МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

на тему: *«Применение стеганографических методов для сокрытия информации»*

Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»
Дисциплина «Защита информации»

Выполнил:

студентка группы 201-361

Саблина Анна Викторовна

Проверил:

Харченко Елена Алексеевна

Москва 2023

Теоретическая часть

BMP (Bitmap) является одним из форматов файлов изображений. Структура BMP-файла состоит из заголовка файла (File Header), заголовка изображения (Image Header) и массива пикселей (Pixel Array). Рассмотрим каждую часть подробнее:

1. Заголовок файла (File Header): ***// 14 байтов***
    - Поле «Signature» (2 байта): обычно содержит символы "BM", указывающие на формат BMP.
    - Поле «File Size» (4 байта): размер файла в байтах.
    - Поле «Reserved» (4 байта): резервные значения, обычно заполняются нулями.
    - Поле «Data Offset» (4 байта): смещение, указывающее на начало массива пикселей.
2. Заголовок изображения (Image Header): ***// 40 байтов***
    - Поле «Header Size» (4 байта): размер заголовка изображения в байтах.
    - Поле «Image Width» (4 байта): ширина изображения в пикселях.
    - Поле «Image Height» (4 байта): высота изображения в пикселях.
    - Поле «Color Planes» (2 байта): количество цветовых плоскостей (обычно 1).
    - Поле «Bits per Pixel» (2 байта): глубина цвета, то есть количество бит на пиксель (обычно 24).
    - Поле «Compression» (4 байта): тип сжатия (обычно без сжатия).
    - Поле «Image Size» (4 байта): размер изображения в байтах.

- Поле «Horizontal Resolution» (4 байта): горизонтальное разрешение изображения в пикселях на метр.
- Поле «Vertical Resolution» (4 байта): вертикальное разрешение изображения в пикселях на метр.
- Поле «Colors Used» (4 байта): количество используемых цветов.
- Поле «Important Colors» (4 байта): количество "важных" цветов. Обычно все цвета важны, поэтому это значение равно нулю.

3. Массив пикселей (Pixel Array):
   - Массив пикселей содержит фактические цветовые данные изображения. Каждый пиксель обычно представлен последовательностью байтов, определяющих значения компонент цвета (например, красного, зеленого и синего). Формат представления цвета зависит от глубины цвета, указанной в заголовке изображения.
   - Общее количество байтов в массиве пикселей можно вычислить следующим образом:

$$Total\ Bytes = Image\ Width \times Image\ Height \times \frac{Bits\ per\ Pixel}{8}$$

Структура «Image Data» для 24-битного BMP-файла может выглядеть следующим образом:

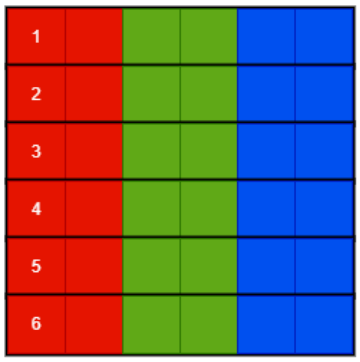

Рисунок 1 – Цветовые байты пикселей изображения

LSB (Least Significant Bit) Replacement (замена наименее значимого бита) – это метод стеганографии, который используется для встраивания дополнительной информации в низкоразрядные биты данных, например, в изображениях или звуковых файлах.

Основная идея состоит в том, что наименее значимый бит в байте (обычно самый правый бит) обычно имеет наименьшее влияние на восприятие исходных данных. Путем замены наименее значимого бита каждого пикселя или сэмпла звука на биты добавляемой информации можно незаметно скрыть эти данные.

Процесс LSB Replacement обычно состоит из следующих шагов:

1. Разделение исходных данных: встраиваемая информация разделяется на отдельные биты.
2. Выбор контейнера: выбирается контейнерный файл (например, изображение BMP или звуковой файл WAV), в котором будут скрыты данные.
3. Обход контейнера: пиксели (для изображений) или сэмплы (для аудио) контейнера перебираются в определенном порядке.
4. Замена LSB: наименее значимый бит каждого пикселя или сэмпла заменяется битом встраиваемой информации.
5. Завершение и сохранение: обработанный контейнер сохраняется в виде нового файла, который содержит в себе исходные данные и скрытую информацию.

Основным преимуществом метода LSB Replacement является его относительная простота реализации и невидимость изменений для человеческого восприятия, особенно при встраивании небольшого количества данных. Однако он также имеет некоторые ограничения, такие как уязвимость к атакам на обнаружение стеганографии и потерю данных при сжатии или обработке контейнерного файла.

Практическая часть

Для реализации программы, способной внедрить в файл 28.bmp и затем извлечь из него хешкод файла leasing.txt, полученного с помощью алгоритма SHA-1, был наглядно рассмотрен описанный в Теоретической части метод.

На рисунках 2 и 3 отображен общий принцип работы разрабатываемой программы: пунктиром обозначено начало набора байтов, в который будет происходить внедрение байта шифруемого сообщения, а точками – байт, содержащий наименее значимый бит пикселя, который и будет изменен.

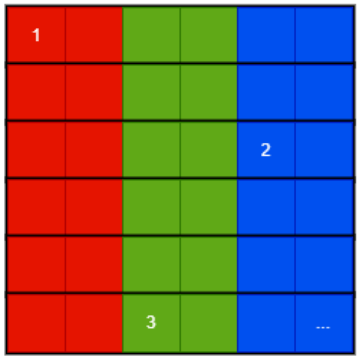

Рисунок 2 – Разбиение массива байтов пикселей на наборы по 16 байт для дальнейшего последовательного внедрения байта шифруемого сообщения в каждый наименее значимый бит 8 пикселей

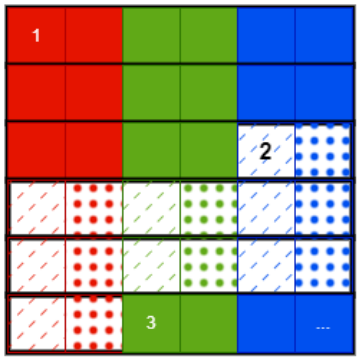

Рисунок 3 – Принцип внедрения наглядно

Был создан проект на языке Java.

Общий принцип работы программы:

1. Генерируется ключ шифрования и хэш SHA-1 для файла `leasing.txt`.
2. Из строки хэш-кода создается последовательность байтов.
3. Хэш-код в качестве шифруемого сообщения внедряется в структуру BMP-файла изображения и сохраняется в новом файле `.bmp`.
4. Ключ шифрования сохраняется в новый файл `.txt`.
5. Хэш-код извлекается из созданного ранее BMP-файла изображения.

Примечание: длина ключа шифрования установлена как 20 изначально, поскольку это число является длиной хэша SHA-1 для файла `leasing.txt`.

```
static int keyLength = 20; // Длина ключа шифрования
static int imageOffset = 54; // Размер хедера BMP-изображения составляет 54 байта

public static void main(String[] args) {
    int[] key = generateKey();
    saveKey(key, "key.txt");

    String hexString = generateSHA1();
    // Создает из строки хэш-кода файла leasing.txt последовательность байтов
    assert hexString != null;
    ByteBuffer buffer = ByteBuffer.allocate(hexString.length() / 2);
    for (int i = 0; i < hexString.length(); i += 2) {
        buffer.put((byte) Integer.parseInt(hexString.substring(i, i + 2), 16));
    }
    byte[] bytes = buffer.array();


    encode(bytes, key);
    key = loadKey("key.txt");
    decode(key);
}
```

Листинг 1 – Декларация глобальных переменных и метод main

Метод `encode(byte[] message, int[] key)` внедряет в BMP-изображение хэшкод файла `leasing.txt` с помощью реализации LSB Replacement.

```
public static void encode(byte[] message, int[] key) {
    File bmpFile = new File("input.bmp");
    byte[] imageBytes = new byte[(int) bmpFile.length()];

    try {
        FileInputStream fis = new FileInputStream(bmpFile);
        fis.read(imageBytes);
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    byte[] messageBytes = new byte[keyLength];

    for (int i = 0; i < message.length && i < keyLength; i++) {
```

```
            messageBytes[i] = message[i];
        }

        // Кодирует сообщение в массив байтов изображения, используя LSB стеганографию
        int messageOffset = 0;
        while (messageOffset < messageBytes.length) {
            byte messageByte = messageBytes[messageOffset];
            encodeByte(key[messageOffset], imageBytes, messageByte);
            messageOffset++;
        }

        // Сохраняет модифицированное BMP-изображение в новый файл
        try {
            FileOutputStream fos = new FileOutputStream("image_with_secret.bmp");
            fos.write(imageBytes);
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }

        System.out.println("Secret message successfully hidden in BMP image!");
    }
```

Листинг 2 – Метод encode(byte[] message, int[] key)

Метод `decode(int[] key)` извлекает из BMP-изображения хэшкод файла `leasing.txt` с помощью реализации LSB Replacement.

```
public static void decode(int[] key) {
    File bmpFile = new File("image_with_secret.bmp");
    byte[] imageBytes = new byte[(int) bmpFile.length()];

    try {
        FileInputStream fis = new FileInputStream(bmpFile);
        fis.read(imageBytes);
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

    // Декодирует ключ из изображения, используя LSB стеганографию
    int messageOffset = 0;
    byte[] messageBytes = new byte[keyLength];

    while (messageOffset < messageBytes.length) {
        byte messageByte = decodeByte(key[messageOffset], imageBytes);
        messageBytes[messageOffset] = messageByte;
        messageOffset++;
    }

    // Переводит из последовательности байтов в строку
    ByteBuffer buffer = ByteBuffer.wrap(messageBytes);
    StringBuilder hexString = new StringBuilder();
    while (buffer.hasRemaining()) {
        hexString.append(String.format("%02x", buffer.get() & 0xff));
    }

    System.out.println("Secret message extracted from BMP image: " + hexString);
}
```

Листинг 3 – Метод decode(int[] key)

Метод `contains(int[] arr, int val)` проверяет наличие передаваемого числа в указанном массиве.

```
public static boolean contains(int[] arr, int val) {
    for (int j : arr) {
        if (j == val) {
            return true;
        }
    }
    return false;
}
```

Листинг 4 – Метод contains(int[] arr, int val)

Метод `generateKey()` генерирует рандомные числа на отрезке `[1; max]` в количестве `keyLength` для формирования ключа шифрования.

Примечание: максимальный размер элемента генерируемого ключа шифрования взят как 400, однако можно в дальнейшем усовершенствовать реализацию программы, рассчитав общее количество байтов в массиве пикселей и брать за основу генерации это число.

```
public static int[] generateKey() {
    int max = 400;
    int size = keyLength;
    int[] key = new int[size];
    Random rand = new Random();

    // Генерирует последовательность уникальных случайных чисел
    for (int i = 0; i < size; i++) {
        int randInt = rand.nextInt(max) + 1;
        while (contains(key, randInt)) {
            randInt = rand.nextInt(max) + 1;
        }
        key[i] = randInt;
    }
    return key;
}
```

Листинг 5 – Метод generateKey()

Метод `saveKey(int[] arr, String filename)` сохраняет полученный ключ шифрования в файл.

```
public static void saveKey(int[] arr, String filename) {
    try (PrintWriter writer = new PrintWriter(filename)) {
        for (int i = 0; i < keyLength; i++) {
            writer.println(arr[i]);
        }
        System.out.println("Array saved to " + filename);
    } catch (FileNotFoundException e) {
        System.out.println("Error: " + e.getMessage());
    }
}
```

Листинг 6 – Метод saveKey(int[] arr, String filename)

Метод `loadKey(String filename)` извлекает из файла записанный ключ шифрования.

```
public static int[] loadKey(String filename) {
    try (Scanner scanner = new Scanner(new File(filename))) {
        int[] loadedArr = new int[keyLength];
        int i = 0;
        while (scanner.hasNextInt()) {
            loadedArr[i++] = scanner.nextInt();
        }
        System.out.println("Array loaded from " + filename);
        return loadedArr;
    } catch (FileNotFoundException e) {
        System.out.println("Error: " + e.getMessage());
        return null;
    }
}
```

Листинг 7 – Метод loadKey(String filename)

Метод `encodeByte(int offset, byte[] imageBytes, byte messageByte)` внедряет в байтовое представление BMP-изображения байт шифруемого сообщения, меняя наименее значимый бит каждого пикселя.

```
public static void encodeByte(int offset, byte[] imageBytes, byte messageByte) {
    for (int i = 0; i < 16; i++) {
        int bit = (messageByte >>> i) & 1;
        int imageByteIndex = imageOffset + (offset * 16) + i;
        imageBytes[imageByteIndex] = (byte) ((imageBytes[imageByteIndex] & 0xFFFE) | bit);
    }
}
```

Листинг 8 – Метод encodeByte(int offset, byte[] imageBytes, byte messageByte)

Метод `decodeByte(int offset, byte[] imageBytes)` извлекает из байтового представления BMP-изображения байт зашифрованного сообщения.

```
public static byte decodeByte(int offset, byte[] imageBytes) {
    byte messageByte = 0;
    for (int i = 0; i < 16; i++) {
        int imageByteIndex = imageOffset + (offset * 16) + i;
        int bit = imageBytes[imageByteIndex] & 1;
        messageByte |= bit << i;
    }
    return messageByte;
}
```

Листинг 9 – Метод decodeByte(int offset, byte[] imageBytes)

Метод генерации хэш-кода SHA1 для файла `leasing.txt` взят из лабораторной работы №1 и адаптирован под работу с ним.

Ссылка на проект в репозитории GitHub:

- https://github.com/LazyShAman/dp/tree/main/2.