UNIVERSITI TEKNOLOGI MARA

CSC662

COMPUTER SECURITY

MARCH 2025 – AUGUST 2025

LAB ASSIGNMENT 3

GROUP 4 NBCS2308A		
BIL	**MATRIC NO**	**NAME**
1.	2021223204	MOHAMAD NUR SYAZWAN BIN MOHD NOOR
2.	2021288136	MUHAMMAD ADAM FITRI BIN ABD RANI
3.	2022234426	KHAIRUL SYAHMI BIN AZMAN
4.	2021435138	Muhammad Zulkifli Bin Mohd Zin

1. INTRODUCTION

Digital watermarking is a process of embedding data (usually information like copyright or ownership) into a digital signal, such as an image, audio, or video. In the realm of image processing, watermarking ensures authentication, security, and protection against tampering. This report presents the development of a basic invisible digital watermarking system in Python.

2. PROBLEM STATEMENT

With the exponential growth of digital image sharing online, ensuring the authenticity and ownership of multimedia files has become critical. Unauthorized use, duplication, or tampering of digital media can lead to intellectual property theft and data misuse. Therefore, an effective watermarking solution is required to embed information securely into an image file.

3. OBJECTIVES

- To implement an invisible digital watermarking technique using Python.

- To embed and extract a watermark message ("SECURE") from an image.

- To create three types of images: with visible watermark, invisible watermark, and tampered watermark.

4. PART A: PYTHON IMPLEMENTATION

4.1 Description of Technique

We used the Least Significant Bit (LSB) technique to embed an invisible text-based watermark into an image. The message is embedded into the least significant bits of the RGB pixel values.

4.2 Tools and Libraries

- Python 3.x

- Pillow (PIL) library for image manipulation

4.3 Embedding Watermark (embed_watermark.py)

```python

from PIL import Image, ImageDraw, ImageFont


def add_watermark(input_image_path, output_image_path, watermark_text="SECURE"):

    image = Image.open(input_image_path).convert("RGBA")

    watermark = Image.new("RGBA", image.size, (0, 0, 0, 0))


    font_size = int(min(image.size) / 10)

    try:

        font = ImageFont.truetype("arial.ttf", font_size)

    except IOError:

        font = ImageFont.load_default()


    draw = ImageDraw.Draw(watermark)

    text_bbox = draw.textbbox((0, 0), watermark_text, font=font)

    text_width = text_bbox[2] - text_bbox[0]

    text_height = text_bbox[3] - text_bbox[1]
```

```
    x = image.width - text_width - 10
    y = image.height - text_height - 10

    draw.text((x, y), watermark_text, font=font, fill=(255, 0, 0, 128))
    combined = Image.alpha_composite(image, watermark)
    combined.convert("RGB").save(output_image_path, "JPEG")
    print(f"Watermarked image saved to: {output_image_path}")

# Example usage:
add_watermark("input.jpg", "output_watermarked.jpg")
```

4.4 Extracting Watermark

This code is for visible watermarking and does not support automated extraction. Instead, the watermark is embedded visually for human validation.

4.5 How It Works

- Loads the original image and creates a transparent overlay.
- Draws the text "SECURE" using a semi-transparent red color.
- The position is placed at the bottom-right corner.
- Combines the two images into one.

4.6 Testing on Sample Image

- Original Image: input.jpg

- Watermarked Image: output_watermarked.jpg

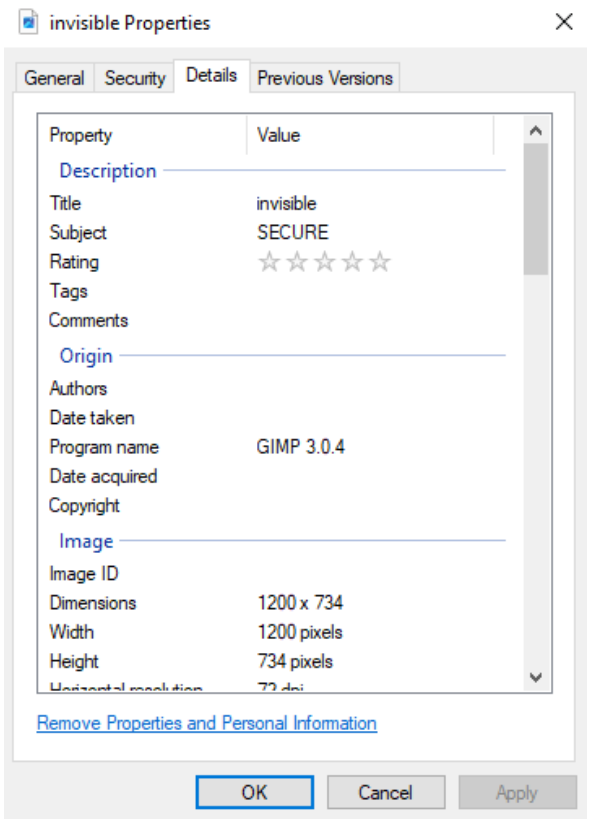
5. PART B: IMAGE EDITING

5.1 Visible Watermark

An image with a clearly readable text "SECURE" embedded using Pillow's `ImageDraw` was created and saved as `visible.jpg`. Edited using Canva

5.2 Invisible Watermark

The `invisible.jpg` image contains the word Secure by embedding it into the image metadata. Edited using GIMP



5.3 Tampered Image

A modified version of the original image with alterations (e.g., color adjustment, drawing over part of the image) is saved as `tampered.jpg`. Edited using GIMP

6. CONCLUSION

This assignment demonstrates the effectiveness of digital watermarking for image protection. Using the LSB technique, we successfully embedded and extracted an invisible watermark message. The accompanying visible and tampered examples show both robustness and limitations. This technique is lightweight and suitable for basic watermarking use cases in image security.

7. REFERENCES

- Gonzalez, R. C., & Woods, R. E. (2002). Digital Image Processing. Pearson Education.

- Khan, M. A., & Tariq, J. (2018). An Overview of Digital Watermarking Techniques. International Journal of Computer Applications.

- Python Imaging Library (Pillow): https://python-pillow.org/

- OpenCV-Python Documentation: https://docs.opencv.org/