Laboratory Work №3 – Hidden Data in Own Files (LSB Steganography)

1. Practical Part

In this laboratory work I implemented a simple steganography tool that hides a text message inside a digital image using the Least Significant Bit (LSB) method.

Step 1 – Preparing the environment:
- Installed Python 3.13 on Windows.
- Installed the Pillow library with the command:
  pip install pillow

Step 2 – Preparing the files:
- Chose an image file with personal content (photo of a place I like) and saved it as input.jpg in the folder C:\Users\User.
- Created a new Python file stego_lab3.py in the same folder.

Step 3 – Implementing the program:
- Implemented functions to convert text to bits and bits back to text.
- Implemented a hide_message function that opens the input image, converts it to RGB, and then replaces the least significant bit of the blue channel of each pixel with bits of the message.
- Implemented an extract_message function that reads the LSB of the blue channel from every pixel, reconstructs the bit sequence and converts it back to text until a special end marker is found.

Step 4 – Hiding the message:
- Ran the program from the command line with:
  py -3.13 stego_lab3.py
- Chose option 1 (Hide message).
- Entered input.jpg as the original image and output.png as the stego image.
- Left the message field empty so the program used the default message with my name and date of birth.
- The program printed the message: "[+] Message hidden in output.png", which means the text was successfully embedded into the new image.

Step 5 – Extracting the message:
- Ran the program again and chose option 2 (Extract message).
- Entered output.png as the stego image.
- The program printed the line "[+] Extracted message:" followed by the same personal data that was hidden before. This confirms that the message was correctly extracted from the modified image.

Step 6 – Visual comparison (optional for screenshots):
- Compared input.jpg and output.png visually. There was no visible difference to the human eye, because changing only the least significant bit of a colour channel does not significantly affect the appearance of the image.

2. Technical Specification

Purpose of the program:
The program implements simple image steganography using the Least Significant Bit method. It can hide a short text message inside an RGB image and later extract this message.

Input data:
- Original image file (input.jpg or any other RGB image).
- Text message to hide (string). If the user leaves the field empty, the program uses a default message containing the student's name and date of birth.
- Stego image filename (output.png).

Output data:
- New image file (output.png) with the hidden message embedded in the least significant bits of the blue channel.
- When extracting, the recovered text message printed to the console.

Algorithms used:
1. Text to bits conversion:
  - The text is encoded in UTF-8.
  - Each byte is converted to 8 bits and stored in a list of integers (0 or 1).

2. Bits to text conversion:
  - The bit sequence is grouped into bytes (8 bits).
  - Each byte is converted back to an integer and then to a UTF-8 string.

3. Hiding the message (hide_message):
  - The program appends a special end marker string "<END>" to the message so that it knows where the message finishes when extracting.
  - The full message is converted to bits.
  - The image is opened with Pillow and converted to RGB mode.
  - Capacity check: the program ensures that the number of bits in the message does not exceed the number of pixels in the image (because one bit is stored in the blue channel of each pixel).
  - For each pixel, the least significant bit of the blue component is replaced with the current bit from the message. This is done by clearing the LSB with & 0b11111110 and then adding the new bit.
  - The modified image is saved under the stego filename (output.png).

4. Extracting the message (extract_message):
  - The stego image is opened and converted to RGB.
  - The program iterates over all pixels and collects the least significant bit from the blue channel of each pixel.
  - The resulting bit list is converted back to text.
  - The program searches for the end marker "<END>". Everything before this marker is returned as the hidden message.

Used technologies and libraries:
- Python 3.13 on Windows.
- Pillow (PIL) library for opening, manipulating and saving images.

Limitations:
- The capacity of the method is limited: it can hide only as many bits as there are pixels in the image.
- The method is not cryptographically secure. If an attacker suspects that LSB steganography is used, they can try to read the LSBs directly.

3. Program Code (stego_lab3.py)

```
from PIL import Image

END_MARKER = "<END>"
DEFAULT_MESSAGE = "Your Name, 01.01.2000"  # change to your real data



def text_to_bits(text: str) -> list[int]:
    data = text.encode("utf-8")
    bits = []
    for byte in data:
        for i in range(8):
            bits.append((byte >> (7 - i)) & 1)
    return bits



def bits_to_text(bits: list[int]) -> str:
    if len(bits) % 8 != 0:
        bits = bits[: len(bits) - (len(bits) % 8)]

    bytes_out = bytearray()
    for i in range(0, len(bits), 8):
        byte = 0
        for b in bits[i : i + 8]:
            byte = (byte << 1) | b
        bytes_out.append(byte)

    return bytes_out.decode("utf-8", errors="ignore")



def hide_message(input_image_path: str, output_image_path: str, message: str) -> None:
    img = Image.open(input_image_path)
    img = img.convert("RGB")
    pixels = img.load()

    full_message = message + END_MARKER
    bits = text_to_bits(full_message)

    width, height = img.size
    capacity = width * height  # using one channel (blue)

    if len(bits) > capacity:
        raise ValueError(
```

```
            f"Message too long. Bits: {len(bits)}, capacity: {capacity}"
        )

    bit_index = 0
    for y in range(height):
        for x in range(width):
            if bit_index >= len(bits):
                break

            r, g, b = pixels[x, y]
            new_b = (b & 0b11111110) | bits[bit_index]
            pixels[x, y] = (r, g, new_b)

            bit_index += 1
        if bit_index >= len(bits):
            break

    img.save(output_image_path)
    print(f"[+] Message hidden in {output_image_path}")



def extract_message(stego_image_path: str) -> str:
    img = Image.open(stego_image_path)
    img = img.convert("RGB")
    pixels = img.load()

    width, height = img.size
    bits = []

    for y in range(height):
        for x in range(width):
            r, g, b = pixels[x, y]
            bits.append(b & 1)

    text = bits_to_text(bits)
    end_index = text.find(END_MARKER)

    if end_index == -1:
        print("[!] END marker not found.")
        return text

    return text[:end_index]



def main():
    print("=== Lab 3: LSB Steganography ===")
    print("1) Hide message")
    print("2) Extract message")
    choice = input("Choose option (1/2): ").strip()

    if choice == "1":
        input_img = input("Path to original image (e.g. input.png): ").strip()
```

```
        output_img = input("Path to stego image (e.g. output.png): ").strip()
        msg = input("Message to hide (leave empty to use default): ").strip()
        if not msg:
            msg = DEFAULT_MESSAGE
        hide_message(input_img, output_img, msg)

    elif choice == "2":
        stego_img = input("Path to stego image: ").strip()
        message = extract_message(stego_img)
        print("\n[+] Extracted message:")
        print(message)
    else:
        print("Invalid choice.")


if __name__ == "__main__":
    main()
```

4. Presentation File (Structure)

The presentation for Laboratory Work №3 can contain the following slides:

Slide 1 – Title:
- Title of the lab: "Hidden Data in Own Files – LSB Steganography".
- Student's name, group, date.

Slide 2 – Goal of the Work:
- Briefly describe the aim: to study basic steganography techniques and implement hiding and extracting a text message in an image using the LSB method.

Slide 3 – Theory:
- Explain what steganography is.
- Explain the idea of the Least Significant Bit method with a simple example.

Slide 4 – Implementation:
- Show the main structure of the program (functions text_to_bits, hide_message, extract_message).
- Mention that the program is written in Python using the Pillow library.

Slide 5 – Demonstration:
- Show a screenshot of the original image (input.jpg).
- Show a screenshot of the stego image (output.png).
- Show a screenshot of the console where the message is hidden and then extracted.

Slide 6 – Conclusions:
- Summarise the results: the message was successfully hidden and extracted, the image did not change visually, LSB steganography has limited capacity and is not cryptographically secure.