*
5. Chọn (key, plaintext) có score tốt nhất.

3.3.4. Code minh họa (rút gọn)

(1) Index of Coincidence

```python
def _index_of_coincidence(seq: str) -> float:
    """
    IC cho chuỗi seq (chỉ gồm A-Z).
    Optimized: faster counting with list, reduced operations.
    """
    N = len(seq)
    if N <= 1:
        return 0.0

    counts = [0] * 26
    for ch in seq:
        if ch in ALPHABET_SET:
            counts[ord(ch) - 65] += 1

    # Calculate IC efficiently
    numerator = sum(c * (c - 1) for c in counts)
    return numerator / (N * (N - 1))
```

Hình 13. *Hàm Index of Coincidence*

(2) Solver chính *_break_vigenere_internal*

```python
def _break_vigenere_internal(ciphertext: str, max_key_len: int = 30, top_k: int = 10):
    """
    Solver chinh:
    - Lay chuoi letters = chi cac chu cai A-Z tu ciphertext.
    - Dung IC de chon ra mot so do dai khoa ung vien (top_k).
    - Moi key_len ung vien:
        + Chia letters thanh key_len subset.
        + Moi subset giai bang chi-square -> 1 ky tu khoa.
        + Ghep thanh key, giai toan ciphertext, tinh chi-square toan cuc.
    - Chon key co chi-square nho nhat.
    """
    letters = "".join(ch for ch in ciphertext.upper() if ch in ALPHABET)
    if len(letters) < 20:
        # too short to guess reliably
        return "A", decrypt_vigenere(ciphertext, "A"), float("inf")

    candidates = _guess_key_lengths_by_ic(letters, max_key_len, top_k)

    best_key = None
    best_plain = None
    best_score = float("inf")

    for key_len, avg_ic in candidates:
        shifts = []
        for i in range(key_len):
            subset = "".join(letters[j] for j in range(i, len(letters), key_len))
            shift = _best_shift_for_subset(subset)
            shifts.append(shift)

        key = "".join(ALPHABET[s] for s in shifts)
        plain = decrypt_vigenere(ciphertext, key)

        chi = _chi_square_text(plain)
        if chi < best_score:
            best_score = chi
            best_key = key
            best_plain = plain

    if best_key is not None:
        best_key = _reduce_repeating_key(best_key)
        best_plain = decrypt_vigenere(ciphertext, best_key)
```

Hình 14. *Hàm giải mã chính*

– Kết hợp tất cả các bước:

- Đoán độ dài khóa.
- Với mỗi độ dài: tìm các shift (phá các Caesar con).
- Lắp lại thành key, giải toàn bộ và chấm điểm.

g) 3.4. Task 4 – DES Encryption/Decryption (ECB, CBC)

3.4.1. Ý tưởng & cấu trúc DES

– *Block cipher Feistel 16 vòng* với block 64-bit và key 56-bit hiệu dụng.

– DES mã hóa theo các bước:

1. **Key Schedule:**
   - Key đầu vào: 8 bytes (64 bit) chứa 8 parity bits → 56 bit hiệu dụng.
   - Dùng bảng PC1, PC2 và mảng *SHIFTS* để tạo 16 subkey 48-bit.
2. **Mã hóa một block:**
   - IP – Initial Permutation.
   - Chia L0, R0 (32-bit).
   - Với mỗi vòng i=1..16:
     - L_i = R_{i-1}
     - R_i = L_{i-1} ⊕ F(R_{i-1}, K_i)
   - Sau 16 vòng: hoán đổi & áp dụng FP (Final Permutation).
3. **Hàm F (R, K):**
   - Expansion E: 32 → 48 bit.
   - XOR với subkey K (48 bit).
   - Chia thành 8 block 6-bit, qua 8 **S-box** → 32 bit.
   - P-box permutation → 32 bit.

3.4.2. Modes & Padding

- **ECB**:
  - Chia plaintext thành block 8 byte, pad bằng ***PKCS#7***.
  - Mã hóa từng block độc lập.
- **CBC**:
  - Dùng IV 8 byte.
  - Encrypt:

$$C_0 = IV, C_i = E(P_i \oplus C_{i-1})$$

  - Decrypt:

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

- **PKCS#7**: nếu độ dài P không chia hết 8 thì thêm k byte có giá trị k.

3.4.3. Code minh họa (rút gọn)

(1) Hàm Feistel F – _feistel

```python
def _feistel(R: int, subkey: int) -> int:        You, 2 weeks ag
    """
    F-function: 32-bit R -> 32-bit output
    """
    # Expansion E: 32 -> 48
    E_R = 0
    for pos in E:
        E_R = (E_R << 1) | ((R >> (32 - pos)) & 1)

    # XOR với subkey
    x = E_R ^ subkey

    # Đi qua 8 S-box (6 bits mỗi box -> 4 bits)
    out32 = 0
    for i in range(8):
        # Lấy 6 bit cho box i
        six_bits = (x >> (42 - 6 * i)) & 0x3F
        row = ((six_bits & 0x20) >> 4) | (six_bits & 0x01)
        col = (six_bits >> 1) & 0x0F
        s_val = S_BOXES[i][row][col]
        out32 = (out32 << 4) | s_val

    # Hoán vị P
    perm = 0
    for pos in P:
        perm = (perm << 1) | ((out32 >> (32 - pos)) & 1)
```

Hình 15. *Hàm Feistel*

- Mở rộng R 32-bit thành 48-bit bằng bảng E.
- XOR với subkey.

- Chia thành 8 nhóm 6-bit, đưa qua 8 S-box → thu được 32-bit.
- Áp dụng hoán vị P.

(2) Mã hóa 1 block – des_encrypt_block

```python
def des_encrypt_block(block8: bytes, subkeys) -> bytes:
    """
    Mã hóa 1 block 8 bytes bằng DES với list 16 subkeys.
    """
    block = int.from_bytes(block8, "big")

    # Initial Permutation
    perm = 0
    for pos in IP:
        perm = (perm << 1) | ((block >> (64 - pos)) & 1)

    L = (perm >> 32) & 0xFFFFFFFF
    R = perm & 0xFFFFFFFF

    # 16 vòng
    for i in range(16):
        new_L = R
        new_R = L ^ _feistel(R, subkeys[i])
        L, R = new_L, new_R

    # Swap L, R
    pre_output = (R << 32) | L

    # Final Permutation
    out = 0
    for pos in FP:
        out = (out << 1) | ((pre_output >> (64 - pos)) & 1)

    return out.to_bytes(8, "big")
```

Hình 16. *Hàm mã hóa 1 block*

h) 3.5. Task 5 – AES Encryption/Decryption (ECB, CBC)

3.5.1. Ý tưởng & cấu trúc AES

– AES là block cipher theo mô hình *Substitution–Permutation Network (SPN)*: block 128-bit, key 128/192/256-bit.

– Sử dụng *state 4×4 byte* và các phép biến đổi:

1. **Key Expansion:**
   - Từ key gốc sinh ra (Nr + 1) round key.
   - Dùng RotWord, SubWord (S-box), hằng số Rcon.
2. **Vòng mã hóa (encryption rounds):**
   - Initial: *AddRoundKey*.
   - Với mỗi vòng i từ 1 đến Nr-1:
     - SubBytes
     - ShiftRows
     - MixColumns
     - AddRoundKey
   - Vòng cuối bỏ MixColumns.
3. **Giải mã:**
   - Dùng các phép ngược: *InvShiftRows, InvSubBytes, InvMixColumns, AddRoundKey*.

3.5.2. Modes & Padding

- Block size: 16 byte.
- Modes:
  - **ECB**: từng block độc lập.
  - **CBC**: XOR với block trước + sử dụng IV 16 byte.
- Padding: **PKCS#7** (giống DES nhưng block_size = 16).

3.5.3. Code minh họa (rút gọn)

(1) ECB mode AES

```python
def _ecb_encrypt(plaintext: bytes, round_keys) -> bytes:
    plaintext = pkcs7_pad(plaintext, BLOCK_SIZE)
    out = []
    for i in range(0, len(plaintext), BLOCK_SIZE):
        block = plaintext[i : i + BLOCK_SIZE]
        out.append(aes_encrypt_block(block, round_keys))
    return b"".join(out)


def _ecb_decrypt(ciphertext: bytes, round_keys) -> bytes:
    if len(ciphertext) % BLOCK_SIZE != 0:
        raise ValueError("Ciphertext length not multiple of block size")
    out = []
    for i in range(0, len(ciphertext), BLOCK_SIZE):
        block = ciphertext[i : i + BLOCK_SIZE]
        out.append(aes_decrypt_block(block, round_keys))
    return pkcs7_unpad(b"".join(out))
```

Hình 17. *Hàm ECD mode AES*

(2) CBC mode AES

```python
def _cbc_encrypt(plaintext: bytes, round_keys, iv: bytes):
    if iv is None:
        iv = os.urandom(BLOCK_SIZE)
    if len(iv) != BLOCK_SIZE:
        raise ValueError("IV must be 16 bytes for AES CBC")
    plaintext = pkcs7_pad(plaintext, BLOCK_SIZE)
    out = []
    prev = iv
    for i in range(0, len(plaintext), BLOCK_SIZE):
        block = plaintext[i : i + BLOCK_SIZE]
        x = bytes(a ^ b for a, b in zip(block, prev))
        c = aes_encrypt_block(x, round_keys)
        out.append(c)
        prev = c
    return b"".join(out), iv


def _cbc_decrypt(ciphertext: bytes, round_keys, iv: bytes) -> bytes:
    if iv is None or len(iv) != BLOCK_SIZE:
        raise ValueError("IV must be 16 bytes for AES CBC decryption")
    if len(ciphertext) % BLOCK_SIZE != 0:
        raise ValueError("Ciphertext length not multiple of block size")

    out = []
    prev = iv
    for i in range(0, len(ciphertext), BLOCK_SIZE):
        block = ciphertext[i : i + BLOCK_SIZE]
        x = aes_decrypt_block(block, round_keys)
        p = bytes(a ^ b for a, b in zip(x, prev))
        out.append(p)
        prev = block
    return pkcs7_unpad(b"".join(out))
```

Hình 18. *Hàm CBC mode AES*

4. Mô tả cách hoạt động của chương trình sau khi chạy

1. Khởi động chương trình

- Cài thư viện: *pip install -r requirements.txt*
- Chạy server Flask: *python app.py*
- Truy cập ứng dụng qua trình duyệt tại địa chỉ *http://localhost:5000*.

2. **Giao diện chính**
   - Màn hình web gồm: tiêu đề Lab06, thông tin sinh viên, thanh chọn 5 task (Caesar, Substitution, Vigenère, DES, AES), vùng nhập dữ liệu và vùng hiển thị kết quả.
   - Có nút chuyển Dark/Light Mode và icon mở AI Chatbot hỗ trợ giải thích lý thuyết.
3. **Luồng hoạt động chung**
   - Người dùng chọn task, nhập dữ liệu:
     - Task 1–3: nhập hoặc upload ciphertext.
     - Task 4–5: nhập plaintext hoặc ciphertext, kèm key, IV, mode (ECB/CBC).
   - Frontend kiểm tra sơ bộ dữ liệu (định dạng, độ dài, hex) rồi gửi request đến Flask.
   - Flask nhận dữ liệu, gọi hàm tương ứng trong thư mục crypto (break_caesar, break_substitution, break_vigenere, des_encrypt/decrypt, aes_encrypt/decrypt).
   - Kết quả (plaintext, ciphertext, key, IV, điểm đánh giá) được render lại lên trang web, kèm nút copy và thông báo lỗi nếu có.
4. **Hoạt động nhóm phá mã (Task 1, 2, 3)**
   - Server chuẩn hóa ciphertext, sau đó:
     - Task 1: thử 26 khóa Caesar, chọn khóa có Chi-Square nhỏ nhất.
     - Task 2: dùng hill-climbing trên bảng hoán vị, chọn key cho score ngôn ngữ cao nhất.
     - Task 3: dùng Index of Coincidence đoán độ dài khóa, tách thành nhiều dãy và phá từng dãy như Caesar.
   - Hiển thị khóa suy ra và bản rõ tương ứng.
5. **Hoạt động nhóm mã hóa hiện đại (Task 4, 5)**
   - Server kiểm tra key, IV và mode:
     - DES: khối 8 byte, key 8 byte, IV 8 byte (CBC).
     - AES: khối 16 byte, key 16/24/32 byte, IV 16 byte (CBC).

- Khi mã hóa: áp dụng PKCS#7 padding, thực hiện DES/AES với mode ECB hoặc CBC, trả về ciphertext ở dạng hex và IV dùng trong CBC.
- Khi giải mã: chuyển ciphertext hex sang bytes, giải mã, bỏ padding rồi trả lại plaintext hiển thị cho người dùng.

LINK WEB DEPLOY

5. Nguồn tài liệu tham khảo

1. Giáo trình, slide môn An toàn mạng máy tính – Bộ môn An toàn thông tin, Khoa Mạng máy tính và Truyền thông, UIT.
2. DES (mã hóa) – Wikipedia tiếng Việt: giải thích cấu trúc Feistel 16 vòng, IP/FP, S-box, key schedule.
3. Advanced Encryption Standard (AES) – tiêu chuẩn FIPS PUB 197 của NIST: mô tả chi tiết thuật toán AES-128/192/256, SubBytes, ShiftRows, MixColumns, Key Expansion.
4. Bài viết "Advanced Encryption Standard" – Wikipedia tiếng Anh: tham khảo thêm về số vòng, kích thước khóa, dạng state 4×4 byte.
5. Các tài liệu về frequency analysis, Chi-Square test, Index of Coincidence, n-gram statistics dùng trong phân tích mật mã cổ điển (giáo trình mật mã, bài viết trên Crypto StackExchange, tài liệu thống kê tiếng Anh).

HẾT