Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №8
на тему

СТЕГАНОГРАФИЧЕСКИЕ МЕТОДЫ

Выполнил: студент гр. 253503
Минич С.В.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В качестве объекта для практического изучения стеганографических методов защиты информации был выбран метод сокрытия текстовых сообщений непосредственно в частотной области JPEG-изображений путём модификации наименьших значащих битов (*LSB*) коэффициентов дискретного косинусного преобразования (*DCT*). Данный подход сочетает в себе преимущества работы в сжатой области изображения и простоту реализации классического *LSB*-метода, обеспечивая при этом высокую степень визуальной и статистической скрытности.

В отличие от традиционного встраивания в пространственную область декодированного растрового изображения, предлагаемый метод оперирует непосредственно с *DCT*-коэффициентами яркостной компоненты (*Y*). Встраивание осуществляется путём замены младшего бита выбранного ненулевого коэффициента на бит скрываемого сообщения. Такой выбор позиции позволяет минимизировать визуальные искажения, поскольку изменение только младшего бита практически не сказывается на качестве восстановленного изображения.

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

В ходе лабораторной работы была реализована стеганографическая система сокрытия и извлечения текстовых сообщений в *JPEG*-изображениях путём прямой модификации младших битов коэффициентов дискретного косинусного преобразования. Реализация выполнена на языке *Python* с использованием библиотеки *jpeglib*, которая обеспечивает доступ к *DCT*-коэффициентам.

Для встраивания выбран фиксированный коэффициент с индексами (7,7) в каждом 8×8-блоке яркостной компоненты *Y*. Изменение затрагивает исключительно младший бит этого коэффициента, что достигается установкой вместо него очередного бита скрываемого сообщения. Такой подход практически не влияет на визуальное качество изображения.

Процесс сокрытия начинается с преобразования текстовой строки в байты и определения длины полученного массива. Далее проводится проверка достаточности количества доступных блоков для размещения 32-битного заголовка длины и собственно данных сообщения. Первые 32 блока последовательно используются для записи длины сообщения, после чего в последующие блоки побитно встраивается само сообщение, начиная со старшего бита каждого байта.

Процедура извлечения выполняется в обратном порядке: сначала считываются младшие биты коэффициентов (7,7) первых 32 блоков для восстановления длины сообщения, затем извлекается ровно указанное количество битов данных, которые собираются в байтовый массив и декодируются в исходную строку *UTF-8*.

В практической части работа программы проверена на реальном *JPEG*-изображении. В качестве контейнера использовано изображение «*cat.jpg*», в которое было успешно встроено тестовое сообщение «*Hello, world*!». Полученное стегоизображение «*hidden_cat.jpg*» визуально полностью идентично исходному. Применение функции извлечения позволило безошибочно восстановить скрытый текст, что подтвердило полное совпадение исходного и извлечённого сообщений.

Вывод результатов выполнения программы представлен на рисунке 1.

```
PS C:\sem7\MZI> & C:/Users/imsve/AppData/Local/Programs/Python/Python312/python.exe c:/sem7/MZI/lab8/l8.py
Исходное сообщение: Hello, world!
Извлеченное сообщение: Hello, world!
Совпадение: True
```

Рисунок 1 – Результат выполнения алгоритма

Таким образом, в рамках лабораторной работы успешно реализован и протестирован метод стеганографического сокрытия данных непосредственно в частотной области *JPEG*-изображений.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был подробно изучен и практически реализован стеганографический метод сокрытия текстовых сообщений непосредственно в частотной области *JPEG*-изображений путём замены младших битов выбранных *DCT*-коэффициентов. С использованием библиотеки *jpeglib* разработана полнофункциональная программа на языке *Python*, обеспечивающая встраивание и извлечение скрытой информации без необходимости доступа к исходному контейнеру.

Проведённое тестирование на реальных *JPEG*-изображениях показало полную корректность работы всех этапов алгоритма: тестовое сообщение успешно встраивалось и извлекалось без единой ошибки, при этом полученное стегоизображение оставалось визуально неотличимым от оригинала, а изменение качества оставалось в пределах, незаметных для человеческого глаза.

Цель лабораторной работы достигнута: выполнена программная реализация и практическая проверка одного из эффективных методов стеганографии в сжатой области *JPEG*. Полученные результаты позволили глубоко освоить внутреннюю структуру формата *JPEG*, особенности работы с *DCT*-коэффициентами, преимущества и ограничения *LSB*-встраивания в частотную область, а также закрепить понимание современных подходов к незаметной передаче конфиденциальной информации в мультимедийных данных.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Стеганография в JPEG: методы, история развития и защита | Вектoриум | Дзен. [Электронный ресурс]. – Режим доступа: https://dzen.ru/a/aIOCYR7b2XJb76CY?ysclid=mirc1week8185418527 – Дата доступа: 04.12.2025

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
import jpeglib

BITS_FOR_LENGTH = 32
COEF_POS = (7, 7)

def string_to_bytes(s: str):
    return s.encode('utf-8')

def bytes_to_string(b: bytes):
    return b.decode('utf-8')

def hide_string_in_jpeg(string: str, source_image_path: str, hidden_image_path: str):
    with jpeglib.read_dct(source_image_path) as dct:
        y_coefficients = dct.Y

        height_blocks, width_blocks, _, _ = y_coefficients.shape
        total_blocks = height_blocks * width_blocks

        bytes_string = string_to_bytes(string)
        string_length = len(bytes_string)
        bits_needed = BITS_FOR_LENGTH + string_length * 8

        if bits_needed > total_blocks:
            raise Exception(f"Изображение слишком маленькое. Нужно {bits_needed} блоков, доступно
{total_blocks}")

        for bit_index in range(BITS_FOR_LENGTH):
            block_y = bit_index // width_blocks
            block_x = bit_index % width_blocks

            if block_y >= height_blocks or block_x >= width_blocks:
                break

            bit = (string_length >> (BITS_FOR_LENGTH - 1 - bit_index)) & 1
            coef_value = y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]]
            new_value = (coef_value & 0b11111110) | bit
            y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]] = new_value

        for bit_idx in range(string_length * 8):
            data_bit_index = BITS_FOR_LENGTH + bit_idx
            block_y = data_bit_index // width_blocks
            block_x = data_bit_index % width_blocks

            if block_y >= height_blocks or block_x >= width_blocks:
                break

            byte_idx = bit_idx // 8
            bit_pos = bit_idx % 8
            bit = (bytes_string[byte_idx] >> (7 - bit_pos)) & 1

            coef_value = y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]]
            new_value = (coef_value & 0b11111110) | bit
            y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]] = new_value

        dct.Y = y_coefficients
        dct.write_dct(hidden_image_path)

def extract_string_from_jpeg(hidden_image_path: str):
    with jpeglib.read_dct(hidden_image_path) as dct:
        y_coefficients = dct.Y
        height_blocks, width_blocks, _, _ = y_coefficients.shape
        total_blocks = height_blocks * width_blocks

        string_length = 0
        for bit_index in range(BITS_FOR_LENGTH):
            block_y = bit_index // width_blocks
            block_x = bit_index % width_blocks

            if block_y >= height_blocks or block_x >= width_blocks:
                break

            coef_value = y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]]
            bit = coef_value & 1
            string_length = (string_length << 1) | bit

        if string_length <= 0:
            raise Exception(f"Неверная длина данных: {string_length}.")

        bytes_string = bytearray()
        for bit_idx in range(string_length * 8):
            data_bit_index = BITS_FOR_LENGTH + bit_idx

            block_y = data_bit_index // width_blocks
            block_x = data_bit_index % width_blocks

            if block_y >= height_blocks or block_x >= width_blocks:
                break
```

```
            coef_value = y_coefficients[block_y, block_x, COEF_POS[0], COEF_POS[1]]
            bit = coef_value & 1

            byte_idx = bit_idx // 8
            bit_pos = bit_idx % 8

            if bit_pos == 0:
                bytes_string.append(0)

            bytes_string[byte_idx] = (bytes_string[byte_idx] << 1) | bit

        return bytes_to_string(bytes(bytes_string))

if __name__ == "__main__":
    message = "Hello, world!"

    hide_string_in_jpeg(message, "lab8/cat.jpg", "lab8/hidden_cat.jpg")
    extracted = extract_string_from_jpeg("lab8/hidden_cat.jpg")

    print(f"Исходное сообщение: {message}")
    print(f"Извлеченное сообщение: {extracted}")
    print(f"Совпадение: {message == extracted}")
```