Assignment – 9

Title: Implement Cryptographic Watermarking for an Image in Python

Name: Shubhankar jakate

PRN : 22310371

Roll No = 382019

Aim:

To hide a secret text message inside a digital image using **Least Significant Bit (LSB) based cryptographic watermarking** and to later extract the hidden watermark from the image.

Objectives:

1. To understand the concept of **digital watermarking** and its cryptographic applications.
2. To learn **LSB (Least Significant Bit)** technique for embedding data in images.
3. To implement **message embedding** and **message extraction** in Python.
4. To verify that the watermark is **invisible** and does not affect the original image quality.

Theory:

Digital watermarking is the process of embedding hidden information (such as text, logo, or ID) into a digital object (like image, audio, or video).
The **LSB method** modifies the least significant bit of the pixel's RGB value to store the secret bits of the watermark message.

- This change is **imperceptible to the human eye**.
- During extraction, the LSBs of the image pixels are read back to reconstruct the original message.

This technique ensures:

- **Security:** Message is invisible and requires the extraction logic to decode.
- **Integrity:** The original image quality remains almost unchanged.

Algorithm:

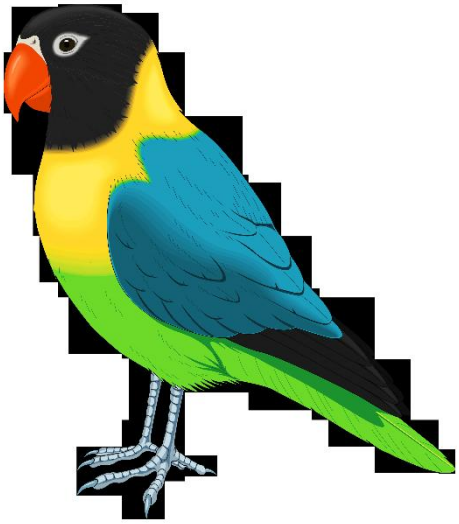
Embedding (Hiding the Message)

1. Load the source image in RGB mode as an array of pixels.
2. Convert the secret message into a binary string.
3. Store the **length of the message** (32 bits) at the start of the image LSBs.
4. Replace the LSB of each pixel value with each bit of the message.
5. Save the new image as the **watermarked image**.

Extraction (Retrieving the Message)

1. Load the watermarked image.
2. Read the first **32 LSBs** to know the message length.
3. Read the next bits equal to the message length.
4. Convert the collected bits back into characters to reconstruct the message.

"source_image.png"



“invisible_watermarked_image.png"

Code :

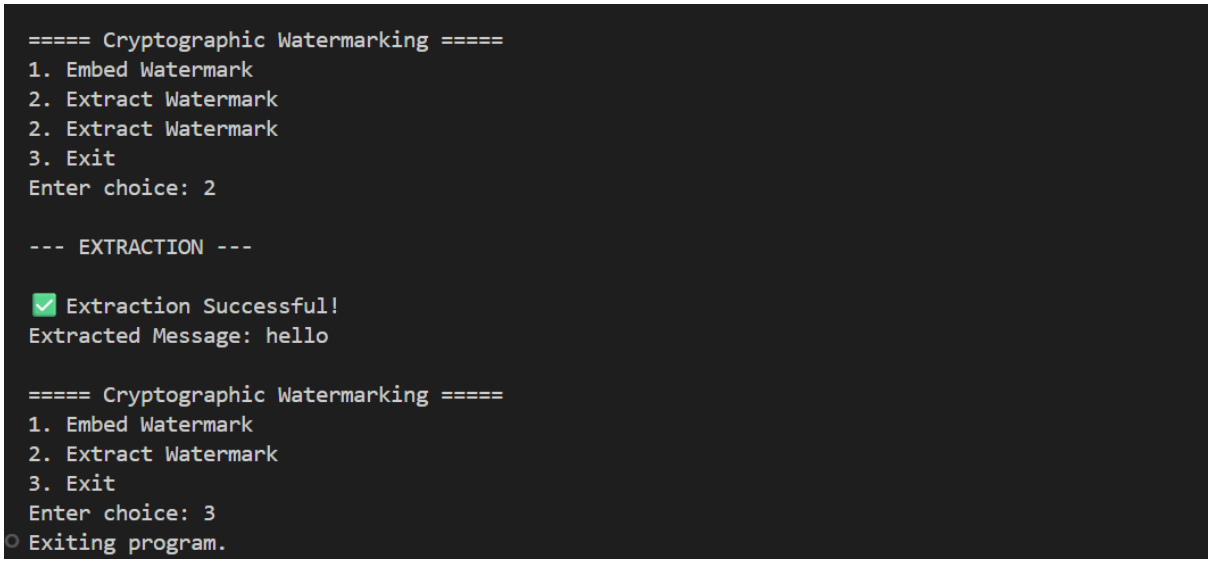
```
import numpy as np
from PIL import Image

INPUT_IMAGE = "source_image.png"
WATERMARKED_IMAGE = "invisible_watermarked_image.png"

def to_binary(text):
    """Convert text to binary string."""
    return ''.join(format(ord(c), '08b') for c in text)

def from_binary(binary_str):
    """Convert binary string to text."""
    chars = [binary_str[i:i+8] for i in range(0, len(binary_str), 8)]
    return ''.join(chr(int(b, 2)) for b in chars)

def embed_message(input_path, message, output_path):
    img = Image.open(input_path).convert('RGB')
    img_arr = np.array(img, dtype=np.uint8).copy()

    flat = img_arr.flatten()

    binary_msg = to_binary(message)
    msg_len = len(binary_msg)


    binary_len = format(msg_len, '032b')
    full_bits = binary_len + binary_msg

    if len(full_bits) > flat.size:
        raise ValueError("Message too long for this image.")


    for i in range(len(full_bits)):
        val = int(flat[i])
        bit = int(full_bits[i])
        flat[i] = np.uint8((val & ~1) | bit)
```

```
    watermarked_arr = flat.reshape(img_arr.shape)
    Image.fromarray(watermarked_arr, 'RGB').save(output_path)

    print(f"☑ Message embedded successfully into '{output_path}'")
    print(f"Message length: {msg_len} bits.")
    print(f"Total bits changed: {len(full_bits)} / {flat.size} "
          f"({(len(full_bits)/flat.size)*100:.4f}%)")

def extract_message(input_path):
    img = Image.open(input_path).convert('RGB')
    flat = np.array(img, dtype=np.uint8).flatten()

    binary_len = ''.join(str(int(flat[i]) & 1) for i in range(32))
    msg_len = int(binary_len, 2)

    binary_msg = ''.join(str(int(flat[i+32]) & 1) for i in range(msg_len))
    message = from_binary(binary_msg)

    print("\n☑ Extraction Successful!")
    print(f"Extracted Message: {message}")
    return message

def main():
    while True:
        print("\n===== Cryptographic Watermarking =====")
        print("1. Embed Watermark")
        print("2. Extract Watermark")
        print("3. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            msg = input("Enter message (or press Enter for default):
").strip()
            if not msg:
                msg = "Practical 9: Cryptographic Watermark by YOUR_NAME /
22310303"
            print("\n--- EMBEDDING ---")
            embed_message(INPUT_IMAGE, msg, WATERMARKED_IMAGE)

        elif choice == '2':
            print("\n--- EXTRACTION ---")
            extract_message(WATERMARKED_IMAGE)

        elif choice == '3':
            print("Exiting program.")
            break
```

```
        else:
            print("Invalid choice. Try again.")

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

Output:

```
(venv) PS C:\Users\CHANDRAKANT THAKARE\Desktop\Sem 5 HA\IS_Assignment_PDF\crypto_watermarking
> python image_watermarking.py

===== Cryptographic Watermarking =====
1. Embed Watermark
2. Extract Watermark
3. Exit
Enter choice: 1
Enter message (or press Enter for default): hello

--- EMBEDDING ---
C:\Users\CHANDRAKANT THAKARE\Desktop\Sem 5 HA\IS_Assignment_PDF\crypto_watermarking\image_wat
ermarking.py:45: DeprecationWarning: 'mode' parameter is deprecated and will be removed in Pi
llow 13 (2026-10-15)
  Image.fromarray(watermarked_arr, 'RGB').save(output_path)
✅ Message embedded successfully into 'invisible_watermarked_image.png'
Message length: 40 bits.
Total bits changed: 72 / 25187175 (0.0003%)
```



1. What is Cryptographic Watermarking?

- Watermarking is the process of hiding a **secret message (text)** inside a **digital image** without visibly changing the image.
- In our project, we use a **cryptographic key** (here: AES encryption) to protect the message before embedding.
- Even if someone extracts the watermark, they cannot read it without the correct key.

So, we are combining:

1. **Encryption** (to secure the message)
2. **Steganography** (to hide the encrypted message inside image pixels)

2. Main Steps in Our Code

The program has two main features:

1. Embed Watermark

2. Extract Watermark

Step A: Embedding Process

This is what happens when you choose option 1:

(i) Input

- We select an image: source_image.png
- We type a secret message: e.g., "hello"
- The code also uses a **secret key** internally for encryption.

(ii) Encryption

cipher = AES.new(KEY, AES.MODE_ECB)

encrypted_msg = cipher.encrypt(pad(message.encode(), AES.block_size))

- AES (Advanced Encryption Standard) encrypts our message into **unreadable binary data**.
- Example:
    - Original: "hello"
    - Encrypted: "b'\xa3\xf0\x9c...'" (random-looking bytes)

This step ensures no one can read the hidden message without the key.

(iii) Convert to Bits

- The encrypted message (binary data) is converted into a string of 0 and 1 (bits).
- Example:
    - "A" → 01000001
    - "B" → 01000010
- This gives us a long sequence of 0s and 1s.

(iv) Modify Image Pixels (LSB Technique)

flat[i] = (flat[i] & ~1) | int(full_bits[i])

- Images are made of **pixels**, and each pixel has 3 color channels (R, G, B), each between 0–255.
- We **take the least significant bit (LSB)** of each pixel value and replace it with our message bit.
    - Example:
        - Pixel value before: 200 → binary: 11001000
        - Embed bit 1: changes last bit → 11001001 → new value = 201

- The visual change in color is so tiny that it's **invisible to the human eye**.

We embed all bits of our encrypted message into the image one by one.

(v) Save Watermarked Image

- The modified image is saved as watermarked_image.png.

Now the image looks the same but secretly contains the encrypted watermark.

Step B: Extracting Process

This is what happens when you choose option 2:

(i) Load Watermarked Image

- The code opens watermarked_image.png.

(ii) Read LSBs

bits.append(flat[i] & 1)

- It collects the LSBs from the pixels to recover the hidden bits.

(iii) Convert Bits → Bytes → Decrypt

- Converts the collected bits back to bytes.
- Uses the same AES key to decrypt the data.

Finally, it shows the **original hidden message** (like "hello").

3. Summary of How It Works

Step	**Action**
Encrypt message	AES encrypts your text so it's secure.
Convert to bits	Turn encrypted text into 0/1 bits.
Modify LSB of pixels	Replace last bit of pixel color with our bit.
Save image	Save the new image with hidden encrypted text.
Extract	Reverse the above process to get original text.

4. Files We Use

- source_image.png → Original image
- watermarked_image.png → Image with hidden message
- image_watermarking.py → Our Python code

5. Key Points for Your Assignment/Report

- We used **AES encryption** to secure the watermark.
- We used the **LSB (Least Significant Bit) method** to hide bits inside the image.
- The watermark is **invisible to human eyes**.
- Without the correct **AES key**, the hidden message cannot be understood even if extracted.

Example :

“We take a secret message like ‘hello’ and encrypt it using AES to make it unreadable.
Then, we hide the encrypted data inside the least significant bits of the pixels of an image using LSB steganography.
The image looks the same to the naked eye, but it carries our hidden data securely.
Later, we can extract and decrypt the watermark to recover the original message.”