TP1 – Stegano Mission: hiding a message in an image

Proposed Scenario

You work in pairs: **Alice & Bob**.
Your mission is to exchange secret messages by using images that have been modified in an imperceptible way.
To do so, you will conceal text inside the **Least Significant Bits (LSB)** of the image's pixels.

The goal of this exercise is to understand **LSB-based steganography**.

Objectives

You must be able to:

- Implement an **encoding function**
- Hide a message inside an image
- Implement a **decoding function**
- Extract a hidden message from an image
- Implement an **analysis function**
- Verify the **visual and statistical impact** of data hiding
- Identify the **limitations** of the technique

Part 1 – Introduction

Instruction:
Read the annexes (1 to 3) before starting the practical work.

Part 2 – Hiding a Message Inside an Image

You will embed a text message inside a PNG image.

Instructions :

1. Develop the encode_message() function
2. Develop the decode_message() function
3. Write a Markdown operational guide explaining how to use both functions
4. Find and download a simple image of your choice
5. Use the encode_message() function to hide your message
6. Add a 00000000 terminator at the end of the message
7. Generate the final image containing the hidden data
8. Send the image *and* the decoding function + operational guide to your partner

Part 3 – Extracting the Message

You receive a modified image and your partner's decoding function.

Instructions :

1. Use the provided decode_message() function
2. Have your partner validate that the extracted message is correct

Part 4 – Analysis

Instructions :

1. Develop an image comparison function
2. Visually compare the two images
3. Compute the number of modified bits
4. Compute the MSE (Mean Squared Error) between the two images
5. Comment on whether the modification is visible
6. Generate a heatmap showing the modified pixels

Part 5 – Limitations of Steganography

Write a short paragraph explaining:

- Why steganography alone does *not* guarantee confidentiality
- How an attacker may detect a hidden message
- Which image formats can destroy the hidden data

Deliverables

1. Complete encoding script
2. Complete decoding script
3. Original image
4. Image containing the hidden message
5. The extracted message
6. Analysis script
7. Markdown report (max. 1 page) including:
    - summary of the principle
    - results of the image analyses (yours + your partner's)
    - identified limitations

Development recommendation:
You may use either a Python project or a Jupyter Notebook.
The project may be submitted through GitHub.

ANNEX 1 – Understanding the LSB (Least Significant Bit)

1. What is a bit?

A bit is the fundamental unit of digital information.
It can only take two values: 0 or 1.
Digital systems represent everything, text, images, audio, with millions or billions of bits.

2. One byte

A byte contains 8 bits.

Example: 10110101

Each bit contributes differently to the value of the byte.

3. Most Significant Bit vs. Least Significant Bit

In a byte:

- The leftmost bit is the Most Significant Bit (MSB)
- The rightmost bit is the Least Significant Bit (LSB)

Example:

1 0 1 1 0 1 0 1

↑ ↑

MSB LSB

Why "significant"?

Because each bit has a different weight:

- MSB → large impact (128 in an 8-bit value)
- LSB → minimal impact (1 unit)

Changing the LSB has almost no effect on the byte's visual or numerical value.

4. The LSB in an Image Pixel

A color image (RGB) is composed of pixels.
Each pixel = 3 bytes:

- Red (R)
- Green (G)
- Blue (B)

Example :

R = 11001100

G = 10011101

B = 01010110

The displayed color depends on *all* these bits.

5. Why does steganography use the LSB?

Because changing the LSB barely affects the pixel's color.

Example:

Before: R = 11001100
After : R = 11001101

Only the last bit changes. **Visually:**
→ The pixel looks identical
→ The modification is invisible

6. How to hide a message using the LSB?

Steps:

1. Convert the message into binary (ASCII)
    - Example : "A" → 01000001
2. Insert each bit into consecutive pixel LSBs
3. For each pixel, replace the LSB of R, G, B

Since each pixel provides 3 bits, an image of 800×600 pixels provide:

800 × 600 × 3 = 1,440,000 bits ≈ 180,000 characters

This is more than enough to hide any reasonable message.

7. How to recover the message?

1. Traverse each pixel
2. Extract the LSBs of R, G, B
3. Reconstruct bytes of 8 bits
4. Stop when you reach 00000000 (end-marker)

ANNEX 2 – Development Guidelines

Part A – encode_message() Function

Objective : Write a function that:

1. Loads an image
2. Converts a text message into bits
3. Replaces pixel LSBs with these bits
4. Saves the modified image

1. Load an image with PIL (Pillow)

Use: Image.open() ,convert to "RGB" mode if needed. Access pixel values using: pixels[x, y] # (R, G, B)

2. Convert a text message to bits

Guidelines:

- Convert each character to its ASCII code
- Convert the code to an 8-bit binary string
- Concatenate all bits into a single sequence
- Add the 00000000 terminator

3. Check image capacity

Compute: capacity = width × height × 3 # total number of LSBs

Ensure that the message fits inside the image. If not, raise a clear error.

4. Insert bits into the image

For each pixel: Extract (R, G, B) / Modify each LSB / Write updated values back to pixels[x, y]

Stop when all message bits have been inserted.

5. Save the modified image

Expected result:

- A valid image containing the hidden message
- The image must look visually identical to the original

Part B – decode_message() Function

Objective: Write a function that:

a. Reads all pixel LSBs
b. Reconstructs bytes
c. Converts bytes to text
d. Stops at the first 00000000

Steps

1. Load the image and traverse pixels in the same order as encoding
2. Extract LSBs: bit = R & 1
3. Group bits into 8-bit blocks
4. Stop at "00000000"
5. Convert each byte to a character
6. Build the output message

Expected result:

- Correct message reconstruction
- No extra or invalid characters

Part C – analyse_image() Function

Objective : Develop functions that:

- Compare two images
- Count modified bits
- Compute MSE
- Display a heatmap of modifications

Bonus Options

Encoding:

- Handle message-too-large errors
- Implement global error handling
- Add unit tests

Decoding:

- Raise an error if no terminator is found
- Protect against incomplete data
- Return the number of extracted bits

ANNEX 3 – Required Libraries

1. Pillow (PIL)

Standard library for:

- Opening images (Image.open())
- Accessing pixels (pixels[x, y])
- Editing pixel values
- Saving images (.save())

Essential for LSB manipulation.

2. NumPy

Useful for:

- High-speed pixel array manipulation
- Converting images to matrices
- Fast comparison (for analysis)
- Heatmap generation

3. Matplotlib

Useful for:

- MSE visualisation
- LSB histograms
- Heatmaps

4. os / pathlib (standard libraries)

Used to:

- Manage file paths
- Check file existence
- List files in a directory

Helps avoid FileNotFoundError.