Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа 1

по дисциплине «Прикладная стеганография»

Выполнил: ст. гр. ЗМП-41 Лёвкин И. А.

Проверила: Мерзлякова Е.Ю.

Новосибирск 2025

**Обзор методов класса LSB**

Методы LSB (Least Significant Bit) относятся к стеганографическим техникам, которые скрывают информацию в наименее значимых битах пикселей изображения. Основные методы класса LSB включают:

1. LSB Substitution (Замена LSB):
   * Заменяет наименее значимый бит пикселя на бит сообщения.
   * Прост в реализации, но уязвим к статистическому анализу, так как создает асимметричные искажения в гистограмме изображения.
   * Обнаруживается методами RS-анализа, SP-анализа и другими статистическими тестами.
2. LSB Matching (Сопоставление LSB):
   * Если бит сообщения не совпадает с LSB пикселя, значение пикселя случайно увеличивается или уменьшается на 1.
   * Более устойчив к статистическим атакам, так как сохраняет симметрию гистограммы.
   * Сложнее обнаружить, но требует более аккуратной реализации для уменьшения количества артефактов.
3. Модификации LSB Matching:
   * Включают адаптивные методы, такие как Edge Adaptive LSB Matching, которые скрывают данные в областях с высокой текстурой для повышения устойчивости.

Для реализации был выбран метод LSB Matching по следующим причинам**:**

* LSB Matching сложнее обнаружить с помощью стандартных методов стегоанализа, таких как RS или SP, благодаря случайному выбору между +1 и -1.
* Метод остается относительно простым в реализации, несмотря на повышенную устойчивость.
* Позволяет скрывать данные без значительных искажений, что важно для сохранения визуального качества изображения.

Алгоритм шифрования:

1. Загрузка изображения-контейнера.
2. Преобразование сообщения в бинарный формат.
3. Для каждого бита сообщения выбирается соответствующий пиксель. Если LSB пикселя не совпадает с битом сообщения, случайно выбирается +1 или -1 (с учетом границ 0 и 255).

Алгоритм дешифрования:

1. Загрузка изображения-контейнера.
2. Чтение LSB пикселей для восстановления сообщения.
3. Преобразование бинарного формата в сообщение.

**Оценка алгоритма**

Проведём встраивание данных в изображение и оценим ёмкость и величину искажения PSNR на 8-битном изображении с палитрой из оттенков серого. Встраиваемый текст на английском языке и составляет размер 16 КБ. Размер контейнера в свою очередь составляет 257 КБ.

Вывод программы:

|  |
| --- |
| Embedding analysis: - Capacity: 32768 bytes - Message size: 16373 bytes (50.0%) - PSNR: 53.96 dB |

Здесь можно видеть, что ёмкость встраивания составляет 32 КБ т.к. мы меняем только наименее значимый бит пикселя то ёмкость = размер контейнера / 8. Изображение 8-битное и каждый пиксель величиной в 1 байт т.к. это индекс цвета в палитре.

Мы встроили текст размером в 50% от максимальной вместимости контейнера. В таком случае PSNR составляет 53.96 ДБ, при таком значении всё ещё трудно обнаружить следы кодирования сообщения при визуальном анализе.

|  |  |
| --- | --- |
| Рис. 1. Контейнер без сообщения | Рис. 2. Контейнер после встраивания сообщения |

Проведём визуальную атаку чтобы выделить артефакты появившиеся при встраивании сообщения

|  |  |
| --- | --- |
| Рис. 3. Визуальная атака на контейнер без сообщения | Рис. 4. Визуальная атака на контейнер с сообщением |

**Листинг**

import argparse

import os

from pathlib import Path

import sys

from PIL import Image

import numpy as np

ROOT\_DIR = Path(\_\_file\_\_).resolve().parent.parent

if str(ROOT\_DIR) not in sys.path:

    sys.path.append(str(ROOT\_DIR))

import lsb

import utils.stego as stego

def main():

    parser = argparse.ArgumentParser(description="LSB Stenography for 8-bit BMP images")

    subparsers = parser.add\_subparsers(dest="command", required=True)

    enc = subparsers.add\_parser("encode", help="Encode message into image")

    enc.add\_argument("-m", "--message", required=True, help="Message file")

    enc.add\_argument("-i", "--input", required=True, help="Input BMP image")

    enc.add\_argument("-o", "--output", required=True, help="Output stego image")

    dec = subparsers.add\_parser("decode", help="Decode message from image")

    dec.add\_argument("-i", "--input", required=True, help="Input BMP image")

    dec.add\_argument("-o", "--output", required=True, help="Output message file")

    args = parser.parse\_args()

    if args.command == "encode":

        encode\_cmd(args)

    elif args.command == "decode":

        decode\_cmd(args)

def encode\_cmd(args):

    image = load\_image(args.input)

    message = open(args.message, "rb").read()

    image\_array = np.array(image)

    capacity = len(image\_array.flatten()) // lsb.BITS\_PER\_BYTE

    if len(message) > capacity:

        raise ValueError(

            f"Message too large. Capacity: {capacity} bytes, message: {len(message)} bytes"

        )

    stego\_array = lsb.embed\_lsb\_matching(image\_array, image.mode, message)

    stego\_img = Image.fromarray(stego\_array)

    print("Embedding analysis:")

    print(f"- Capacity: {capacity} bytes")

    print(

        f"- Message size: {len(message)} bytes ({len(message) / capacity \* 100:.1f}%)"

    )

    print(f"- PSNR: {stego.psnr(image\_array, stego\_array, image.mode):.2f} dB")

    attack\_path = os.path.splitext(args.output)[0] + "\_difference.bmp"

    attack\_img = stego.generate\_lsb\_attack\_image(stego\_img)

    attack\_img.save(attack\_path)

    print(f"Visual attack image saved to {attack\_path}")

    stego\_img.save(args.output)

def decode\_cmd(args):

    stego\_img = load\_image(args.input)

    stego\_array = np.array(stego\_img)

    message = lsb.extract\_lsb\_matching(stego\_array, stego\_img.mode)

    with open(args.output, "wb") as f:

        f.write(message)

def load\_image(path):

    img = Image.open(path)

    if img.mode not in ("P", "L", "RGB"):

        raise ValueError(

            "Only 24-bit, 8-bit indexed or grayscale BMP images are supported."

        )

    return img

if \_\_name\_\_ == "\_\_main\_\_":

    main()

from PIL import Image

import numpy as np

BITS\_PER\_BYTE = 8

BYTE\_ORDER = "big"

MESSAGE\_LENGTH\_BYTES = 4

def embed\_lsb\_matching(

    image\_array: np.ndarray, mode: str, message\_bytes: bytes

) -> Image.Image:

    match mode:

        case "P" | "L":

            return embed\_lsb\_matching\_8bit(image\_array, message\_bytes)

        case "RGB":

            return embed\_lsb\_matching\_24bit(image\_array, message\_bytes)

        case \_:

            raise ValueError(

                "Only 24-bit, 8-bit indexed or grayscale BMP images are supported."

            )

def extract\_lsb\_matching(stego\_array: np.ndarray, mode: str) -> Image.Image:

    match mode:

        case "P" | "L":

            return extract\_lsb\_matching\_8bit(stego\_array)

        case "RGB":

            return extract\_lsb\_matching\_24bit(stego\_array)

        case \_:

            raise ValueError(

                "Only 24-bit, 8-bit indexed or grayscale BMP images are supported."

            )

def prepare\_for\_lsb(image\_array: np.ndarray, message\_bytes: bytes):

    if not isinstance(message\_bytes, bytes):

        raise ValueError("Message must be bytes")

    # Add the message length (4 bytes) to the beginning

    message\_length = len(message\_bytes)

    length\_bytes = message\_length.to\_bytes(MESSAGE\_LENGTH\_BYTES, byteorder=BYTE\_ORDER)

    full\_message = length\_bytes + message\_bytes

    message\_bits = "".join([format(byte, "08b") for byte in full\_message])

    # Check capacity

    total\_pixels = image\_array.size

    required\_pixels = len(message\_bits)

    if required\_pixels > total\_pixels:

        raise ValueError("The message is too big to fit in the image")

    return message\_bits

def embed\_lsb\_matching\_8bit(

    image\_array: np.ndarray, message\_bytes: bytes

) -> np.ndarray:

    message\_bits = prepare\_for\_lsb(image\_array, message\_bytes)

    image\_array = image\_array.copy()

    flat\_array = image\_array.flatten()

    for i in range(len(message\_bits)):

        message\_bit = int(message\_bits[i])

        if (flat\_array[i] & 1) != message\_bit:

            # Randomly change LSB if bits are not equal

            if flat\_array[i] == 255:

                flat\_array[i] -= 1

            elif flat\_array[i] == 0:

                flat\_array[i] += 1

            else:

                flat\_array[i] += np.random.choice([-1, 1])

    # Reshape embedded image in 2d

    embedded\_array = flat\_array.reshape(image\_array.shape)

    return embedded\_array

def embed\_lsb\_matching\_24bit(

    image\_array: np.ndarray, message\_bytes: bytes

) -> np.ndarray:

    message\_bits = prepare\_for\_lsb(image\_array, message\_bytes)

    image\_array = image\_array.copy()

    bit\_index = 0

    for row in image\_array:

        for pixel in row:

            for channel in range(3):  # B, G, R

                if bit\_index >= len(message\_bits):

                    break

                bit = int(message\_bits[bit\_index])

                if (pixel[channel] & 1) != bit:

                    # Randomly change LSB if bits are not equal

                    if pixel[channel] == 255:

                        pixel[channel] -= 1

                    elif pixel[channel] == 0:

                        pixel[channel] += 1

                    else:

                        pixel[channel] += np.random.choice([-1, 1])

                bit\_index += 1

    return image\_array

def extract\_lsb\_matching\_8bit(image\_array: np.ndarray) -> bytes:

    image\_array = image\_array.copy()

    flat\_img = image\_array.flatten()

    length\_bits\_count = MESSAGE\_LENGTH\_BYTES \* BITS\_PER\_BYTE

    # Extract message length (first 32 bits)

    length\_bits = [str(pixel & 1) for pixel in flat\_img[:length\_bits\_count]]

    length = int("".join(length\_bits), 2)

    # Extract message

    message\_bits = []

    for i in range(length\_bits\_count, length\_bits\_count + length \* BITS\_PER\_BYTE):

        message\_bits.append(str(flat\_img[i] & 1))

    # Convert message bits to bytes

    return bits\_str\_to\_bytes(message\_bits, length)

def extract\_lsb\_matching\_24bit(image\_array: np.ndarray):

    image\_array = image\_array.copy()

    length\_bits\_count = MESSAGE\_LENGTH\_BYTES \* BITS\_PER\_BYTE

    # Extract message length (first 32 bits)

    length\_bits = []

    bit\_count = 0

    for row in image\_array:

        for pixel in row:

            for channel in range(3):

                if bit\_count >= length\_bits\_count:

                    break

                length\_bits.append(str(pixel[channel] & 1))

                bit\_count += 1

    length = int("".join(length\_bits), 2)

    # Extract message

    message\_bits = []

    total\_bits = length\_bits\_count + length \* BITS\_PER\_BYTE

    bit\_count = 0

    for row in image\_array:

        for pixel in row:

            for channel in range(3):

                if bit\_count >= total\_bits:

                    break

                if bit\_count >= length\_bits\_count:  # Skip length

                    message\_bits.append(str(pixel[channel] & 1))

                bit\_count += 1

    # Convert message bits to bytes

    return bits\_str\_to\_bytes(message\_bits, length)

def bits\_str\_to\_bytes(bits: str, length: int) -> bytes:

    return bytes(

        [

            int("".join(bits[i \* BITS\_PER\_BYTE : (i + 1) \* BITS\_PER\_BYTE]), 2)

            for i in range(length)

        ]

    )