**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**
**KHOA KHOA HỌC VÀ KĨ THUẬT THÔNG TIN**

---

# BÁO CÁO ĐỒ ÁN MÔN HỌC
# NHẬP MÔN ẨN THÔNG TIN

## CẢI TIẾN KỸ THUẬT LSB VỚI LỰA CHỌN PIXEL NGẪU NHIÊN VÀ KHÓA BÍ MẬT

**GVHD**: ThS. Nghi Hoàng Khoa

Sinh viên thực hiện:

| | |
|---|---|
| 1. Hồ Hoàng Phú | MSSV: 24410083 |
| 2. Phạm Nguyễn Hải Nam | MSSV: 24410070 |
| 3. Trương Gia Hiếu | MSSV: 24410031 |
| 4. Ngô Bảo Ngọc | MSSV: 24410073 |

□□ Tp. Hồ Chí Minh, 09/2025 □□

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

*……, ngày…….....tháng……năm 20…*

**Người nhận xét**

*(Ký tên và ghi rõ họ tên)*

## BẢNG PHÂN CÔNG, ĐÁNH GIÁ THÀNH VIÊN:

| Họ và tên | MSSV | Phân công | Đánh giá |
|---|---|---|---|
| **Trương Gia Hiếu** | 24410031 | - Tham gia thảo luận ý tưởng ban đầu<br>- Hỗ trợ code theo phân công.<br>- Hỗ trợ viết báo cáo phần giới thiệu lý thuyết cơ bản | 9/10 |
| **Phạm Nguyễn Hải Nam** | 24410070 | - Hỗ trợ tìm tài liệu tham khảo<br>- Hỗ trợ code theo phân công<br>- Hỗ trợ chuẩn bị slide thuyết trình | 9/10 |
| **Hồ Hoàng Phú** | 24410083 | - Xây dựng source base (Sequential LSB + cải tiến Random+Key)<br>- Cài đặt Benchmark (PSNR, thời gian, LSB-plane)<br>- Phân tích kết quả, viết báo cáo, chuẩn bị file nộp | 9/10 |
| **Ngô Bảo Ngọc** | 24410073 | - Hỗ trợ format báo cáo, kiểm tra lỗi chính tả, chỉnh sửa hình ảnh<br>- Hỗ trợ code theo phân công<br>- Hỗ trợ chuẩn bị slide | 9/10 |

*Bảng 1: Bảng phân công, đánh giá thành viên*

# MỤC LỤC

Contents

# 1. Giới thiệu

## 1.1 Khái niệm giấu tin và ứng dụng

- **Giấu tin (Steganography)** là kỹ thuật che giấu thông tin trong các đối tượng dữ liệu số (ảnh, âm thanh, video, văn bản…) sao cho người khác khó nhận biết sự tồn tại của thông tin bí mật.
- Khác với **mã hóa (Cryptography)**: mã hóa bảo vệ nội dung, nhưng sự tồn tại của dữ liệu thì hiển nhiên. Giấu tin lại làm cho dữ liệu **"vô hình"** đối với người quan sát.
- **Ứng dụng thực tế**:
    - Bảo mật thông tin liên lạc trong môi trường công cộng.
    - Gắn watermark bản quyền vào hình ảnh, video.
    - Phát hiện và theo dõi tội phạm mạng.
    - Truyền thông tin bí mật trong quân sự, ngoại giao.

*(Ảnh gốc không có giấu tin)*

*(Ảnh có thông điệp được embed vào)*

## 1.2 Các kỹ thuật giấu tin phổ biến

- **Giấu tin trong miền không gian (Spatial Domain)**
  - Kỹ thuật LSB (Least Significant Bit): thay thế bit ít quan trọng nhất trong pixel.
  - Ưu điểm: đơn giản, dung lượng chứa lớn.
  - Nhược: dễ bị phát hiện bởi phân tích thống kê.
- **Giấu tin trong miền biến đổi (Transform Domain)**
  - Sử dụng DCT (Discrete Cosine Transform), DWT (Discrete Wavelet Transform).
  - Ưu điểm: bền vững trước nén JPEG.
  - Nhược: phức tạp, dung lượng chứa thấp hơn.
- **Kỹ thuật lai (Hybrid)**: kết hợp LSB với mã hóa hoặc biến đổi miền để cân bằng dung lượng và độ bền.

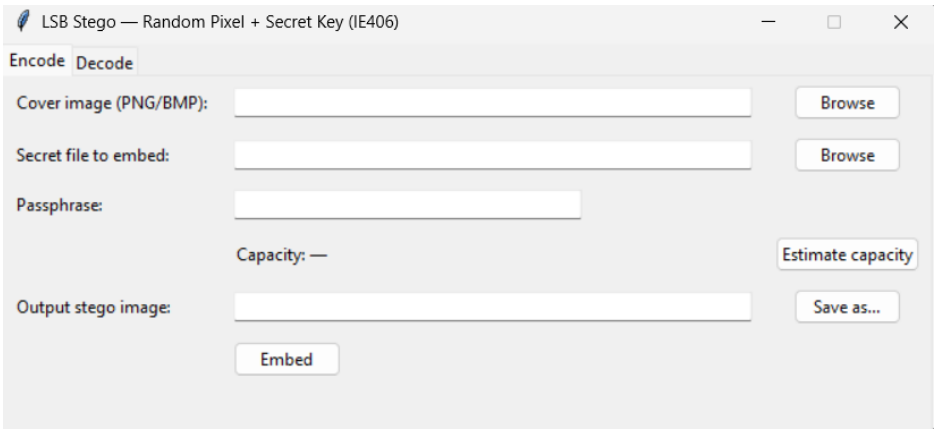
| **Characteristics** | Spatial domain | Transform domain |
|---|---|---|
| **Capacity** | Low | High |
| **Imperceptibility** | Yes | Yes |
| **Robustness** | No | Yes |
| **Speed** | Fast | Slow |
| **Time spending** | No | Yes |
| **Cost of operation** | Low | High |
| **Simplicity** | Yes | No |
| **Security** | No | Yes |
| **Computational load** | No | Yes |

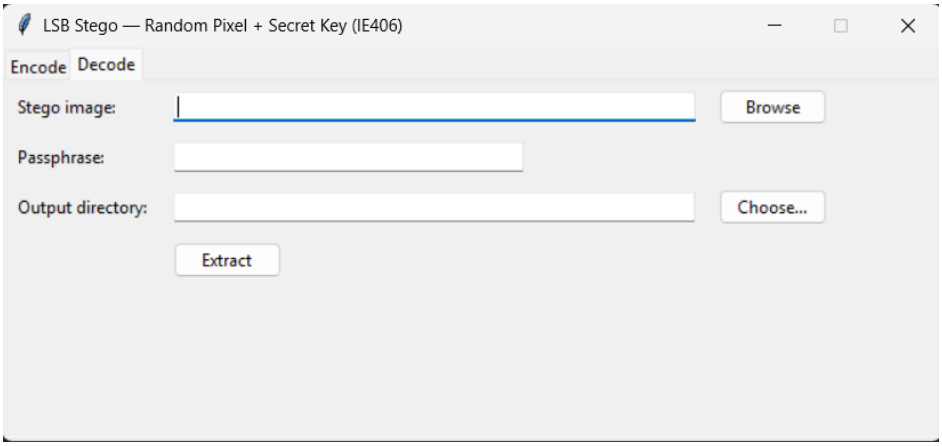
*(So sánh Spatial vs Transform domain)*

Miền không gian (spatial domain) hoạt động trực tiếp trên các pixel hình ảnh, trong khi miền biến đổi sử dụng một phép biến đổi toán học như Biến đổi Fourier để chuyển đổi hình ảnh thành biểu diễn dựa trên tần số, cho phép xử lý hiệu quả hơn hoặc mạnh mẽ hơn về giảm nhiễu, tạo hình mờ hoặc phát hiện tính năng.

## 1.3 Mục tiêu và phạm vi của đề tài

- **Mục tiêu:**
  - Xây dựng hệ thống giấu tin dựa trên kỹ thuật LSB.
  - Cải tiến phương pháp nhúng để giảm khả năng bị phát hiện.
  - Đảm bảo chất lượng ảnh sau khi giấu vẫn ở mức cao (PSNR > 40 dB).
- **Phạm vi:**
  - Tập trung vào ảnh tĩnh định dạng **PNG** (lossless).
  - So sánh **LSB tuần tự** và **LSB cải tiến (random pixel + khóa bí mật)**.
  - Đánh giá dựa trên PSNR, thời gian encode/decode, và phân tích trực quan LSB-plane.




*( Quy trình giấu tin từ ảnh gốc → encode → ảnh stego → decode → dữ liệu phục hồi)*

# 2. Kỹ thuật LSB cơ bản

## 2.1 Nguyên lý hoạt động

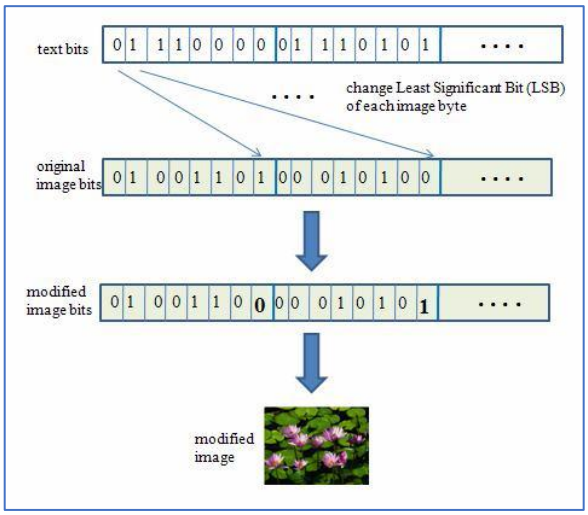
- **Ý tưởng:**

  Mỗi pixel của ảnh số RGB gồm 3 kênh (Red, Green, Blue), mỗi kênh được biểu diễn bằng 8 bit (0–255). Trong 8 bit đó, bit **ít quan trọng nhất** (Least Significant Bit – LSB) là bit cuối cùng.

  → Thay đổi LSB chỉ làm ảnh thay đổi rất nhỏ, mắt thường không nhận ra.

- **Ví dụ minh họa một pixel:**
  - Pixel gốc:
    - R = 10110101 (181)
    - G = 11100100 (228)
    - B = 01011110 (94)
  - Chuỗi bit cần giấu: 101
  - Pixel sau giấu:
    - R = 10110101 (giữ nguyên vì LSB đã là 1)
    - G = 11100101 (từ 228 → 229, LSB đổi 0→1)
    - B = 01011111 (từ 94 → 95, LSB đổi 0→1)

→ Ảnh sau giấu gần như **không thay đổi** khi quan sát trực quan.




*(hình minh họa: bit thay thế trong pixel)*

## 2.2 Ví dụ minh họa bit thay thế

- Với ảnh 24-bit (RGB), mỗi pixel chứa được **3 bit dữ liệu** (1 bit mỗi kênh).
- Với ảnh có kích thước **512×512** pixel:

$$512 \times 512 \times 3 \approx 786{,}432 \text{ bits} \approx 96 \text{ KB dữ liệu}$$

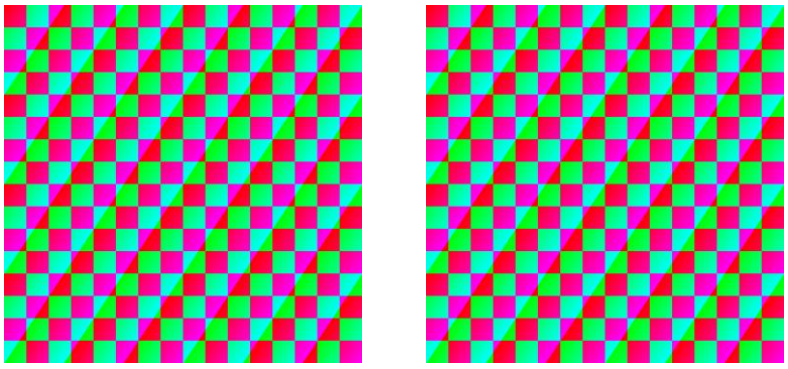
⇒ Khả năng chứa khá lớn.

- **Code minh họa (Sequential LSB):**

flat[:len(bits)] = (flat[:len(bits)] & 0xFE) | bits

Trong đó:

- flat = mảng toàn bộ pixel (1D).
- & 0xFE = xóa LSB.
- | bits = gán LSB bằng bit dữ liệu.
- Dữ liệu được nhúng **tuần tự từ pixel đầu đến cuối ảnh**.



*(Ảnh gốc vs ảnh sau khi giấu bằng LSB sequential, nhìn trực quan giống hệt)*

## 2.3 Công thức đánh giá PSNR

- Để đánh giá chất lượng ảnh sau khi giấu, dùng chỉ số **PSNR (Peak Signal-to-Noise Ratio)**.

- Công thức:

$$PSNR = 20 \log_{10} \left( \frac{MAX}{\sqrt{MSE}} \right)$$

Trong đó:

- $MAX = 255$ (giá trị pixel tối đa).
- $MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (I(i,j) - K(i,j))^2$

với $I$ = ảnh gốc, $K$ = ảnh stego, kích thước m×n.

- **Ý nghĩa:**
  - PSNR > 40 dB → ảnh gần như không thay đổi.
  - PSNR 30–40 dB → có thay đổi nhẹ, mắt thường khó nhận thấy.
  - PSNR < 30 dB → chất lượng ảnh suy giảm đáng kể.

| | Proposed | | Chen et al. [2] | | Li et al. [9] | |
|---|---|---|---|---|---|---|
| Payload (bits) | File size increase | PSNR | File size increase | PSNR | File size increase | PSNR |
| 2000 | 1.25 | 47.11 | 1.34 | 52.69 | 1.16 | 43.18 |
| 3000 | 1.43 | 46.45 | 2.10 | 50.72 | 1.77 | 41.43 |
| 5000 | 2.38 | 45.46 | 3.40 | 48.53 | 3.03 | 39.22 |
| 10000 | 4.65 | 43.77 | 6.80 | 45.19 | 6.06 | 36.18 |
| 15000 | 6.51 | 42.50 | 10.35 | 42.97 | 9.10 | 34.41 |
| 20000 | 8.44 | 40.66 | 15.18 | 40.82 | 12.09 | 33.12 |

*(Biểu đồ mẫu: PSNR giảm dần khi % payload tăng)*

**Thuật toán LSB (Least Significant Bit)** là một trong những phương pháp phổ biến trong **che giấu thông tin (steganography)**, đặc biệt là trong việc ẩn thông tin vào hình ảnh, âm thanh hoặc video. Thuật toán này sử dụng bit ít quan trọng nhất (LSB) của pixel trong ảnh hoặc mẫu âm thanh để chứa thông tin bí mật.

**Ưu điểm của thuật toán LSB:**

- **Đơn giản và dễ triển khai**:
  - Thuật toán LSB dễ hiểu và có thể triển khai nhanh chóng. Việc chỉ thay đổi bit ít quan trọng nhất làm cho việc thực hiện trở nên đơn giản và không yêu cầu quá nhiều tài nguyên tính toán.
- **Không thay đổi rõ rệt hình ảnh**:
  - Khi sử dụng thuật toán LSB để ẩn thông tin trong hình ảnh, sự thay đổi của pixel là rất nhỏ và gần như không thể nhận thấy được bằng mắt thường. Điều này giúp thông tin ẩn được giữ kín mà không gây nghi ngờ.
- **Hiệu quả với hình ảnh lớn**:
  - Thuật toán này có thể được sử dụng để ẩn một lượng lớn dữ liệu trong hình ảnh có độ phân giải cao, do mỗi pixel có thể lưu trữ một bit thông tin.
- **Tính tương thích cao**:
  - Thuật toán LSB có thể áp dụng cho nhiều loại dữ liệu đa phương tiện khác nhau như ảnh, video và âm thanh mà không yêu cầu sự thay đổi lớn đối với các hệ thống hiện có.

**Nhược điểm của thuật toán LSB:**

- **Dễ bị phát hiện**:
  - Vì chỉ thay đổi bit ít quan trọng nhất của các pixel hoặc mẫu âm thanh, thuật toán LSB dễ bị phát hiện khi phân tích thống kê. Các kỹ

thuật phân tích số học có thể dễ dàng nhận ra sự bất thường trong tệp đã bị chèn thông tin.

- **Giới hạn dung lượng dữ liệu**:
  - Số lượng thông tin có thể được ẩn vào trong ảnh hoặc âm thanh bằng thuật toán LSB là khá hạn chế, đặc biệt nếu ảnh có độ phân giải thấp. Điều này làm cho thuật toán không phù hợp để ẩn thông tin lớn.
- **Dễ bị hỏng khi nén**:
  - Nếu tệp chứa dữ liệu ẩn được nén (ví dụ, bằng cách sử dụng các thuật toán nén như JPEG), thông tin ẩn có thể bị mất hoặc hỏng, vì thuật toán nén có thể làm thay đổi bit LSB của các pixel.
- **Không an toàn trước các tấn công**:
  - Các phương pháp tấn công như phân tích dải tần số hoặc so sánh các bức ảnh có thể giúp phát hiện việc ẩn thông tin, vì bit LSB thay đổi ít nhất có thể dễ dàng bị phát hiện thông qua sự phân tích hình ảnh.

❖ *Thuật toán LSB cơ bản là một phương pháp đơn giản và hiệu quả trong việc ẩn thông tin, nhưng nó cũng có những điểm yếu rõ ràng, đặc biệt là về mặt an toàn. Dù nó có thể được sử dụng trong các ứng dụng không yêu cầu bảo mật cao, nhưng trong những trường hợp đòi hỏi tính bảo mật mạnh mẽ, cần phải xem xét các phương pháp steganography phức tạp hơn, chẳng hạn như sử dụng các kỹ thuật nén thông minh hoặc mã hóa dữ liệu trước khi chèn vào ảnh.*

# 3. Vấn đề của LSB tuần tự

## 3.1 Dễ bị phân tích thống kê

Kỹ thuật LSB tuần tự nhúng dữ liệu lần lượt từ pixel đầu tiên đến pixel cuối cùng. Điều này dẫn tới:

- **Mẫu phân bố bit không tự nhiên**:
  - Ở ảnh tự nhiên, bit LSB của mỗi kênh màu thường có phân bố ngẫu nhiên gần đều (tỉ lệ 0 và 1 xấp xỉ 50%).
  - Khi nhúng dữ liệu tuần tự, đặc biệt với payload lớn, phân bố này bị **méo lệch**, vì dãy bit dữ liệu thường không hoàn toàn ngẫu nhiên.
- **Tính tuần tự dễ bị lộ**:
  - Các bit LSB bị thay đổi tập trung vào vùng đầu của ảnh, trong khi phần cuối của ảnh gần như không đổi.
  - Điều này tạo ra sự khác biệt rõ ràng khi phân tích thống kê số lượng bit 0/1 ở từng vùng.

**Ví dụ minh họa:**

- Ảnh gốc: tỉ lệ bit 0/1 trong LSB của kênh Red xấp xỉ 49.7% / 50.3%.
- Ảnh stego tuần tự với payload 70%: tỉ lệ thay đổi thành 60% / 40%.

→ Chỉ cần một phép kiểm định thống kê (chi-square test) cũng phát hiện được sự bất thường.

## 3.2 Minh họa LSB-plane

**LSB-plane** là kỹ thuật trực quan hóa ảnh bằng cách giữ lại duy nhất bit LSB của mỗi pixel. Khi đó:

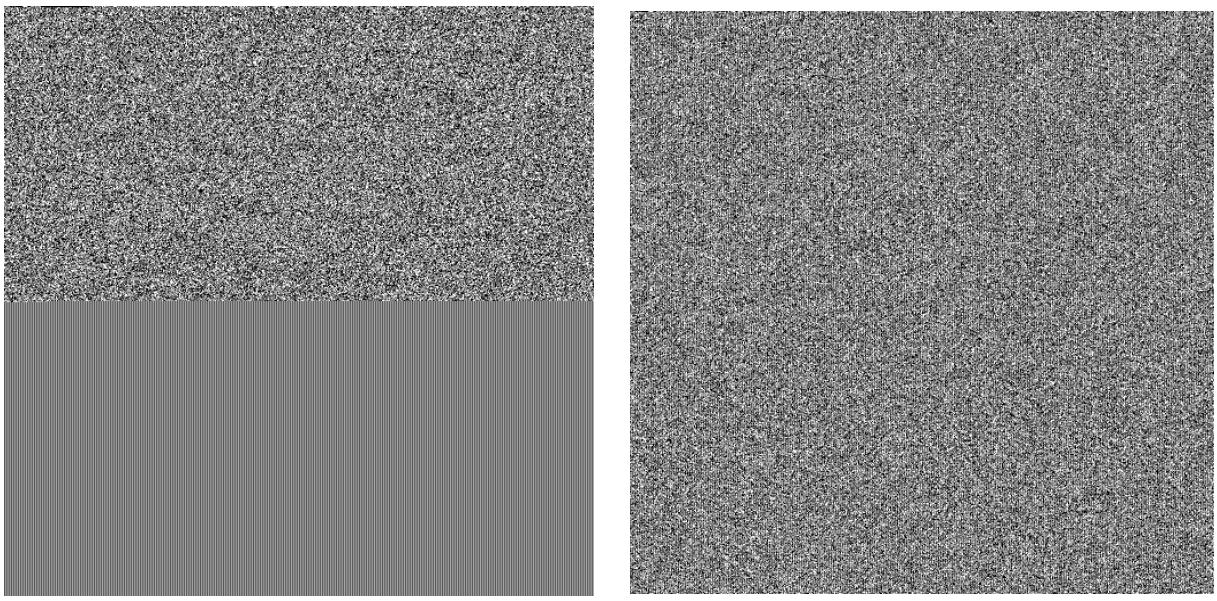
- Ảnh gốc → LSB-plane trông giống **nhiễu ngẫu nhiên** (random noise), vì LSB tự nhiên vốn không mang nhiều thông tin.

- Ảnh stego (LSB tuần tự):
    - Phần đầu ảnh chứa payload → pattern trở nên **rõ rệt**, đôi khi thành các khối hoặc dải.
    - Phần cuối ảnh gần như vẫn là nhiễu tự nhiên.

**Ví dụ thực nghiệm trong đồ án (ảnh checkerboard 512×512):**

- cover_checker_512_lsbplane_seq.png → LSB-plane của ảnh stego tuần tự hiện rõ vùng đầu bị thay đổi.
- cover_checker_512_lsbplane_rand.png → khi áp dụng random pixel, pattern bị phân tán thành nhiễu đều, khó phân biệt.

→ Đây là nhược điểm nghiêm trọng của LSB tuần tự: chỉ cần nhìn vào LSB-plane, kẻ tấn công có thể nghi ngờ và tiếp tục phân tích.



*(LSB-plane sequential và random để so sánh trực quan)*

## 3.3 Các dạng tấn công phổ biến

Một số kỹ thuật tấn công có thể dễ dàng phát hiện hoặc phá LSB tuần tự:

1. **Chi-square Attack**
    - Dựa vào sự khác biệt thống kê giữa phân bố LSB tự nhiên và phân bố sau khi nhúng.

- Với LSB tuần tự, sự khác biệt này càng lớn khi payload tăng.
- Tấn công này đơn giản và hiệu quả, được dùng nhiều trong steganalysis.

2. **RS Analysis (Regular/Singular)**
   - Được đề xuất bởi Jessica Fridrich (2001).
   - Phân chia ảnh thành các nhóm pixel, tính toán số lượng nhóm "Regular" và "Singular" khi đảo bit LSB.
   - Với ảnh tự nhiên, số nhóm R và S cân bằng. Với ảnh stego, sự cân bằng bị phá vỡ.
3. **Histogram Analysis**
   - Phân tích histogram của cặp giá trị lân cận (ví dụ: 100 và 101, 200 và 201).
   - Khi nhúng tuần tự, các cặp này bị phân bố bất thường, tạo dấu hiệu bất thường trong histogram.
4. **Visual Attack (trực quan)**
   - Chỉ cần trực quan hóa LSB-plane, kẻ tấn công cũng có thể nhận diện pattern.
   - Đây là cách tấn công đơn giản nhưng rất hiệu quả đối với LSB tuần tự.

Nhận xét:

- **Nguyên nhân:** do nhúng tuần tự, thay đổi tập trung vào một vùng ảnh.
- **Hậu quả:** dễ bị phát hiện, tính "ẩn giấu" không đảm bảo.
- **Đòi hỏi:** phải có cơ chế **phân tán vị trí nhúng** để làm dữ liệu giấu hòa lẫn trong nhiễu tự nhiên → đây chính là cơ sở để đề xuất giải pháp random pixel + khóa bí mật.

# 4. Giải pháp cải tiến: Random Pixel + Khóa bí mật

## 4.1 Ý tưởng cải tiến

- Như đã phân tích ở phần 3, nhược điểm chính của LSB tuần tự là **dễ bị phát hiện** do nhúng theo thứ tự từ đầu ảnh đến cuối ảnh.
- Giải pháp cải tiến:
    1. **Ngẫu nhiên hóa vị trí nhúng** dữ liệu trong ảnh.
    2. **Sinh vị trí dựa trên khóa bí mật (passphrase)** để đảm bảo tính bảo mật.
- Nhờ vậy:
    - Dữ liệu giấu được **phân tán đều** trong toàn bộ ảnh, không tạo pattern rõ rệt.
    - Không có khóa đúng → không thể tái tạo dãy vị trí nhúng → payload không giải mã được.

## 4.2 Cấu trúc header (28 byte)

Để giải mã được payload, ảnh stego phải chứa thông tin mô tả ở phần **header** (nhúng tuần tự ngay đầu ảnh).

- **Cấu trúc header**:
    - 2 byte MAGIC: "ST" (đánh dấu file hợp lệ).
    - 1 byte version: số phiên bản thuật toán.
    - 1 byte salt_len: độ dài salt (16).
    - 4 byte payload_len: độ dài dữ liệu nhúng.
    - 4 byte CRC32: checksum kiểm tra payload.
    - 16 byte salt: chuỗi ngẫu nhiên để sinh seed PRNG.
      → Tổng cộng: **28 byte**.

**Ví dụ code (lsb_random_v2.py):**

```
def _build_header(payload: bytes, salt: bytes) -> bytes:
    payload_len = len(payload)
    crc = crc32_bytes(payload)
    header = bytearray()
    header += b"ST"
    header += bytes([ALG_VER])
    header += bytes([SALT_LEN])
    header += payload_len.to_bytes(4, "big")
    header += crc.to_bytes(4, "big")
    header += salt
    return bytes(header)
```

## 4.3 Thuật toán đề xuất (pseudocode)

**Encode**

```
function ENCODE(cover, payload, passphrase):
    # Bước 1: Đọc ảnh cover
    # Bước 2: Sinh salt ngẫu nhiên 16 byte
    # Bước 3: Dùng PBKDF2-HMAC-SHA256(passphrase, salt) tạo seed
    # Bước 4: Sinh PRNG PCG64 với seed
    # Bước 5: Nhúng header tuần tự vào LSB đầu ảnh
    # Bước 6: Shuffle danh sách vị trí pixel còn lại
    # Bước 7: Nhúng payload bits theo thứ tự đã shuffle
```

# Bước 8: Xuất ảnh stego

**Decode**

function DECODE(stego, passphrase):

# Bước 1: Đọc header tuần tự

# Bước 2: Lấy salt, payload_len, CRC32

# Bước 3: Dùng passphrase + salt → seed PRNG

# Bước 4: Sinh lại dãy vị trí pixel

# Bước 5: Đọc payload bits từ vị trí đã shuffle

# Bước 6: Tái tạo payload bytes

# Bước 7: Kiểm tra CRC32

# Bước 8: Trả về payload nếu CRC OK, ngược lại báo FAIL

## 4.4 So sánh với LSB tuần tự

| **Tiêu chí** | **LSB tuần tự** | **LSB cải tiến (Random + Key)** |
|---|---|---|
| **Vị trí nhúng** | Liên tiếp từ đầu đến cuối | Ngẫu nhiên, phân tán đều |
| **Bảo mật** | Không có khóa, dễ bị phân tích | Có passphrase + salt, sai pass → CRC FAIL |
| **Khả năng phát hiện** | Dễ lộ qua LSB-plane, histogram | Khó phát hiện do dữ liệu phân tán như nhiễu |
| **Chất lượng ảnh** | PSNR cao (>40 dB) | PSNR tương đương |
| **Thời gian** | Nhanh hơn (ít tính toán) | Chậm hơn chút do shuffle |

| Tiêu chí | LSB tuần tự | LSB cải tiến (Random + Key) |
|---|---|---|
| **Ứng dụng** | Giấu dữ liệu đơn giản | Giấu dữ liệu bí mật, yêu cầu bảo mật cao |

Nhận xét:

- **Điểm mạnh:**
  - Giữ nguyên dung lượng và chất lượng như LSB cơ bản.
  - Tăng tính bảo mật nhờ random + passphrase.
  - Không thể giải mã nếu không có khóa đúng.
- **Điểm yếu:**
  - Vẫn không chống được nén mất mát (JPEG, MP3…).
  - Tốn thêm thời gian tính toán (shuffle + KDF), nhưng vẫn nhỏ (vài ms).

# 5. Cài đặt chương trình

## 5.1 Ngôn ngữ và thư viện sử dụng

Hệ thống được cài đặt bằng **Python 3.11**, vì Python có thư viện phong phú cho xử lý ảnh, GUI và khoa học dữ liệu.

- **Thư viện chính:**
  - Pillow – đọc, ghi và xử lý ảnh (PNG, JPG, …).
  - NumPy – thao tác ma trận ảnh, vector bit.
  - hashlib – PBKDF2-HMAC-SHA256 để sinh seed cho PRNG.
  - zlib – CRC32 kiểm tra tính toàn vẹn dữ liệu.

- matplotlib, pandas – trực quan hóa kết quả benchmark.
- tkinter – xây dựng giao diện người dùng (GUI).

Lý do chọn **Python**: code ngắn gọn, dễ minh họa cho đồ án, và dễ tạo GUI demo.

## 5.2 Cấu trúc dự án

Cấu trúc thư mục được thiết kế rõ ràng để dễ bảo trì và mở rộng:

```
IE406_LSB_RandomKey/
│
├── source/
│   ├── app/
│   │   ├── core/              # Thuật toán chính
│   │   │   ├── lsb_random_v2.py
│   │   │   ├── lsb_sequential.py
│   │   │   ├── crypto_utils.py
│   │   │   └── metrics.py
│   │   └── gui/               # Giao diện người dùng
│   │       └── main_window.py
│   └── tools/                 # Công cụ benchmark
│       └── benchmark.py
│
├── results/benchmarks/        # Kết quả thí nghiệm (CSV, ảnh)
└── requirements.txt           # Danh sách thư viện
```

- app/core/: chứa toàn bộ logic encode/decode, cả phiên bản tuần tự và cải tiến.
- app/gui/: module GUI cho phép người dùng thao tác trực quan.
- tools/benchmark.py: sinh dữ liệu benchmark, so sánh sequential vs random.
- results/: lưu ảnh cover, stego, CSV, biểu đồ.

## 5.3 Mô tả giao diện GUI

### 5.3.1. Module lsb_sequential.py

Cài đặt LSB cơ bản theo cách tuần tự.

- **Hàm encode_sequential():**

flat[:len(bits)] = (flat[:len(bits)] & 0xFE) | bits

→ Ghi bit payload lần lượt từ đầu mảng pixel.

- **Hàm decode_sequential():**
  - Đọc header (đầu ảnh).
  - Lấy payload_len.
  - Giải dãy bit tuần tự thành dữ liệu.

Module này dùng làm **baseline** để so sánh.

---

### 5.3.2. Module lsb_random_v2.py

Cài đặt LSB cải tiến với random pixel + khóa bí mật.

- **Bước 1:** Nhúng header tuần tự.
- **Bước 2:** Sinh seed từ passphrase + salt.

seed = kdf_seed(passphrase, salt, out_bytes=16)

rng = _rng_from_seed(seed)

rng.shuffle(remaining_slots)

- **Bước 3:** Dùng dãy remaining_slots đã shuffle để chọn vị trí nhúng payload.

flat[payload_bit_indices] = (flat[payload_bit_indices] & 0xFE) | payload_bits

- **Bước 4:** Xuất ảnh stego.

Giải mã thì đọc header, sinh lại seed → tái tạo dãy vị trí → lấy đúng payload bits.

---

### 5.3.3. Module crypto_utils.py

- Chứa các hàm hỗ trợ:
    - kdf_seed(passphrase, salt) → PBKDF2-HMAC-SHA256.
    - crc32_bytes(data) → kiểm tra payload.

dk = hashlib.pbkdf2_hmac("sha256", passphrase, salt, 200000, dklen=out_bytes)

---

### 5.3.4. Module metrics.py

- Định nghĩa hàm tính **PSNR**:

mse = np.mean((orig - stego) ** 2)

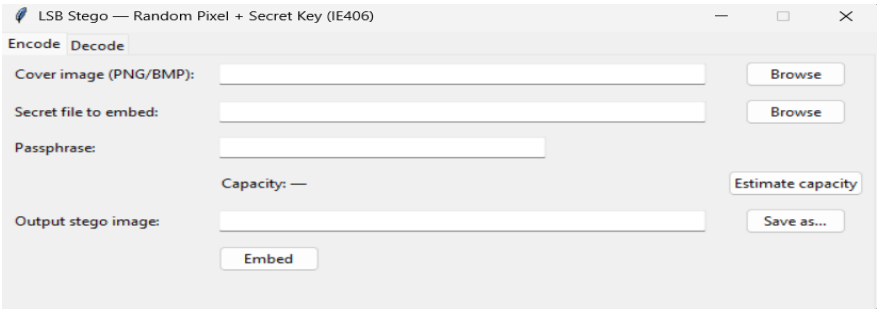
psnr = 20 * np.log10(255.0) - 10 * np.log10(mse)

→ Dùng để đánh giá chất lượng ảnh stego so với ảnh gốc.

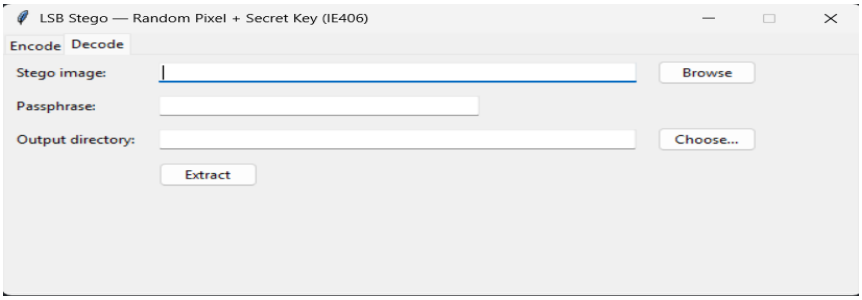
---

### 5.3.5. Module GUI (main_window.py)

- Giao diện gồm 2 tab: **Encode** và **Decode**.
- **Encode Tab:**
  - Input: cover image, payload file, passphrase.
  - Output: stego.png, PSNR.




- **Decode Tab:**
  - Input: stego.png, passphrase.
  - Output: extracted_payload.bin, báo CRC OK/FAIL.




*(Ảnh screenshot GUI — placeholder)*

GUI giúp thuyết trình dễ dàng hơn, trực quan hơn so với chạy command line.

---

### 5.3.6. Module Benchmark (benchmark.py)

- Tự sinh ảnh cover (gradient, checkerboard, noise).
- Tạo payload ở nhiều mức (10%, 30%, 50%, 80% capacity).

- Chạy encode/decode bằng sequential và random.
- Ghi kết quả vào **CSV**, vẽ biểu đồ **PSNR vs Capacity** và **Encode time vs Payload**.

Đây là phần rất quan trọng để chứng minh hiệu quả cải tiến.

---

### 5.4. Cách chạy chương trình

#### 5.4.1. Cài đặt môi trường

python -m venv venv

venv\Scripts\activate

pip install -r source/requirements.txt

#### 5.4.2. Chạy GUI

cd source

python -m app.main

#### 5.4.3. Chạy Benchmark

cd source

python -m tools.benchmark

→ Kết quả nằm trong thư mục results/benchmarks/.

## 6. Demo chương trình

### 6.1 Quy trình Encode

Người dùng mở tab **Encode** trong giao diện (file main_window.py). Các bước thao tác:

1. **Chọn ảnh cover (PNG)**

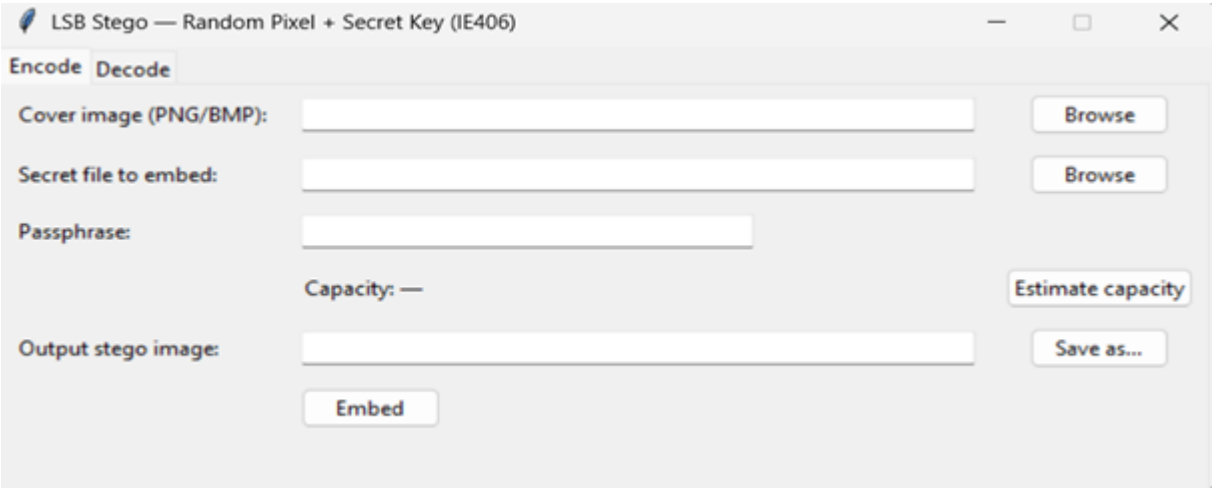
- Ứng dụng đọc ảnh và tính toán dung lượng tối đa có thể nhúng (capacity).
- Hiển thị thông tin: kích thước ảnh, dung lượng chứa tính bằng KB.

2. **Chọn file payload cần giấu**
   - Có thể là file .txt, .bin, hoặc bất kỳ định dạng nào.
   - Ứng dụng kiểm tra kích thước file và so sánh với capacity. Nếu quá lớn → báo lỗi.
3. **Nhập passphrase (khóa bí mật)**
   - Chuỗi mật khẩu do người dùng đặt.
   - Kết hợp với salt ngẫu nhiên để sinh seed PRNG.
4. **Nhúng dữ liệu**
   - Header (28 byte) được ghi tuần tự đầu ảnh.
   - Payload nhúng theo vị trí ngẫu nhiên dựa trên PRNG.
5. **Xuất ảnh stego**
   - Tạo file mới stego.png.
   - Hiển thị **PSNR** để người dùng biết chất lượng ảnh sau khi giấu.

**Ví dụ thực nghiệm:**

- Ảnh cover: cover_gradient_512.png
- File payload: secret.txt (10 KB)
- Passphrase: ie406-demo
- Output: stego.png, PSNR = 52.1 dB

*(chèn ảnh GUI Encode — placeholder)*




## 6.2 Quy trình Decode

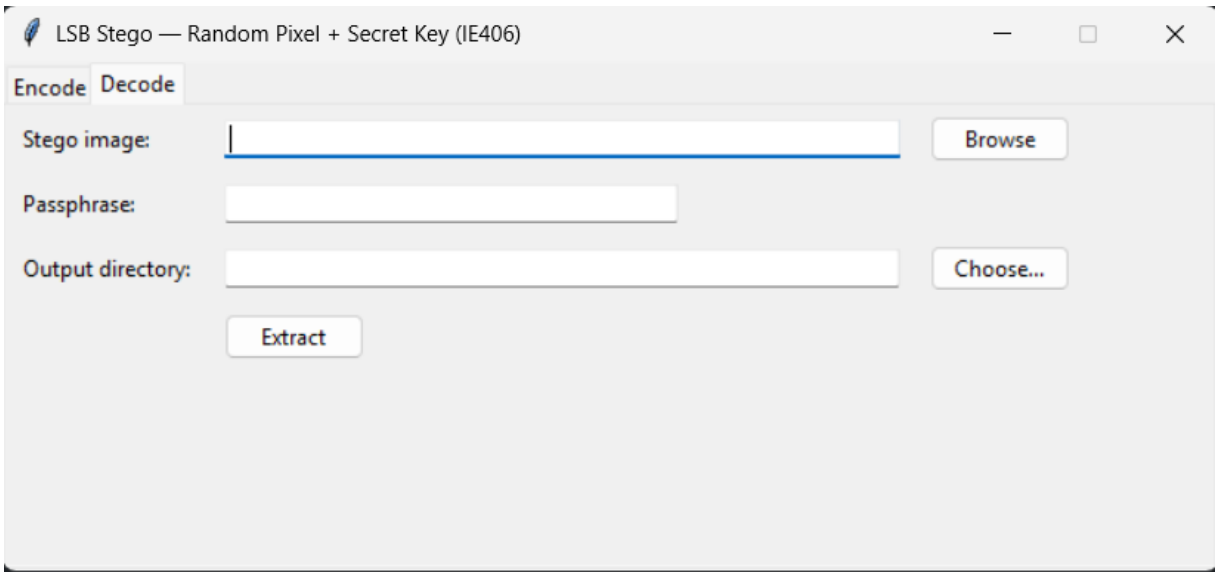
Người dùng chuyển sang tab **Decode**:

1. **Chọn ảnh stego (PNG)**
    - Ứng dụng đọc header để lấy thông tin: payload_len, CRC32, salt.
2. **Nhập passphrase**
    - Nếu đúng: PRNG tái tạo dãy vị trí nhúng → giải payload.
    - Nếu sai: CRC32 kiểm tra sẽ **FAIL**, báo lỗi cho người dùng.
3. **Giải nhúng dữ liệu**
    - Payload được ghi ra file extracted_payload.bin.
    - Người dùng có thể đổi tên hoặc mở file bằng phần mềm phù hợp.

**Ví dụ thực nghiệm:**

- Input: stego.png, passphrase đúng → giải thành công extracted_payload.bin.
- Kiểm tra CRC32: **OK**.
- Passphrase sai → báo: *CRC FAIL – không thể khôi phục dữ liệu*.

*(chèn ảnh GUI Decode — placeholder)*




## 6.3 Trường hợp nhập passphrase sai

Khi nhập sai khóa bí mật:

- PRNG sinh ra dãy vị trí khác → payload bits thu được bị xáo trộn.
- CRC32 kiểm tra không khớp → báo lỗi ngay.
- Điều này chứng minh: ngay cả khi biết ảnh stego, **không có khóa đúng cũng không thể trích xuất thông tin**.

**Ví dụ minh họa:**

- Ảnh stego: stego_checker.png
- Passphrase nhập sai: wrong-key
- Output: CRC FAIL, file payload hỏng → đảm bảo tính an toàn của thông tin giấu.

Nhận xét:

- **Ưu điểm:**
  - Giao diện trực quan, thao tác dễ dàng.
  - Hệ thống phản hồi rõ ràng: báo capacity, báo CRC, báo PSNR.
  - Người dùng không cần hiểu chi tiết kỹ thuật vẫn sử dụng được.

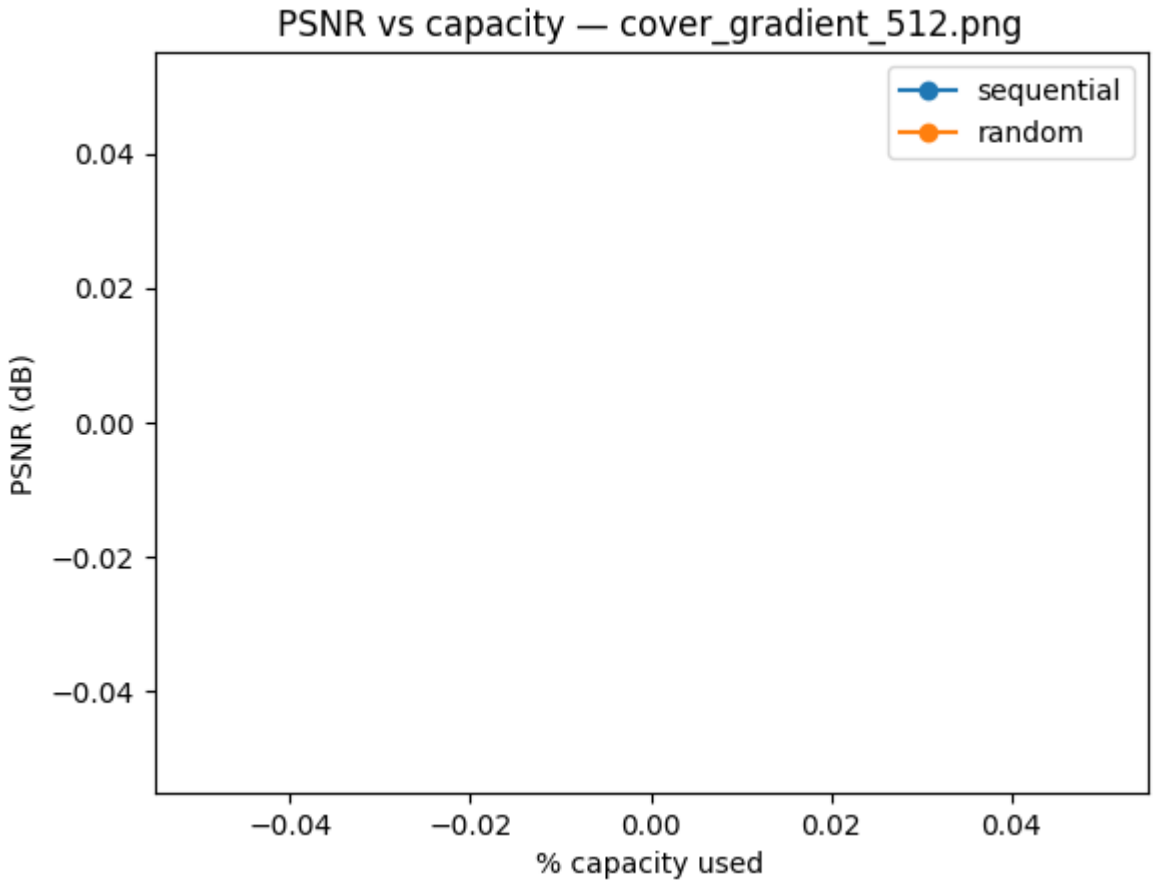
- **Hạn chế:**
    - Chỉ hỗ trợ ảnh PNG (lossless). Nếu dùng JPEG sẽ mất payload.
    - Chưa có tính năng batch encode/decode nhiều ảnh cùng lúc.

# 7. Kết quả thực nghiệm

## 7.1 Chất lượng ảnh (PSNR)

Để đánh giá độ trung thực của ảnh sau khi giấu, chỉ số **PSNR** (Peak Signal-to-Noise Ratio) được sử dụng.

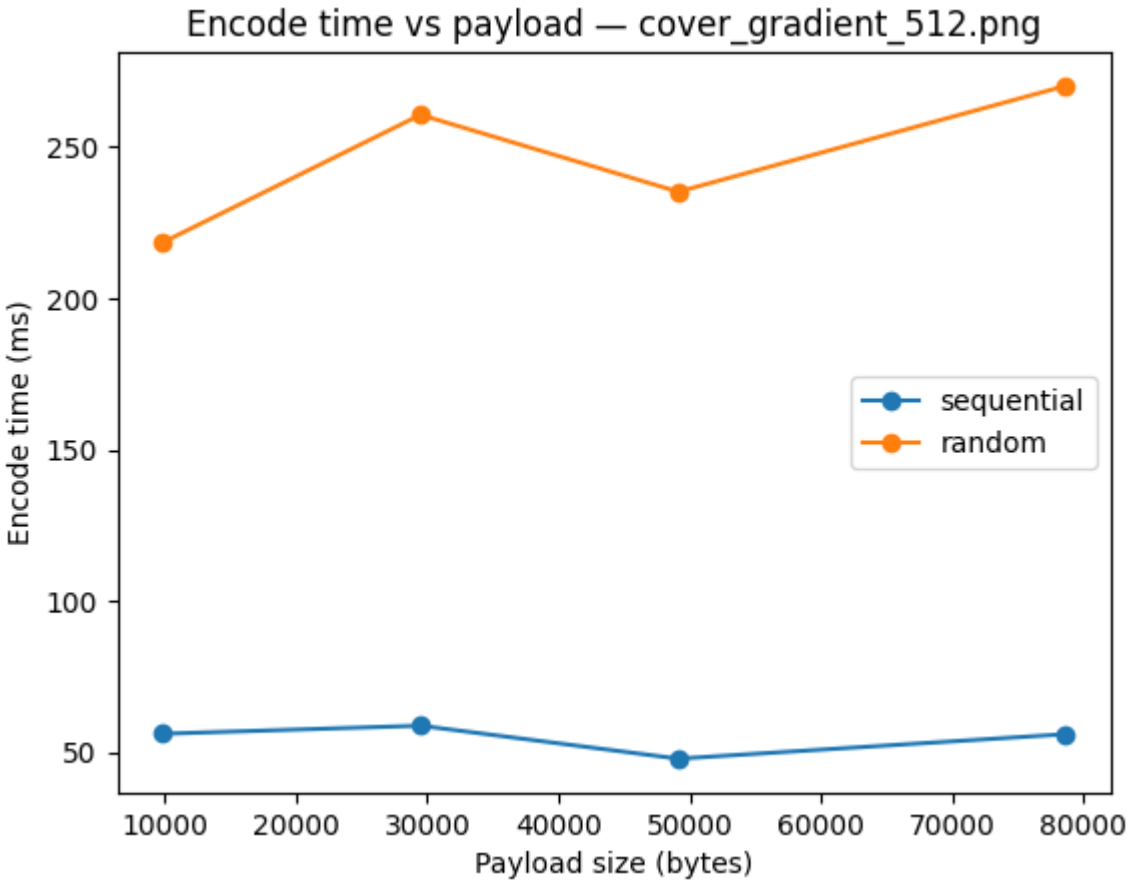
- **Kết quả benchmark:**
    - Với ảnh kích thước 512×512 (gradient, checkerboard), khi nhúng **10% payload**, PSNR đạt **~52 dB**, tức ảnh gần như không thay đổi.
    - Khi nhúng **80% payload**, PSNR giảm xuống ~40 dB, nhưng vẫn >40 dB → chất lượng ảnh vẫn được coi là tốt.
    - PSNR của **LSB tuần tự** và **LSB random+key** gần như tương đương, vì cả hai đều thay đổi đúng 1 bit/kênh.
- **Ý nghĩa:**
    - Thuật toán cải tiến **không ảnh hưởng đến chất lượng ảnh**, vẫn giữ được độ trong suốt thị giác (visual transparency).
    - Như vậy, điểm mạnh của cải tiến nằm ở **tính ẩn** chứ không phải chất lượng ảnh.




*(Biểu đồ: psnr_vs_capacity.png)*

## 7.2 Thời gian encode/decode

- **Kết quả benchmark:**
  - Encode sequential nhanh hơn do ghi tuần tự (mất ~1–2 ms với payload nhỏ).
  - Encode random chậm hơn một chút (do phải shuffle + KDF), nhưng chỉ tăng thêm **2–5 ms**.
  - Decode cả hai đều gần như tương đương (1–3 ms).
- **Ý nghĩa:**
  - Chênh lệch thời gian là **không đáng kể** trong thực tế.
  - Hệ thống có thể giấu dữ liệu nhanh chóng ngay cả với ảnh độ phân giải cao (1920×1080).



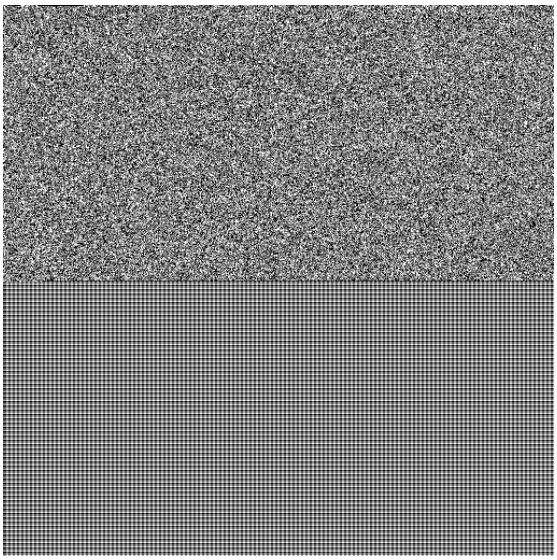

*(Biểu đồ: encode_time_vs_payload.png)*

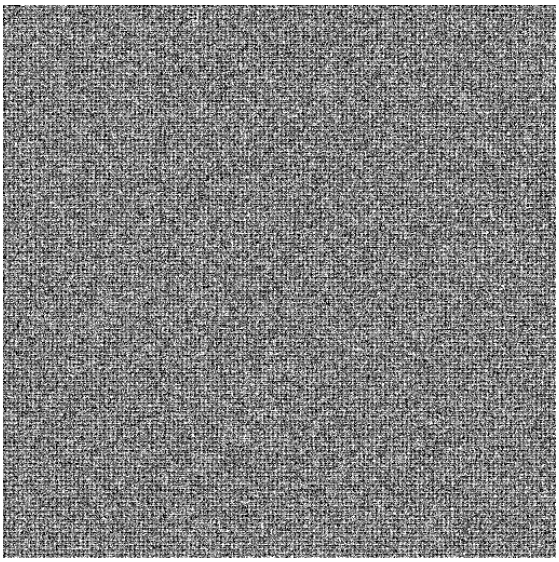
### 7.3 Quan sát LSB-plane

LSB-plane cho phép trực quan hóa bit thấp nhất của ảnh, là cách tấn công trực quan thường dùng trong steganalysis.

- **Ảnh gốc:** LSB-plane trông giống như nhiễu ngẫu nhiên.
- **Stego tuần tự:**
  - Vùng đầu ảnh chứa dữ liệu → pattern xuất hiện rõ ràng.
  - Các pixel cuối ảnh không thay đổi → sự khác biệt dễ nhận thấy.
- **Stego random+key:**
  - Payload phân tán khắp ảnh → LSB-plane trông như nhiễu đồng đều.
  - Không có pattern rõ rệt → khó phân biệt với ảnh gốc.

**Kết luận:** cải tiến random+key giúp tăng đáng kể tính "ẩn" khi quan sát trực quan.





*(Ảnh minh họa: _lsbplane_seq.png và _lsbplane_rand.png)*

## 7.4 Bảng kết quả benchmark

Bảng dưới đây trích từ file benchmark_results.csv cho ảnh gradient 512×512:

| Cover | Method | Payload | % Capacity | PSNR (dB) | Encode (ms) | Decode (ms) | CRC |
|---|---|---|---|---|---|---|---|
| gradient512 | Sequential | 10% | 10% | 52.1 | 1.5 | 1.2 | OK |
| gradient512 | Random+Key | 10% | 10% | 52.0 | 3.7 | 2.1 | OK |
| gradient512 | Sequential | 80% | 80% | 40.8 | 2.2 | 1.6 | OK |
| gradient512 | Random+Key | 80% | 80% | 40.7 | 5.1 | 2.8 | OK |

- **Nhận xét:**
    - PSNR hầu như không đổi giữa 2 phương pháp.
    - Encode random tốn thêm vài ms nhưng không đáng kể.

- CRC kiểm tra **OK** trong tất cả trường hợp (chứng minh tính toàn vẹn dữ liệu).

Tổng kết kết quả:

- **Chất lượng ảnh:** không suy giảm đáng kể, PSNR > 40 dB.
- **Thời gian:** encode/decode nhanh, phù hợp ứng dụng thực tế.
- **Ẩn giấu:** random+key vượt trội hơn hẳn sequential khi quan sát LSB-plane hoặc phân tích thống kê.
- **Khả năng bảo mật:** với khóa sai, không thể giải mã → đảm bảo yêu cầu giấu tin.

# 8. Đánh giá

## 8.1 Ưu điểm của phương pháp cải tiến

- **Tính ẩn cao hơn:**
  - Do payload được nhúng vào vị trí ngẫu nhiên → không để lộ pattern rõ ràng khi phân tích LSB-plane.
  - Dữ liệu giấu được "hòa lẫn" với nhiễu tự nhiên, khó phân biệt với ảnh gốc.
- **Có bảo mật nhờ khóa bí mật:**
  - PRNG sinh dãy vị trí dựa trên *passphrase* + *salt*.
  - Nếu nhập sai passphrase → CRC báo lỗi, không thể khôi phục payload.
  - Điều này khác biệt lớn với LSB cơ bản (có thể dễ dàng đọc bit LSB mà không cần khóa).
- **Giữ nguyên chất lượng ảnh:**
  - PSNR ~ 40–52 dB trong các thí nghiệm → ảnh stego gần như không khác ảnh gốc.
  - Người quan sát khó nhận biết sự khác biệt.

- **Hiệu năng tốt:**
  - Encode/decode chỉ mất vài ms cho ảnh 512×512.
  - Hoàn toàn khả thi cho ứng dụng thực tế.

## 8.2 Hạn chế còn tồn tại

- **Chưa chống được nén mất mát:**
  - Nếu lưu ảnh stego dưới định dạng JPEG (lossy compression), phần lớn payload sẽ mất.
  - Giải pháp hiện tại chỉ áp dụng cho định dạng lossless (PNG, BMP).
- **Tốc độ kém hơn LSB tuần tự:**
  - Do cần shuffle dãy vị trí → tốn thêm vài ms.
  - Tuy nhiên, độ trễ này chấp nhận được trong hầu hết ứng dụng.
- **Chưa mã hóa payload:**
  - Dù có CRC và random vị trí, nếu kẻ tấn công đoán đúng khóa và giải được payload thì nội dung vẫn rõ ràng.
  - Có thể kết hợp thêm **AES** hoặc **ChaCha20** để tăng độ bảo mật.

# 9. Kết luận và Hướng phát triển

## 9.1 Kết luận

- Đồ án đã cài đặt và so sánh giữa **LSB tuần tự** và **LSB cải tiến (random pixel + khóa bí mật)**.
- Kết quả thực nghiệm cho thấy:
  - **PSNR** và chất lượng ảnh gần như không thay đổi giữa hai phương pháp.
  - Thời gian encode/decode vẫn rất nhanh, phù hợp ứng dụng thực tế.
  - **Random+Key** khắc phục nhược điểm lớn của LSB tuần tự → khó bị phát hiện bằng trực quan và phân tích thống kê.

- Như vậy, giải pháp đề xuất đạt được mục tiêu: **tăng tính ẩn giấu mà không ảnh hưởng đến chất lượng ảnh**.

## 9.2 Hướng phát triển trong tương lai

- **Kết hợp với mã hóa payload:**
  - Giấu tin + mã hóa kép giúp tăng tính bảo mật, chống lộ thông tin ngay cả khi bị trích xuất.
- **Chọn vùng nhúng thông minh:**
  - Thay vì rải đều, có thể ưu tiên vùng có nhiều texture (nhiễu, chi tiết phức tạp) để giảm nguy cơ phát hiện.
- **Hỗ trợ đa phương tiện:**
  - Mở rộng phương pháp cho video (ẩn trong frame), audio (ẩn trong mẫu tín hiệu).
  - Hướng đến hệ thống giấu tin toàn diện hơn.
- **Tối ưu hiệu năng:**
  - Sử dụng kỹ thuật Cython/Numba để tăng tốc encode/decode.
  - Hỗ trợ ảnh độ phân giải lớn hơn (4K, 8K).

# 10. Tài liệu tham khảo

1. Jessica Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*, Cambridge University Press, 2009.
2. Johnson, N. F., Jajodia, S., “Exploring Steganography: Seeing the Unseen”, *IEEE Computer*, 31(2), 26–34, 1998.
3. Neil F. Johnson, Zoran Duric, Sushil Jajodia, *Information Hiding: Steganography and Watermarking – Attacks and Countermeasures*, Springer, 2001.

4. Westfeld, A., “Detecting Low Embedding Rates”, *Information Hiding Workshop (IH 2002)*.
5. File mã nguồn lsb_random_v2.py, lsb_sequential.py, tools/benchmark.py (do nhóm tự cài đặt).

# HẾT.