

## СОДЕРЖАНИЕ

Обозначения и сокращения . . . . .	6
Введение . . . . .	7
1 Общие положения и описание стеганографических методов защиты информации . . . . .	8
1.1 Основные понятия . . . . .	8
1.2 Цифровые водяные знаки и цифровые отпечатки . . . . .	9
1.3 Обобщенные стеганографические методы . . . . .	11
1.4 Стегоанализ . . . . .	11
2 Скрытия методом наименее значимого бита . . . . .	13
2.1 Общие сведения . . . . .	13
2.2 Алгоритм . . . . .	14
2.3 Формат файла PNG . . . . .	16
2.4 Реализация LSB для контейнера PNG . . . . .	19
3 Скрытие в спектральной области . . . . .	23
3.1 Дискретное косинусное преобразование . . . . .	23
3.2 JPEG . . . . .	24
3.3 JSteg . . . . .	30
3.4 Метод относительной замены величин коэффициентов ДКП . . . . .	34
4 Стегоанализ . . . . .	37
4.1 Методы стегоанализа . . . . .	37
4.2 Субъективная атака LSB . . . . .	37
4.3 Атака оценки числа переходов значений младших бит в соседних элементах контейнера . . . . .	38
4.4 Атака хи-квадрат . . . . .	40
4.5 Методы противодействия . . . . .	42
5 Экономическая оценка проекта . . . . .	44
5.1 Постановка задачи . . . . .	44
5.2 Оценка стоимости объектов интеллектуальной собственности . . . . .	44
5.3 Оценка стоимости разработки . . . . .	48
5.4 Оценка стоимости использования оборудования и сопровождения системы . . . . .	48
5.5 Экономическое обоснование проекта . . . . .	49

Заключение .....	50
Список использованных источников .....	51
Приложение А Реализация метода Бенгама-Мемона-Эо-Юнга на Python .....	52

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

CRC — cyclic redundancy code (циклический избыточный код)

LSB — least significant bit (наименее значащий бит)

DRM — digital rights management (цифровое управление правами)

ЦВЗ — цифровой водяной знак

RGB — red, green, blue (красный, синий, зеленый)

ДКП — дискретное косинусное преобразование

DCT — discrete cosine transform (дискретное косинусное преобразование)

НИР — научно-исследовательская работа

ОИС — объект интеллектуальной собственности

ГПСЧ — генератор псевдослучайных чисел

## ВВЕДЕНИЕ

Стеганография — это практика сокрытия сообщения внутри другого сообщения или физического объекта. Тогда как криптография скрывает содержимое сообщения, стеганография скрывает сам факт существования какого-либо сообщения.

Стеганография часто используется совместно с криптографией, дополняя ее. Стеганографические методы сокрытия сообщения снижают вероятность обнаружения самой передачи сообщения. Если сообщение к тому же зашифровано, то это обеспечивает еще большую защищенность.

В настоящее время наибольшее распространение получила цифровая стеганография. Особенностью цифровой стеганографии является сокрытие информации внутри других цифровых объектов, таких как текст, изображения, видео, аудио и другие. Со все большим возрастанием роли интернет-технологий в жизни человека значимость стеганографии также возрастает.

Области применения стеганографии включают в себя:

- а) Защита авторского права и DRM (digital rights management).
- б) Незаметная передача информации.
- в) Защита конфиденциальной информации от несанкционированного доступа.

Цифровая стеганография является молодым и бурно развивающимся направлением. В последние годы стеганография все больше находит применение в области защиты прав собственности на информацию.

В своей работе я хочу рассмотреть основные стеганографические алгоритмы, определить области их применения, достоинства и недостатки, а также провести анализ возможных уязвимостей этих алгоритмов.

# 1 Общие положения и описание стеганографических методов защиты информации

## 1.1 Основные понятия

а) Стеганографическая система (стегосистема) — совокупность методов и средств, используемых для создания скрытого канала для передачи информации. Основные требования, предъявляемые к стегосистеме:

1) Безопасность системы определяется секретностью ключа. Это означает, что даже если потенциальный враг представляет работу стеганографической системы и статистические характеристики сообщений и контейнеров, это не дает ему дополнительных преимуществ при выявлении наличия или отсутствия сообщения в конкретном контейнере.

2) При обнаружении противником наличия скрытого сообщения он не должен смошь извлечь сообщение до тех пор, пока он не будет владеть ключом.

3) Алгоритм сокрытия информации не нарушает ее целостность и аутентичность. В некоторых случаях дополнительно требуется, чтобы алгоритм обеспечивал целостность сообщения при деформации контейнера.

4) Система с цифровым водяным знаком должна иметь низкую вероятность ложного обнаружения скрытого сообщения.

б) Сообщение — информация, передачу которой нужно скрыть.

в) Контейнер — любая информация, используемая для сокрытия тайного сообщения. Контейнер может находиться в одном из двух состояний:

1) Пустой контейнер — контейнер, не содержащий сообщение.

2) Заполненный контейнер (стегоконтейнер) — контейнер, содержащий сообщение.

г) Стеганографический канал (стегоканал) — канал передачи стегоконтейнера.

д) Ключ (стегоключ) — секретный ключ, нужный для сокрытия стегоконтейнера. По аналогии с криптографией стегоключи подразделяются на 2 типа:

- 1) Закрытый стегоключ. В системах с закрытым стегоключем ключ должен быть создан до начала обмена сообщениями, либо передан по защищенному каналу связи.
- 2) Открытый стегоключ. Такой ключ может быть передан по открытому незащищенному каналу связи. Открытый стегоключ должен обладать таким свойством, чтобы по нему вычислительно нецелесообразно было восстанавливать закрытый ключ.

## 1.2 Цифровые водяные знаки и цифровые отпечатки

Цифровой водяной знак (ЦВЗ) — технология, созданная для защиты авторских прав на цифровые объекты. В связи с быстрым развитием информационных технологий все более актуальным становится вопрос защиты авторских прав и интеллектуальной собственности, представленной в цифровом виде. Примерами цифровых объектов могут выступать аудиозаписи, видеозаписи, изображения, электронные книги и другие. ЦВЗ могут быть как видимыми, так и невидимыми. Решение о наличии в цифровом объекте невидимого ЦВЗ принимаются на основе процедуры декодирования.

В общем виде стегосистема ЦВЗ может быть разбита на части следующим образом:

- а) Прекодер — часть, которая приводит ЦВЗ к удобному для встраивания в стегоконтейнер виду.
- б) Стегокодер — часть, которая вкладывает сообщение в стегоконтейнер.
- в) Выделение встроенного сообщения — процедура, выделяющая сообщение из стегоконтейнера.
- г) Стегодетектор — часть, определяющая наличие ЦВЗ.
- д) Декодер — часть, восстанавливающая исходное сообщение.

Контейнер, содержащий ЦВЗ, может подвергаться преднамеренным атакам или случайным помехам. Стегосистема ЦВЗ должна обеспечивать как различимость самого стегоконтейнера человеком, потому как в качестве стегоконтейнера выступает интеллектуальная собственность, направленная на конечного потребителя, так и различимость ЦВЗ стегодетекто-

ром, который может подтвердить или опровергнуть авторские права на интеллектуальную собственность. В связи с этим в стегосистемах ЦВЗ применяется помехоустойчивое кодирование и метод широкополосного сигнала.

Одной из основных характеристик ЦВЗ является надежность. Под надежностью понимается устойчивость к различным деформациям контейнера. По отношению к этой характеристике ЦВЗ распадается на три класса:

а) Хрупкие. Такие ЦВЗ разрушается при небольших модификациях заполненного стегоконтейнера. Такие ЦВЗ применяются для аутентификации сигнала. Например, такие ЦВЗ используются для подтверждения подлинности цифрового объекта.

б) Полухрупкие. Такие ЦВЗ чувствительны к некоторым преобразованиям контейнера и нечувствительны к другим. Например, ЦВЗ, встроенное в изображение, может быть нечувствительно к его компрессии, но в то же время быть чувствительно к вырезке из этого изображения фрагмента.

в) Робастные или надежные. Такие ЦВЗ устойчивы к разным видам воздействия на контейнер. Такие ЦВЗ часто применяются при защите от копирования.

Чтобы осуществить вложение ЦВЗ в стегоконтейнер, ЦВЗ преобразуют к более удобному виду. Например, если ЦВЗ является изображением, то удобно будет представить его как двумерную битовую матрицу. Также если ЦВЗ является изображением, разумно будет использовать не само изображение, а его Вайвлет преобразование или дискретное косинусное преобразование. Изображения обладают большой визуальной избыточностью. Данные преобразования концентрируют большую часть энергии (визуальной информации) в нижних частотах. Поэтому их можно использовать как низкочастотные фильтры. То же самое относится и к контейнерам.

Похожим на ЦВЗ, но отличающимся понятием является цифровой отпечаток. ЦВЗ предполагает встраивание одного и того же сообщения в различные контейнеры. В случае же цифрового отпечатка в каждый контейнер встраивается уникальное сообщение. Часто областью применения цифровых отпечатков становится защита исключительного права. В каче-

стве сообщения в таком случае встраивается информация, указывающая на идентифицирующие данные покупателя. Эти данные позволяют отследить источник распространения, если произойдет утечка стегоконтейнера.

### 1.3 Обобщенные стеганографические методы

К настоящему моменту разработано множество стеганографических методов сокрытия информации. Для их систематичного изучения удобно группировать их по схожим признакам.

а) Пространственные методы. Особенностью этих методов является сокрытие информации напрямую в пространственной области контейнера. Например, в случае звукового контейнера таким пространством могут быть семплы, а в случае изображения — пиксели.

б) Частотные методы. Такие методы сначала используют одно из интегральных преобразований сигнала, чтобы перейти в его частотную область. Далее кодирование сообщение производится за счет изменения частотных характеристик сигнала. После этого используется обратное преобразование, чтобы получить модифицированный сигнал, содержащий закодированное сообщение.

в) Алгоритмы, использующие особенности формата файла. Такие алгоритмы, как правило, записывают сообщение в метаданные файла или в иные неиспользуемые поля файла.

### 1.4 Стегоанализ

Стегоанализ — это наука о выявлении сообщений, скрытых методами стеганографии. Задача стегоанализа — выявить подозрительные контейнеры, определить, есть ли в них скрытое сообщение, и, если возможно, восстановить это сообщение.

Если в случае криptoанализа аналитик начинает работу сразу с зашифрованным сообщением, то в случае стегоанализа аналитик начинает работу с множества подозрительных файлов, о которых, как правило, мало что известно. Деятельность аналитика в таком случае начинается с сокра-

щения этого множества файлов до подмножества, в котором файлы, скорее всего, были заполнены сообщением.

Самой простым методом стегоанализа является субъективная атака. Атака заключается в попытке “на глаз” определить, содержит тот или иной контейнер стего. Несмотря на свою простоту, атака часто применяется на начальном этапе стегоанализа системы.

Основной техникой, используемой в стегоанализе, является статистический анализ. Сначала множество незаполненных контейнеров одного типа анализируется для получения различной статистики. Затем эта статистика используется при классификации контейнера как заполненного или пустого. При такой классификации могут быть использованы самые различные наблюдения:

- а) Сокрытие информации может приводить к изменению статистической структуры контейнера, в результате чего соседние элементы контейнера становятся попарно ближе друг к другу. На этом основана атака хи-квадрат.
  - б) Сокрытие информации увеличивает энтропию контейнера. В результате чего он хуже поддается сжатию. На этом основана атака с помощью алгоритмов сжатия.
  - в) Различные статистические данные контейнера и его областей можно использовать как вектор признаков. Собрав большой датасет таких признаковых описаний объектов стего, на нем можно обучить нейросеть, которая будет классифицировать изображения.
- .

## 2 Сокрытия методом наименее значимого бита

### 2.1 Общие сведения

LSB (least significant bit) — стеганографический метод сокрытия информации, основанный на замене последних значащих бит элементов контейнера битами сообщения. Этот метод использует тот факт, что уровень детализации во многих контейнерах гораздо выше того, что может воспринять и различить человек. Следовательно, заполненный контейнер будет неотличим от оригинального для человеческого восприятия. В качестве примера можно взять полутоновое изображение с градациями серого. Цвет кодируется одним байтом. Человеческий глаз воспринимает только первые 7 бит, а самый младший бит вносит там мало информации, что человек не сможет заметить разницу.

LSB обладает следующими достоинствами:

- а) Простота реализации и эффективность.
- б) Низкая вычислительная сложность.
- в) Пустой и заполненный контейнер неразличимы для органов восприятия человека

И недостатками:

- а) Метод применим лишь к контейнерам, которые хранят данные без сжатия или используют сжатие без потерь, так как информация, закодированная в наименее значимых битах, может быть потеряна в процессе сжатия.
- б) Небольшие трансформации контейнера приводят к невозможности восстановить сообщение. Например, если сообщение скрыто в изображении методом LSB, то небольшие линейные трансформации (вращение, движение, отражение, гомотетия, сжатие, растяжение) уничтожают сообщение. Также сообщение разрушается в результате сжатия с потерями. Все это говорит о том, что метод обладает низкой робастостью.
- в) Факт сокрытия изображения легко обнаруживается методами стеганализа.

Ввиду перечисленных выше недостатков, очевидной кажется недопустимость использования данного метода для сокрытия ЦВЗ.

## 2.2 Алгоритм

Перейдем к конкретным реализациям этого метода. Алгоритм 1 демонстрирует псевдокод сокрытия методом LSB.

---

### Алгоритм 1: LSB Кодирование

---

**Data:** Контейнер, Сообщение

**Result:** Заполненный стегоконтейнер

$N \leftarrow$  Длина сообщения в битах;

$Message \leftarrow$  Бинарное представление сообщения;

$Container \leftarrow$  Массив с элементами контейнера;

**for**  $i = 1, 2, \dots, N$  **do**

**if**  $Container[i] \equiv Message[i] \pmod{2}$  **then**

**continue**;

**else**

$Container[i] \leftarrow (Container[i] \wedge \neg 1) \vee Message[i];$

---

---

### Алгоритм 2: LSB Декодирование

---

**Data:** Заполненный контейнер

**Result:** Сообщение в бинарном представлении

$Message \leftarrow$  Пустой список;

$Container \leftarrow$  Массив с элементами контейнера;

$N \leftarrow$  Длина  $Container$ ;

**for**  $i = 1, 2, \dots, N$  **do**

**if**  $Container[i] \equiv 0 \pmod{2}$  **then**

$Message.append(0);$

**else**

$Message.append(1);$

---

Как видно, сначала сообщение преобразуется в бинарный вид, а затем кодируется в элементах контейнера за счет изменения четности младшего бита. Логические операции в данном случае соответствуют бинарным операциям на компьютере. В итоге последние биты элементов контей-

нера в точности повторяют сообщение. Также можно заметить, что единственная часть алгоритма, зависящая от контейнера — это выделение массива элементов из контейнера. Алгоритм 2 показывает, как декодировать сообщение из заполненного стегоконтейнера.

Реализуем этот алгоритм в виде класса на Python. Как уже было сказано, существенная часть алгоритма не зависит от контейнера, поэтому целесообразно реализовать алгоритм как абстрактный класс, от которого будут наследоваться реализации для конкретных контейнеров. Реализация приведена в листинге 2.1.

### Листинг 2.1 — Абстрактный класс LSB

```
1 import numpy as np
2 from abc import ABC, abstractmethod
3
4
5 class LSB(ABC):
6     def __init__(self, container, message: bytes = None) -> None:
7         """
8             Возвращает простой lsb кодер—,
9             принимает на вход контейнер и сообщение массив( байт).
10            """
11
12     if message is None:
13         # По умолчанию сообщение пустое
14         self.message = []
15
16     else:
17         self.message = message
18
19     self._container = container
20
21     def encode(self) -> None:
22         """
23             Кодирует сообщение в контейнер.
24            """
25
26     # Получаем последовательность элементов контейнера
27     elements = self._to_elements()
28
29     # Преобразуем сообщение к бинарному виду
30     np_message = np.unpackbits(np.frombuffer(
31         self.message, dtype=np.uint8)).ravel()
32
33     n = len(np_message)
34
35     # Меняем наименее значимый бит так,
36     # чтобы он кодировал биты сообщения
37     elements[:n] = (elements[:n] & ~1) | np_message
```

```

33     # Из элементов собираем контейнер обратно
34     self._from_elements(elements)
35
36     def decode(self) -> bytes:
37         """
38             Декодирует сообщение из контейнера.
39         """
40
41         # Получаем последовательность элементов контейнера
42         elements = self._to_elements()
43
44         # Выбираем размер сообщения так, чтобы он был кратен размеру байта
45         size = len(elements) // 8 * 8
46
47         # Сообщение считываем из наименее значащих бит элементов контейнера
48         np_message = (elements[:size] & 1)
49
50         # Преобразуем битовую последовательность в байты
51         message = np.packbits(np_message.reshape(-1, 8), axis=-1).tobytes()
52
53         return message
54
55     @abstractmethod
56     def _to_elements(self) -> np.array:
57         """
58             Преобразует контейнер в последовательность элементов.
59         """
60
61         pass
62
63     @abstractmethod
64     def _from_elements(self, elements: np.array) -> None:
65         """
66             Собирает контейнер из элементов.
67         """
68
69         pass

```

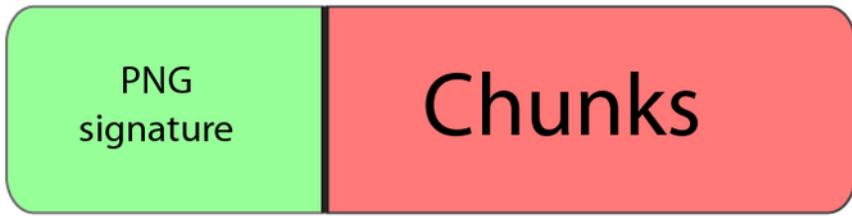
Как видно, в методах нет циклов **for**. Они скрыты за интерфейсом библиотеки `pintpy`. Интерфейс библиотеки `pintpy` позволяет нам применять операции к матрицам, из-за чего код выглядит лаконичнее. К тому же библиотека написана на языке C, поэтому ее код работает очень быстро.

Напишем реализацию LSB для PNG. Прежде чем реализовать метод LSB для PNG-контейнера, имеет смысл кратко изложить формат данных PNG.

## 2.3 Формат файла PNG

В самом общем виде PNG файл представляет из себя сигнатуру, за которой следует последовательность блоков, как показано на рисунке 2.1

Рисунок 2.1 — Общий вид формата PNG



Сигнатурой PNG файла состоит из 8 байт, в hex нотации они выглядят так: **89 50 4E 47 0D 0A 1A 0A**.

Каждый блок состоит из четырех секций: длина, тип, содержание, CRC, — как показано на рисунке 2.2: В длине указывается длина блока в

Рисунок 2.2 — Общий вид чанка

Length (длина)	Тип (имя) чанка	Содержание чанка	CRC
4 байта	4 байта	<i>Length</i> байт	4 байта

байтах. Тип указывается с помощью четырех ascii символов, чувствительных к регистру. С помощью регистра декодеру передается дополнительная информация, а именно:

- Регистр первого символа сообщает, является данный блок критическим или нет. Критические блоки распознаются каждым декодером. Если декодер не может распознать тип такого блока, он аварийно завершает работу.
- Регистр второго символа задает публичность или приватность блока. Публичные блоки обычно официальные и хорошо задокументированы. Чтобы закодировать в библиотеку какую-то специфичную информацию, его тип можно изменить на приватный.
- Регистр третьего символа зарезервирован на будущее. По умолчанию там стоит символ в большом регистре.

г) Регистр четвертого символа сообщает возможность копирования данного блока редакторами.

Список критических блоков:

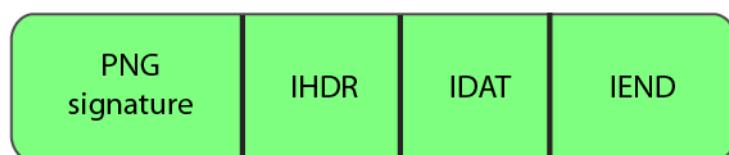
- а) IHDR — заголовочный блок, содержащий основную информацию об изображении.
- б) PLTE — палитра изображения.
- в) IDAT — содержит непосредственно изображение. В любом PNG файле должно быть не менее одного такого блока.
- г) IEND — завершающий чанк. Должен находиться в самом конце файла.

Список некритических блоков:

- а) bKGD — блок, задающий фоновый цвет изображения.
- б) cHRM — блок, используемый для задания цветового пространства CIE 1931.
- в) gAMA — определяет гамму.
- г) hIST — хранит гистограмму изображения либо общее содержания каждого цвета в рисунке.
- д) iTXT — содержит текст в UTF-8
- е) pHYs — содержит размер пикселя или отношение сторон изображения.
- ж) sRGB — свидетельствует об использовании sRGB схемы.
- и) tIME — дата последнего изменения изображения.
- к) tRNS — информация о прозрачности.

Согласно вышесказанному, минимальный PNG файл выглядит так, как показано на рисунке 2.3

Рисунок 2.3 — Минимальный PNG



В следующей секции представлены данные блока. В секции CRC записан CRC блока.

Наиболее интересными для нас являются блоки с типами IHDR и IDAT. IHDR — заголовочный блок, который является обязательным для PNG файла. Он содержит следующие интересующие нас поля:

- а) Ширина изображения в пикселях.
- б) Высота изображения в пикселях.
- в) Битовая глубина, задающее количество бит на каждый сэмпл.
- г) Тип цвета. Возможны следующие значения:
  - 1) Градация серого
  - 2) RGB
  - 3) Индексы из палитры
  - 4) Градация серого и альфа-канал
  - 5) RGB и альфа-канал

Блок IDAT содержит сжатые данные изображения. На данный момент поддерживается только сжатие по алгоритму deflate.

## 2.4 Реализация LSB для контейнера PNG

PNG изображение представимо в виде матрицы, элементами которой являются пиксели. В случае RGB каждый пикセル представляет элементы из трех каналов, каждый из каналов в отдельности может рассматриваться как градация серого. В случае градации серого каждый пикセル просто представлен значением от 0 до 255. Чтобы закодировать сообщение в эту матрицу, склеим ее строки друг с другом в одну большую строку, равно как и склеим каналы, чтобы они образовали последовательность элементов. Именно это делает метод `_to_elements`. Такой метод одинаково хорошо подходит и для разных типов цвета: RGB, RGB и альфа-канала, градации серого, градации серого и альфа-канала. Чтобы из элементов получить двумерную RGB матрицу, проделаем обратную операцию, что и делает метод `_from_elements`.

В функции `main` используем как сообщение книгу “Алиса в стране чудес” в оригинале. Считаем файл с книгой как последовательность байт и закодируем в изображение с помощью LSB. Далее выполним декодиро-

вание и сверим полученные данные. Исходный код представлен в листинге 2.2.

### Листинг 2.2 — реализация LSB для PNG

```
1 import numpy as np
2 from lsb import LSB
3 from PIL import Image
4
5
6 class PNG(LSB):
7     """
8     Реализация алгоритма LSB для файлов формата PNG.
9     """
10
11     def __init__(self, file_name: str, message: bytes = None) -> None:
12         """
13         Возвращает простой PNG кодер,
14         принимает на вход имя файла и сообщение.
15         """
16         self._file_name = file_name
17         container = np.array(Image.open(file_name))
18         super().__init__(container, message)
19
20     def _to_elements(self) -> np.array:
21         """
22         Возвращает представление контейнера
23         как последовательности элементов.
24         """
25         return self._container.ravel()[:]
26
27     def _from_elements(self, elements: np.array) -> None:
28         """
29         Строит контейнер по последовательности
30         элементов.
31         """
32         self._container.ravel()[:] = elements
33
34     def save(self) -> None:
35         """
36         Перезаписывает исходный файл
37         новым контейнером.
38         """
39         image = Image.fromarray(self._container)
40         image.save(self._file_name)
41
42     def save_as(self, file_name: str) -> None:
```

```

43     """
44     Сохраняет контейнер в файл,
45     заданный параметром file_name.
46     """
47     image = Image.fromarray(self._container)
48     image.save(file_name)
49
50
51 def main() -> None:
52     # Считываем сообщение.
53     with open("Messages/Alice in wonderland.txt", "rb") as f:
54         message = f.read()
55
56     # Запоминаем длину сообщения.
57     size = len(message)
58     # Кодируем сообщение.
59     png = PNG("Images/Lenna.png", message)
60     png.encode()
61     png.save_as("Images/LSB_Lenna.png")
62     # Переоткрываем изображение.
63     png = PNG("Images/LSB_Lenna.png")
64     # Декодируем сообщение.
65     decoded = png.decode()
66     # Проверяем, что сообщения до и после совпадают.
67     new_message = decoded[:size]
68     print(f"{new_message == message}")
69
70
71 if __name__ == "__main__":
72     main()

```

Сравнение изображение до и после заполнения контейнера методом LSB приведено на рисунке 2.4. Как видно, два рисунка визуально неотличимы, хотя в одном из них закодировано 150 килобайт информации.



(a) Оригинал



(b) После применения LSB

Рисунок 2.4 — Изображение до и после применения LSB

### 3 Сокрытие в спектральной области

#### 3.1 Дискретное косинусное преобразование

Дискретное косинусное преобразование (ДКП) — одно из дискретных преобразований Фурье. ДКП представляет конечную последовательность в виде суммы функций косинуса, колеблющихся на разных частотах. ДКП широко используется при обработке сигналов и сжатии данных. Например, ДКП используется при сжатии в изображениях (JPEG, HEIF), аудиофайлах (Dolby Digital, MP3), видеофайлах (MPEG, H.26x), в цифровом телевидении (SDTV, HDTV, VOD) и в других.

ДКП является линейным ортогональным преобразованием. Как и любое дискретное линейное преобразование, ДКП можно представить в виде матрицы. Будучи ортогональным преобразованием, обратное к ДКП преобразование задается транспонированной матрицей ДКП, домноженной на какой-то коэффициент.

Использование косинусных, а не синусоидальных функций имеет решающее значение для сжатия, поскольку для аппроксимации типичного сигнала требуется меньше косинусных функций. ДКП подобно дискретному преобразованию Фурье, но использует только действительные числа.

Существует 8 стандартных типов ДКП, однако наиболее употребляемым является второй тип, который часто называют просто ДКП (DCT-II). Формула дискретного косинусного преобразования выглядит так, как показано в формуле 3.1:

$$X_k = \sum_{n=0}^{N-1} x_n \cos \left[ \frac{\pi}{N} \left( n + \frac{1}{2} \right) k \right] \quad k = 0, \dots, N - 1 \quad (3.1)$$

Формула для матрицы преобразования выглядит как в 3.2:

$$DCT-2_n = \left[ \cos \left( k \left( l + \frac{1}{2} \right) \frac{\pi}{n} \right) \right]_{0 \leqslant k, l < n} \quad (3.2)$$

Как и в случае быстрого преобразования Фурье, существуют алгоритмы быстрого ДКП преобразования.

DCT-II часто используется для сжатия с потерями благодаря своему свойству уплотнения энергии: в типичных случаях большая часть инфор-

мации, которую содержит сигнал, концентрируется в нескольких первых коэффициентах разложения.

Существуют так же многомерные ДКП, которые получаются из одномерных путем композиции ДКП по каждому измерению. Вывод такого преобразования для двумерного случая показан в формуле 3.3.

$$\begin{aligned} X_{k_1, k_2} &= \sum_{n_1=0}^{N_1-1} \left( \sum_{n_2=0}^{N_2-1} x_{n_1, n_2} \cos \left[ \frac{\pi}{N_2} \left( n_2 + \frac{1}{2} \right) k_2 \right] \right) \cos \left[ \frac{\pi}{N_1} \left( n_1 + \frac{1}{2} \right) k_1 \right] \\ &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1, n_2} \cos \left[ \frac{\pi}{N_1} \left( n_1 + \frac{1}{2} \right) k_1 \right] \cos \left[ \frac{\pi}{N_2} \left( n_2 + \frac{1}{2} \right) k_2 \right] \end{aligned} \quad (3.3)$$

Здесь  $[x_{n_1, n_2}]$  — матрица до преобразования, и  $[X_{k_1, k_2}]$  — матрица после преобразования. В матричном виде это преобразование может быть представлено так, как показано в формуле 3.4, где  $x$  — матрица, которую нужно преобразовать.

$$X = (DCT-2_n)x(DCT-2_n^T) \quad (3.4)$$

Именно такое преобразование используется при компрессии в JPEG.

## 3.2 JPEG

JPEG является широко используемым методом сжатия с потерями для цифровых изображений. Степень сжатия регулируется, что позволяет выбирать между качеством и размером изображения. JPEG также является наиболее широко используемым стандартом сжатия изображений в мире и наиболее используемым форматом цифровых изображений.

ДКП лежит в основе сжатия методом JPEG. Как уже говорилось выше, ДКП был выбран именно благодаря свойству уплотнения энергии. Чтобы прояснить, о чём идет речь, мной была создана визуализация преобразования ДКП.

Выберем на изображении область 32x32 пикселя, как показано на рисунке 3.1.

Для простоты будем использовать только синий канал изображения. Сначала рассмотрим матрицу пикселей как двумерную дискретную функцию. Расположим координаты так, чтобы в левом верхнем углу распола-



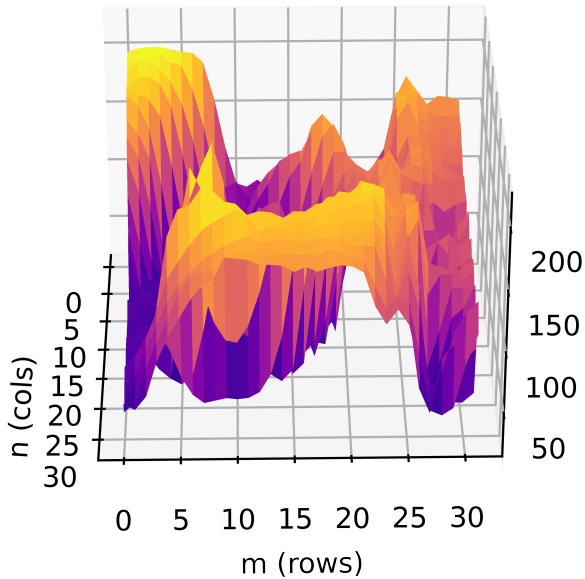
Рисунок 3.1 — Выбираем область

гался пиксель с координатами  $p_{k,j}, k = 0, j = 0$ . Визуализацию можно посмотреть на рисунке 3.2.

Умножим эту функцию справа на транспонированную матрицу ДКП по формуле 3.4. Мы получим новую функцию, которая показана на рисунке 3.3. Таким образом, фактически ДКП применилось к каждой строке матрицы. Из изображения видно, что наибольшие коэффициенты расположены в нижней части спектра, то есть ближе к нулевому столбцу.

К полученной матрице применим ДКП еще раз, в этот раз по столбцам. В полученной матрице наибольшее значение имеет коэффициент с координатами  $k = 0, j = 0$ . Этот коэффициент называется DC-коэффициент. Остальные коэффициенты называются AC-коэффициентами. Матрица показана на рисунке 3.4.

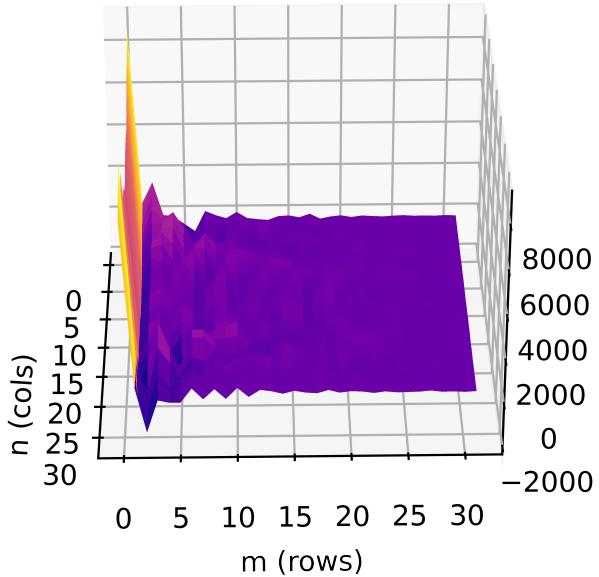
Рисунок 3.2 — Визуализация пикселей



DC-коэффициент блока равен среднему всех пикселей в блоке, взятым с определенным коэффициентом. Удаляя все коэффициенты, кроме DC, мы можем аппроксимировать блок пикселей их средним арифметическим. Чем дальше коэффициент располагается от DC, тем меньше психовизуальной информации он несет для человека, и тем более незаметные детали изображения он хранит в себе. Соответственно, основная идея алгоритма состоит в отбрасывании наименее значимых коэффициентов. Это позволяет производить сжатие изображения с потерями.

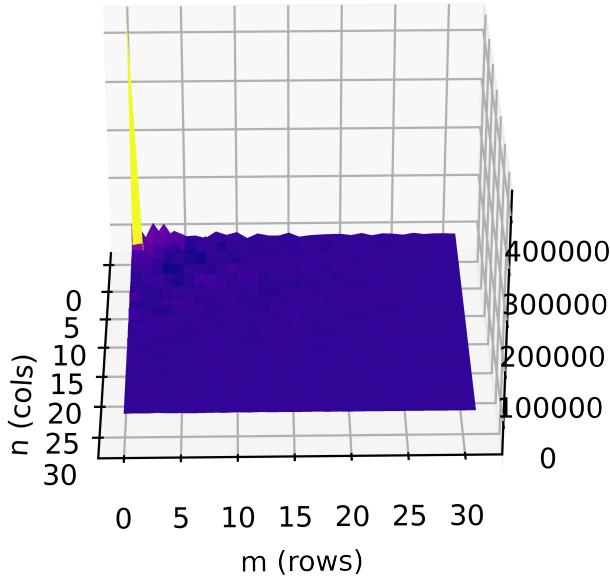
Алгоритм сжатия JPEG работает с каждым каналом отдельно, поэтому для простоты рассмотрим работу JPEG на изображении в режиме градации серого. В самом начале своей работы алгоритм разбивает изображение на блоки 8x8 пикселей. К каждому блоку применяется ДКП преобразование, что равносильно разложению исходной матрицы по базису, состоящему из 64 функций. Эти 64 функции показаны на рисунке 3.5. Из этого рисунка видно, что по мере отдаления от левого верхнего угла функции становятся все более рельефными, что объясняет, почему они несут

Рисунок 3.3 — После применения ДКП к строкам матрицы



наиболее мелкие детали изображения. Также видно, что функция, соответствующая DC-коэффициенту, представлена плоскостью. Очевидно, что лучшим константным приближением функции является ее математическое ожидание. После применения ДКП преобразования к блоку матрицы  $8 \times 8$  получается другая матрица той же размерности. В соответствии с выше-сказанным, эта матрица делится на области низких, средних и высоких частот. В таком порядке убывает информативность коэффициентов. Это можно увидеть на рисунке 3.6. Далее коэффициенты полученной ДКП матрицы квантуются. Квантование происходит с применением специальных

Рисунок 3.4 — После применения ДКП к столбцам матрицы

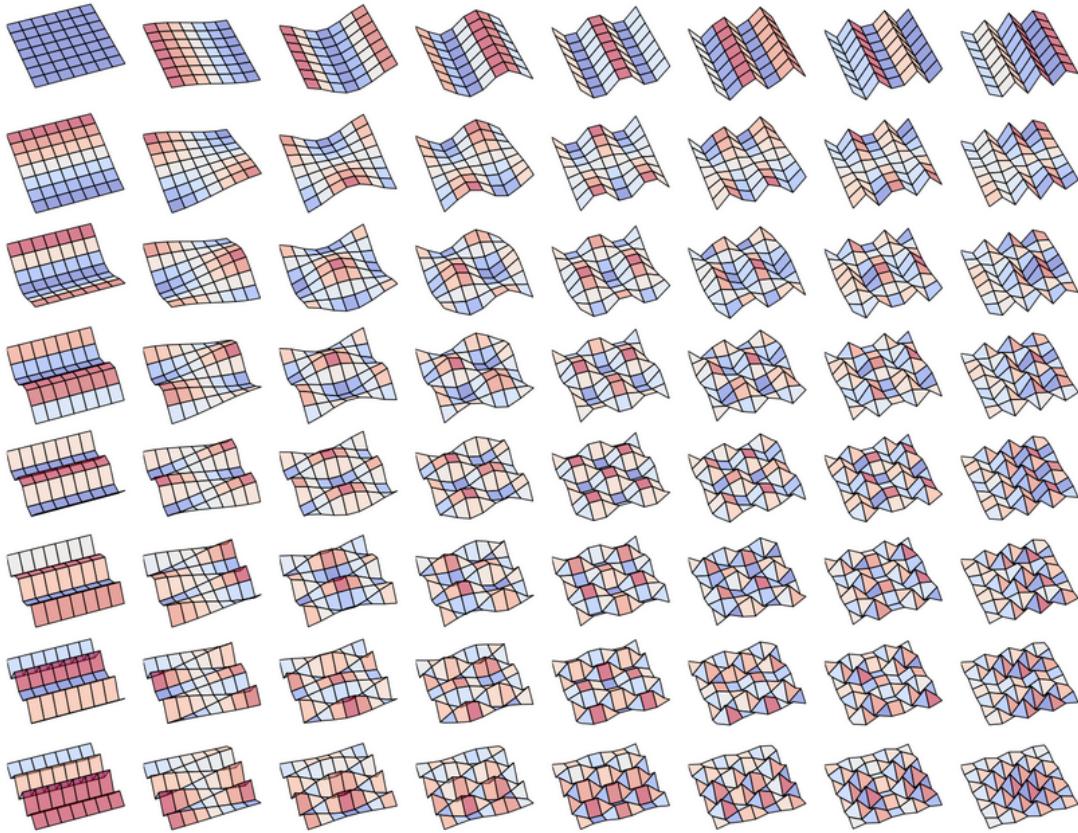


матриц, одна из таких матриц представлена в формуле 3.5.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}. \quad (3.5)$$

Это матрица для 50% качества, указанная в исходном стандарте JPEG. Каждый элемент ДКП матрицы делится на элемент матрицы квантования, стоящий в той же позиции. После этого результат округляется. В результате этой операции обычно бывает так, что многие высокочастотные компоненты округляются до нуля, а многие из остальных становятся небольшими положительными или отрицательными числами, для представления которых требуется гораздо меньше бит.

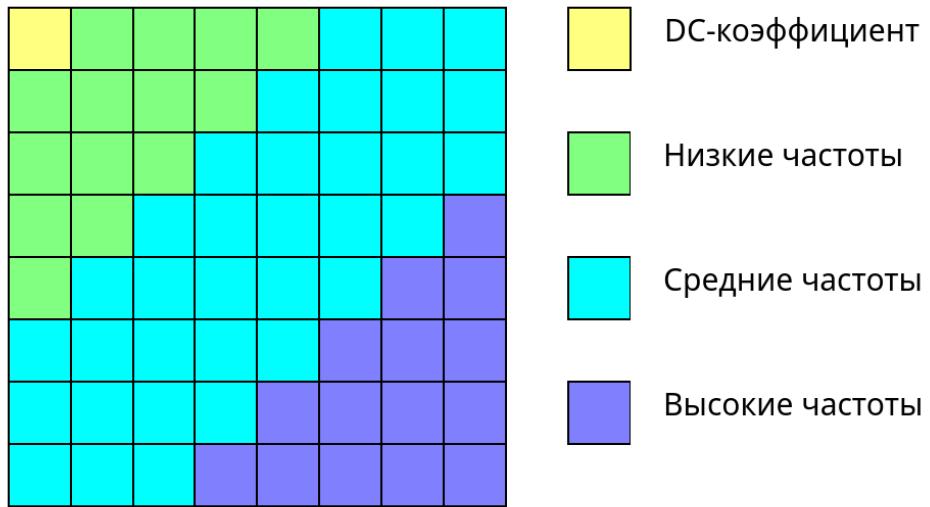
Рисунок 3.5 — Базис ДКП



При декодировании происходит обратный процесс — матрица квантованных коэффициентов почленно умножается на матрицу квантования, но из-за того, что до этого значения были округлены, они восстанавливаются с некоторой погрешностью. Чем больше коэффициент квантования, тем больше будет эта погрешность.

После квантования коэффициенты записываются в специальном зигзагообразном порядке, показанном на рисунке 3.7. Таким образом коэффициенты упорядочиваются от низких частот к высоким. После этого DC и AC коэффициенты кодируются отдельно. Поскольку в изображениях часто встречаются градиентные области, то DC коэффициенты соседних блоков коррелированы, поэтому первым этапом их кодирования становится дифференциальная импульсно-кодовая модуляция (ДИКМ). То есть кодируются не сами коэффициенты, а разница между двумя соседними коэффициентами. AC коэффициенты кодируются с помощью кодирования длин серий (КДС). То есть повторяющиеся символы заменяются на сим-

Рисунок 3.6 — Частотные области ДКП



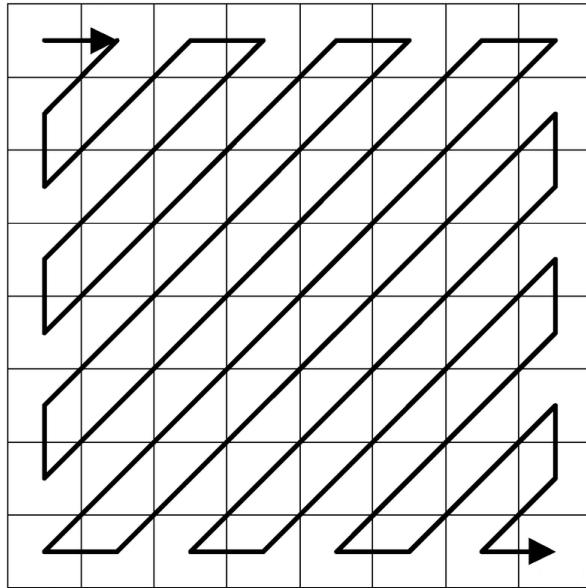
вов и количество его повторов. После этого к DC и AC коэффициентам применяется энтропийное кодирование с помощью алгоритма Хаффмана.

Мы разобрали работу алгоритма для одноканального изображения. В RGB изображениях такой алгоритм применяется к каждому каналу отдельно. Также часто кодированию предшествует дополнительный этап, на котором RGB преобразуется в цветовое пространство  $Y\text{C}_\text{B}\text{C}_\text{R}$ . Дело в том, что человеческий глаз более чувствителен к перепадам яркости, чем к перепаду цвета. В  $Y\text{C}_\text{B}\text{C}_\text{R}$  первый канал  $Y$  отвечает за яркость,  $\text{C}_\text{B}$  и  $\text{C}_\text{R}$  отвечают за синий и красный компоненты. После преобразования в  $Y\text{C}_\text{B}\text{C}_\text{R}$  над  $\text{C}_\text{B}$  и  $\text{C}_\text{R}$  производится субдискретизация: каналы разбиваются на небольшие блоки и значения пикселей в этих блоках усредняются. Таким образом разрешение в этих каналах понижается еще сильнее и изображение сжимается еще сильнее. Вся схема алгоритма представлена на рисунке 3.8

### 3.3 JSteg

JSteg — стеганографический алгоритм, работающий с JPEG файлами. JSteg во многом опирается на работу кодировщика JPEG. Алгоритмы кодирования и декодирования JPEG абсолютно симметричны. JSteg вмешивается в работу декодировщика, а именно прерывает его на этапе умножения ДКП коэффициентов на матрицу квантования. После этого JSteg записывает в упорядоченные зигзагообразным образом ДКП коэффи-

Рисунок 3.7 — Зигзагообразный порядок



циенты кодируемую информацию методом LSB. После этого вызывается кодировщик, который записывает измененные ДКП коэффициенты обратно в JPEG изображение.

Рассмотрим достоинства и недостатки этого метода. К достоинствам можно отнести следующее:

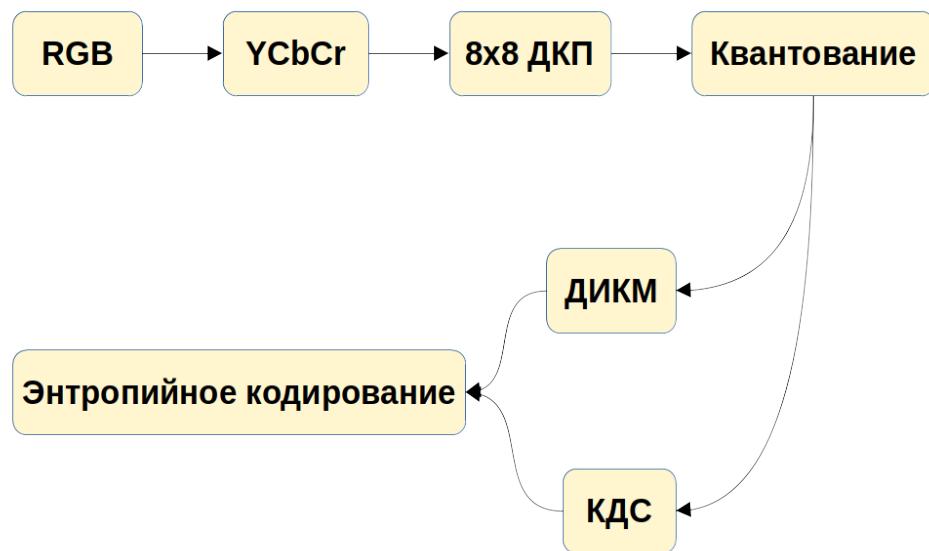
- а) Низкая вычислительная сложность.
- б) Алгоритм обеспечивает большую вместимость стегосообщений: стегосообщение может занимать до 13% объема контейнера.
- в) Изменения, вносимые в контейнер, незаметны для человеческого глаза.

Но у метода также есть и существенные недостатки:

а) Метод неустойчив к квантованию ДКП коэффициентов. Как уже говорилось ранее, операция квантования восстанавливает и сохраняет коэффициент с некоторой погрешностью, поэтому если открыть в редакторе стегоконтейнер, в котором содержится сообщение, закодированное методом JSteg, то после пересохранения этого файла сообщение полностью уничтожится.

б) Из предыдущих соображений становится ясно, что метод неустойчив к сжатию. Это также обусловлено еще и тем, что при сжатии коэффициенты квантования увеличиваются, а значит часть ДКП коэффициентов

Рисунок 3.8—Схематичное изображение JPEG



обнулится, а у другой части погрешность восстановления станет еще больше.

Рассмотрим реализацию метода на Python. Код приведен в листинге 3.1. По сути метод представляет собой LSB, только вместо пространственной области используется спектральная. В данном случае наименее значимый бит меняется у коэффициентов ДКП изображения, упорядоченных зигзагообразным способом.

Листинг 3.1 — Реализация JSteg

```
1 import jpegio as jio
2 import numpy as np
3 import cv2
4 from lsb import LSB
5
6
7 class JSteg(LSB):
8     """
9         Реализация стеганографического алгоритма JSteg .
10    """
11
12    def __init__(self, file_name: str, message: str = None) -> None:
13        """
14            Возвращает простой JSteg кодер ,
15            принимает на вход имя файла и сообщение .
16        """
```

```

16     """
17     self.file_name = file_name
18     # Считываем все коэффициенты ДКП
19     self.dct = jio.read(self.file_name)
20     # Оставляем только  $C_b$  канал.
21     # Коэффициенты упорядочены в зигзагообразном порядке
22     container = self.dct.coef_arrays[1]
23     super().__init__(container, message)
24
25     def _to_elements(self) -> np.array:
26         """
27             Возвращает репрезентацию контейнера
28             как последовательности элементов.
29         """
30
31         return self._container.ravel()[:]
32
33     def _from_elements(self, elements: np.array) -> None:
34         """
35             Строит контейнер по последовательности
36             элементов.
37         """
38
39         self.dct.coef_arrays[1].ravel()[:] = elements
40
41     def save(self) -> None:
42         """
43             Перезаписывает исходный файл
44             новый контейнером.
45         """
46
47         jio.write(self.dct, self.file_name)
48
49     def save_as(self, file_name: str) -> None:
50         """
51             Сохраняет контейнер в файл,
52             заданный параметром file_name.
53         """
54
55         jio.write(self.dct, file_name)
56
57     def main() -> None:
58         """
59             Проверяет работоспособность программы.
60         """
61
62         # Считываем сообщение.
63         with open("Messages/Alice in wonderland.txt", "rb") as f:
64             message = f.read()[:80000]

```

```

62     # Запоминаем длину сообщения.
63     size = len(message)
64     # Кодируем сообщение.
65     jsteg = JSteg("Images/Lenna.jpg", message)
66     jsteg.encode()
67     jsteg.save_as("Images/JSteg_Lenna.jpg")
68     # Переоткрываем изображение.
69     jsteg = JSteg("Images/JSteg_Lenna.jpg")
70     # Декодируем сообщение.
71     decoded = jsteg.decode()
72     # Проверяем, что сообщения до и после совпадают.
73     new_message = decoded[:size].decode()
74     print(f"{new_message == message.decode()}")
75
76
77 if __name__ == "__main__":
78     main()

```

### 3.4 Метод относительной замены величин коэффициентов ДКП

Этот метод использует идеи, схожие с методами расширения спектра, а именно: вместо того, чтобы кодировать 1 бит информации в одном коэффициенте ДКП, метод предлагает кодировать 1 бит информации за счет нескольких коэффициентов ДКП. Этого можно добиться, кодируя информацию за счет изменения разности между набором различных коэффициентов ДКП.

Алгоритм Коха-Жао использует 2 коэффициента ДКП. Формальное описание приводится в алгоритме 3. Алгоритм декодирования строится симметрично. Это простейший метод из данного семейства. К достоинствам метода можно отнести то, что он устойчив к квантованию ДКП-коэффициентов и сжатию. Особенno если применять его в паре с помехоустойчивым кодированием. Но у метода также есть и серьезные недостатки:

- а) Метод вносит заметные искажения в контейнер.
- б) Метод легко детектируется.

Модифицированной версией этого метода является метод Бенгама-Мемона-Эо-Юнга. Модификации подверглись два направления:

---

**Алгоритм 3:** Алгоритм Коха-Жао

---

**Data:** Контейнер, Сообщение

**Result:** Заполненный стегоконтейнер

$N \leftarrow$  Длина сообщения в битах;

$Message \leftarrow$  Бинарное представление сообщения;

$DCT-blocks \leftarrow$  Массив из блоков ДКП контейнера;

$k, l \leftarrow$  Позиция коэффициента ДКП из низкой полосы частот;

**for**  $i = 1, 2, \dots, N$  **do**

**if**  $Message[i] = 0$  **then**

        Сделать  $|DCT-blocks[i][k][l] - DCT-blocks[i][k][l]| < 25$ ;

**else**

        Сделать  $|DCT-blocks[i][k][l] - DCT-blocks[i][k][l]| > 25$ ;

$Container \leftarrow$  Новый контейнер, полученный из обратного

преобразования ДКП-блоков;

---

- а) Встраивание происходит только в наиболее подходящие ДКП-блоки.
- б) Используются не 2, а 3 коэффициента ДКП. Это существенно снижает вносимые в контейнер искажения.
- . Рассмотрим каждую модификацию в отдельности.

Наиболее подходящие коэффициенты выбираются по следующим признакам:

- а) Блок не должен иметь слишком резких переходов яркости.
- б) Блок не должен быть слишком монотонным.

Для оценки этих параметров вводится два коэффициента:  $P_L$  и  $P_H$ . Превышение первого коэффициента или недостижение второго будет указывать на то, что блок не пригоден для встраивания. Для получения первой оценки нужно просуммировать модуляции низкочастотных коэффициентов, а для получения второй оценки нужно просуммировать модули высокочастотных коэффициентов.

Само встраивание происходит в два этапа. На первом этапе выбираются три коэффициента из низкой полосы частот. Для обеспечения большой стойкости они могут выбираться псевдослучайно. На втором этапе

коэффициенты модифицируются. Если кодируется 0, то третий коэффициент должен стать больше первых двух, а если 1, то третий коэффициент должен стать меньше, соответственно. После встраивания запоминаются номера блоков, в которых закодирована информация. Декодирование происходит симметричным образом с той лишь разницей, что вместо поиска подходящих блоков используются запомненные на предыдущем этапе блоки. Реализацию метода на Python можно найти в приложении А.

## 4 Стегоанализ

### 4.1 Методы стегоанализа

Основными методами, применяемыми в стегоанализе, являются визуальные методы и статистические методы.

Субъективная атака проста по своей сути: аналитик пытается “на глаз” определить, содержит ли контейнер стего. Однако эта атака может применяться в различных вариациях. Например, анализу может подвергаться не само изображение, а какой-то его канал, или же изображение, полученное из данного отбрасыванием нескольких старших бит.

Статистические методы основаны на использовании различных статистик изображения и том факте, что эти статистики могут различаться для пустого и заполненного стегоконтейнера.

### 4.2 Субъективная атака LSB

Метод LSB является легко обнаружимым. Для начала рассмотрим самую простую ситуацию: допустим, что сообщение, скрытое в стего, не зашифровано. В таком случае стегоконтейнер поддается визуальной атаке, а именно: попробуем посмотреть на визуальное представление последнего бита сообщения. Для этого используем код, представленный в листинге 4.1.

Листинг 4.1 — Визуализация LSB

```
1 import numpy as np
2 import cv2
3
4
5 def main() -> None:
6     """
7         Проверяет работоспособность программы.
8     """
9     # Читаем синий канал оригинала
10    blue_original = cv2.imread("Images/Lenna.png", 0)
11    # Читаем синий канал модифицированного сообщения
12    blue_stego = cv2.imread("Images/LSB_Lenna.png", 0)
13    # Получим только последний бит.
14    # Для контрастности умножим полученное значение на 255,
15    # таким образом в матрицу будут лишь значения 0 и 255.
```

```

16     bw_original = (blue_original & 1) * 255
17     bw_stego = (blue_stego & 1) * 255
18     # Сохраним полученные изображения в градации серого.
19     cv2.imwrite("Images/BW_Lenna.png", bw_original)
20     cv2.imwrite("Images/BW LSB_Lenna.png", bw_stego)
21
22
23 if __name__ == "__main__":
24     main()

```

Полученные изображения можно увидеть на рисунке 4.1. Как видно, наименее значащий бит оригинального изображения распределен шумоподобно, в то время как у стегоконтейнера этот бит вносит в изображение какую-то структуру. Так происходит из-за неравномерного распределения символов в исходном сообщении. Также из изображения можно увидеть примерную длину сообщения.

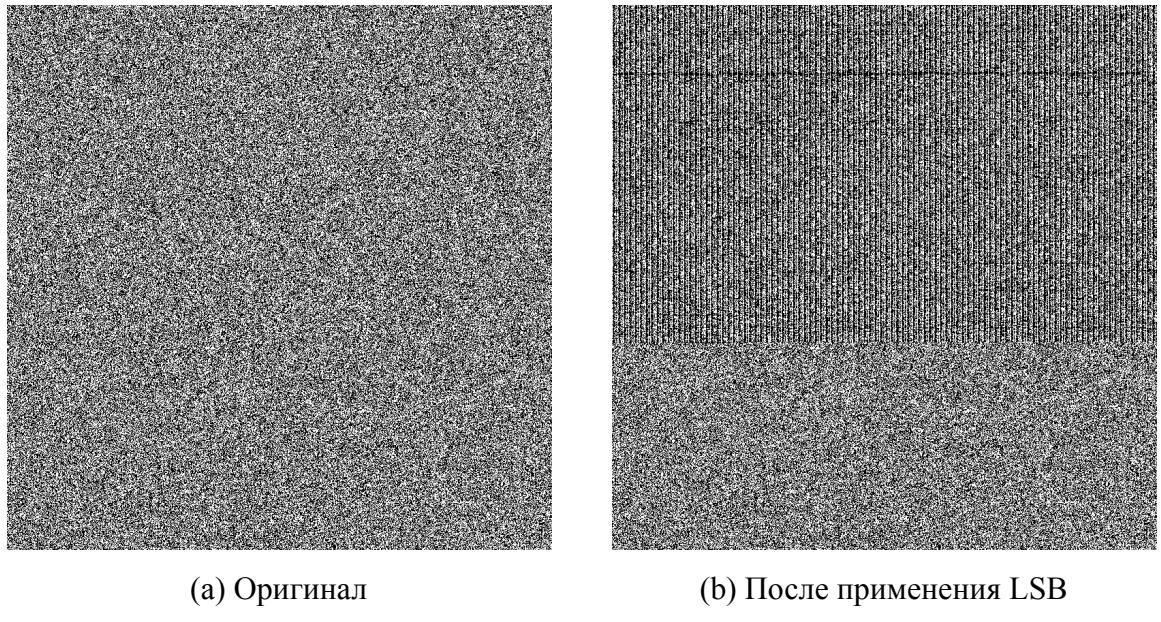


Рисунок 4.1 — Наименее значащий бит до и после LSB

### 4.3 Атака оценки числа переходов значений младших бит в соседних элементах контейнера

Допустим, что стегосообщение предварительно было зашифровано и скрыто методом LSB. В таком случае сообщение внутри контейнера будет обладать предельной энтропией, а это значит, что наименее значимый бит контейнера будет распределен равномерно: 50% объема будет зани-

мать 0 и другие 50% будет занимать 1. При этом два соседних наименее значимых бита не будут коррелированы. Однако в реальных изображениях это не так. В реальном изображении соседние пиксели коррелированы, и ненулевая вероятность того, что они окажутся одинаковыми. Несмотря на то, что изображение 4.1а выглядит как шум, в нем есть некоторая статистическая структура. Чтобы выявить ее, нужно попарно сравнить 2 соседних пикселя и посчитать, какой процент этих пикселей совпал. Такое сравнение проводит в листинге 4.2.

### Листинг 4.2 — Корреляция LSB

```
1 import numpy as np
2 import cv2
3
4
5 def pretty_stat(X: np.array, name: str) -> None:
6     """
7         Печатает функцию распределения данной дискретной выборки.
8     """
9     size = len(X)
10    # Посчитаем частоты элементов последовательности
11    comparison = dict(zip(*np.unique(X, return_counts=True)))
12    # Выведем статистику на экран
13    stat = f'{name}: '
14
15    for key, value in comparison.items():
16        stat += f"P({key}) = {value / size:.2}, "
17
18    if len(comparison) == 0:
19        print(stat + "empty")
20
21    else:
22        print(stat[:-2])
23
24
25 def main() -> None:
26     """
27         Считает корреляцию НЗБ у изображения,
28         заполненного шумоподобным сообщением,
29         и оригинального изображения.
30     """
31
32     # Считаем синий канал оригинала.
33     blue_original = cv2.imread("Images/Lenna.png", 0)
34     # Получим только последний бит.
35     bin_original = (blue_original & 1)
```

```

35     # Преобразуем матрицу бит в массив бит.
36     bin_original = bin_original.ravel() [:]
37     # Для сравнения генерируем псевдослучайную
38     # равномерно распределенную последовательность
39     # бит, чтобы смоделировать стегосообщение.
40     bin_stego = np.random.randint(2, size=len(bin_original))
41     # Сравним 2 соседних бита у оригинального изображения
42     # и промоделированного.
43     original_comparison = (bin_original[1:] == bin_original[:-1])
44     stego_comparison = (bin_stego[1:] == bin_stego[:-1])
45     # Посмотрим на получившееся распределение
46     pretty_stat(original_comparison, "Original")
47     pretty_stat(stego_comparison, "Stego")
48
49
50 if __name__ == "__main__":
51     main()

```

Вывод программы следующий: “Original:  $P(\text{False}) = 0.46$ ,  $P(\text{True}) = 0.54$ ; Stego:  $P(\text{False})=0.5$ ,  $P(\text{True})=0.5$ ”. Как видно, у оригинального сообщения совпадение и несовпадение двух соседних наименее значимых битов не равновероятны.

#### 4.4 Атака хи-квадрат

Допустим, что мы оказались в похожей ситуации: зашифрованное сообщение было скрыто методом LSB, — но в этот раз в контейнере соседние элементы не коррелированы. Построим гистограмму элементов контейнера и рассмотрим элементы, отличающиеся друг от друга в младшем бите. Если это незаполненный стегоконтейнер, то частота двух соседних элементов  $a$  и  $b$  может сильно отличаться. Однако если к такому контейнеру применить LSB, то в паре двух соседних значений  $a$  в 50% случаев не изменится, и в 50% случаев изменится на единицу. То же самое произойдет и с  $b$ . Допустим, что  $a$  был четным. Тогда в новом распределении частота  $a$  будет равна  $\frac{f(a) + f(b)}{2}$ , где  $f$  — функция распределения частот в незаполненном контейнере. То же самое касается и  $b$ . То есть соседние элементы будут распределены одинаково. На этом и основана атака хи-квадрат.

Критерий согласия Пирсона (критерий согласия  $\chi^2$ ) — это критерий принадлежности наблюдаемой выборки  $x_1, x_2, \dots, x_3$  теоретическому

закону распределения  $F(x, \theta)$ , где  $\theta$  — это известный параметр распределения. Процедура проверки гипотез с использованием критерия  $\chi^2$  предусматривает группирование наблюдений. Область определения случайной величины разбивают на  $k$  непересекающихся интервалов граничными точками  $x_{(0)}, x_{(1)}, \dots, x_{(k)}$ . В соответствии с заданным разбиением подсчитывают число  $n_i$  выборочных значений, попавших в  $i$ -й интервал, и вероятности попадания в интервал  $P_i(\theta) = F(x_{(i)}, \theta) - F(x_{(i-1)}, \theta)$  соответствующие теоретическому закону с функцией распределения  $F(x, \theta)$ . При этом  $n = \sum_{i=1}^k n_i$  и  $\sum_{i=1}^k P_i(\theta) = 1$ . В основе критерия согласия Пирсона лежит измерение отклонений  $\frac{n_i}{n}$  от  $P_i(\theta)$ . Статистика критерия согласия  $\chi^2$  Пирсона определяется соотношением 4.1.

$$\chi^2 = n \sum_{i=1}^k \frac{(n_i/n - P_i(\theta))^2}{P_i(\theta)} \quad (4.1)$$

Выделим из изображения все пары элементов, которые отличаются только в младшем бите. Обозначим такие пары через  $(2m, 2m + 1)$ . Пусть  $h_i$  обозначает гистограмму наблюдаемого распределения элементов. Введем новое наблюдаемое распределение  $\{o_m\}$  равное  $o_m = h_{2m}$  и теоретическое распределение  $\{e_m\}$  равное  $e_m = \frac{h_{2m} + h_{2m+1}}{2}$ . Разница между этими двумя распределениями измеряется критерием 4.2 с  $(\nu - 1)$  степенями свободы, где  $\nu$  — количество пар, отличающихся в младшем бите.

$$\chi^2 = \sum_{e_m \neq 0} \frac{(o_m - e_m)^2}{e_m} \quad (4.2)$$

Степень сходства двух распределений  $\{o_m\}$  и  $\{e_m\}$  после этого считается с помощью функции распределения по формуле 4.3.

$$p = 1 - \int_0^{\chi^2} \frac{t^{(\nu-2)/2} e^{-t/2}}{2^\nu \Gamma(\nu/2)} dt \quad (4.3)$$

Реализация атаки на python представлена в листинге 4.3.

#### Листинг 4.3 — Атака хи-квадрат

```

1 import jpegio as jio
2 import scipy.stats
3 import numpy as np
4
```

```

5
6 def chi_attack(file_name: str) -> None:
7     """
8     Реализует атаку хиквадрат- на JPEG файл.
9     """
10    # Считываем ДКП коэффициенты
11    dct = jio.read(file_name)
12    # Выбираем синий канал, в который спрятано сообщение.
13    # Здесь важно, что сообщение представляет собой шум.
14    container = dct.coef_arrays[1].ravel()[:]
15    # Строим гистограмму
16    unique, counts = np.unique(container, return_counts=True)
17    hist = dict(zip(unique, counts))
18    # Ищем соседние пары
19    unique.sort()
20    pairs = [(x, y) for (x, y) in zip(unique[:-1], unique[1:]) if x ^ y == 1]
21    # Строим наблюдаемое и ожидаемое распределения
22    observed = [hist[x] for (x, _) in pairs]
23    expected = [(hist[x] + hist[y]) / 2 for (x, y) in pairs]
24    # Считаем степень сходства
25    _, p = scipy.stats.chisquare(observed, f_exp=expected)
26    print(f'{p:.2f}')
27
28
29 def main() -> None:
30     """
31     Проверяет работоспособность программы.
32     """
33     # Пустой стегоконтейнер
34     chi_attack("Images/Lenna.jpg")
35     # Заполненный шумом стегоконтейнер
36     chi_attack("Images/JSteg_Lenna.jpg")
37
38
39 if __name__ == "__main__":
40     main()

```

Программа выводит 0.0 для пустого контейнера и 0.99 для заполненного.

## 4.5 Методы противодействия

Для эффективного противодействия описанным статистическим и субъективным атакам следует пользоваться следующими рекомендациями при построении стеганографического алгоритма:

- a) Сообщение должно встраиваться в зашифрованном виде для предотвращения субъективных и иных атак.
- б) Сообщение должно встраиваться не в подряд идущие элементы контейнера, а в случайно выбранные элементы, например, с использованием генератора псевдослучайных чисел.
- в) Сообщение должно заполнять лишь малую часть емкости контейнера. Иначе оно может нарушить статистическую структуру контейнера. Так, например, атака хи-квадрат работает намного хуже, когда сообщение рассеяно по всей длине контейнера.

## 5 Экономическая оценка проекта

### 5.1 Постановка задачи

Целью дипломного проекта является анализ стеганографических методов защиты информации. Данный раздел содержит расчет трудоемкости и затрат на проведение анализа предметной области, разработки и сопровождения стеганографической системы.

### 5.2 Оценка стоимости объектов интеллектуальной собственности

Оценка стоимости объектов интеллектуальной собственности (ОИС), созданных на предприятии или по его заказу (при финансировании разработок предприятием) с закреплением за ним по договору прав собственности на них, производится по затратному методу и определяется по формуле 5.1:

$$C_i = C_p + C_n + C_m \quad (5.1)$$

где,

- а)  $C_p$  — приведенные затраты на создание объектов интеллектуальной собственности, руб.;
- б)  $C_n$  — приведенные затраты на правовую охрану объектов интеллектуальной собственности, руб.;
- в)  $C_m$  — приведенные затраты на маркетинговые исследования ОИС, руб.

Приведенные затраты на создание ОИС — сумма фактически произведенных затрат на выполнение научно-исследовательской работы (НИР) в полном объеме (от поиска материалов исследования до формирования отчета) и разработку всей технической документации. Приведенные затраты для НИР состоят из затрат на поисковые работы, включая предварительную проработку проблемы, на теоретические исследования, на проведение экспериментов, испытаний, на услуги сторонних организаций, на составление, рассмотрение и утверждение отчета и прочих затрат.

Приведенные затраты на разработку технической документации (ТД) состоят из затрат на выполнение эскизного проекта, технического задания, рабочего проекта, расчетов, испытаний, услуг сторонних организа-

ций, авторского надзора, дизайна. Кроме того, сюда включаются затраты на доведение ОИС до готовности промышленного использования и коммерческой реализации.

В тех случаях, когда НИР или технологическая и проектная документация выполняется частично, или созданию ОИС предшествует проведение только НИР или разработка технической документации, то расчет стоимости ОИС производится по затратам на фактически выполненные работы, для товарных знаков и промышленных образцов — затратам на дизайн.

Приведенные затраты на правовую охрану ОИС — затраты на оформление заявочных материалов на получение патента (свидетельства), переписка по заявке, оплата пошлин за проведение экспертизы, получение патента (свидетельства) и поддержание его в силе и т.д. Данная составляющая отсутствует для таких ОИС, как ноу-хай, НИР, ТД.

Приведенные затраты на маркетинговые исследования — для целей приведения разновременных стоимостных оценок к конечному году применяется коэффициент  $\alpha_i$ , но в данном случае он будет равен 1, потому что число лет, предшествующих расчетному году равно 0. Затраты, произведенные на выполнение НИР и разработку всех стадий ТД, то есть Ср могут включать в себя:

- а) израсходованные материальные ресурсы;
- б) оплата труда с отчислениями разработчиков;
- в) амортизационные отчисления оборудования, которое использовалось при разработке ОИС;
- г) аренда помещения для разработчиков;

Кроме того, создание любого ОИС или разработка программного обеспечения происходит не всегда согласно производственному заданию, где четко оговорены сроки работы. Разработчик может вне плановых заданий выполнить(изготовить) ОИС или написать программу для ЭВМ. В случае если отсутствует документация по затратам на создание ОИС, то их можно определить расчетным путем.

Затраты на создание ОИС в  $t$ -том году определяются по следующей формуле:

$$C_{\text{пр}} = \frac{3}{m} Kt \quad (5.2)$$

Где,

- а) 3 — среднемесячная заработка разработчика (разработчиков) с учетом районного коэффициента, руб.;
- б)  $m$  — среднее количество рабочих часов в месяце;
- в)  $K$  — коэффициент, учитывающий отчисления с заработной платы (страховые взносы). Ставка страховых взносов в 2021 г. составляет 30% от величины фонда оплаты труда.  $K = 0,3$ ;
- г)  $t$  — время в часах, затрачиваемое разработчиком (разработчиками) на создание (разработку) ОИС, на отладку и адаптацию ОИС к условиям производства.

Для расчета себестоимости необходимы затраты времени. Весь перечень произведённых работ не был оговорен рамками технического задания с указанием конкретных сроков выполнения. Для обеспечения наибольшей достоверности временных затрат используем метод экспертных оценок. Время работы, согласно плану проведения аудита ИБ, учтём в соответствующих этапах.

Для определения средних значений  $a_{i \text{ср}}$ ,  $m_{i \text{ср}}$ ,  $b_{i \text{ср}}$  используются экспертные оценки, данные руководителем и автором проекта. Средние значения найдем по формуле 5.4. Значения  $m_i$  и  $b_i$  рассчитываются аналогично.

$$a_{i \text{ср}} = \frac{3a_{i \text{рук}} + 2a_{i \text{авт}}}{5} \quad (5.3)$$

где,

- а)  $a_{i \text{рук}}$  — оценка, данная руководителем;
- б)  $a_{i \text{авт}}$  — оценка, данная автором.

Таблица 5.1 — Затраты времени на разработку ОИС

Этапы	Величина затрат		
	Минимальная	Вероятная	Максимальная

	Руководитель	Автор	Средняя	Руководитель	Автор	Средняя	Руководитель	Автор	Средняя
Ознакомление с исходными данными	3	4	3.5	4	6	5	5	7	6
Анализ предметной области	9	17	13	13	25	18	25	41	33
Разработка системы	179	210	190	250	310	280	330	410	370
Вывод в эксплуатацию	9	17	13	17	31	24	25	41	33

Ожидаемая величина затрат для  $i$ -го этапа ( $MO_i$ ) и стандартное отклонение этой величины каждого  $i$ -го этапа ( $G_i$ ):

$$MO_i = \frac{\alpha_i + 4m_i + b_i}{6} \quad (5.4)$$

$$G_i = \frac{b_i - a_i}{6} \quad (5.5)$$

Результаты показаны в таблице 5.2

Таблица 5.2 — Затраты времени на разработку ОИС

Этапы разработки ОИС	Средняя величина затрат времени этапа разработки ОИС			$MO_i$	$G_i$
	$a_i$	$m_i$	$b_i$		
Ознакомление с исходными данными	3.5	5	6	4.9	0.4
Анализ предметной области	13	18	33	19.6	3.3
Разработка системы	190	280	370	280	30
Вывод системы в опытную эксплуатацию	13	24	33	23.6	3.3

Зная ожидаемые затраты и стандартное отклонение по каждому этапу, рассчитываются эти показатели в целом по ОИС:

$$MO = \sum_{i=1}^n MO_i \quad (5.6)$$

$$G = \sqrt{\sum_{i=1}^n G_i^2} \quad (5.7)$$

$$MO = 4.9 + 19.6 + 280 + 23.6 = 328.1$$

$$G = \sqrt{0.4^2 + 3.3^2 + 30^2 + 3.3^2} = 30.4$$

### 5.3 Оценка стоимости разработки

Необходимо определить себестоимость разработки системы. Стоимость разработки ОИС найдём по формуле 5.2 при следующих данных:

- а) среднемесячная заработка разработчика с учетом коэффициента составляет 90 000 руб.;
- б) среднее количество рабочих часов в месяце — 168;
- в) затраты времени на разработку ОИС:  $328.1 + 30 = 358.1$  часа.

$$C = \frac{90000}{168} * 358.1 = 191839.3$$

Прочие расходы:

- а) Ставка страховых взносов в 2021 г. составляет 30% от величины фонда оплаты труда. В нашем примере они составят: Страховые взносы =  $C * 0.3 = 57551.7$

Общие затраты на разработку составят:

$$Z = Z_{\text{прям}} + Z_{\text{пр}} = 191839.3 + 57551.1 = 249390.4$$

### 5.4 Оценка стоимости использования оборудования и сопровождения системы

Для развертывания системы будет использоваться выделенный сервер. Траты для сервера с конфигурацией: четырехядерный процессор с

восьмью потоками, 16 гигабайт оперативной памяти, 1 терабайт SSD — 6 000 рублей/месяц. Сопровождением системы займется системный администратор. По формуле выше найдем затраты на сотрудника при учете месячного оклада 40 000 рублей:

$$C = 40000 + 40000 * 0.3 = 52000 \text{ руб.}$$

Суммарные среднемесячные затраты на систему составляют 58000 рублей.

## 5.5 Экономическое обоснование проекта

Стоимость разработки проекта составляет 249 390.4 рублей. Анализ показал, что данная система позволит сократить убытки компании от несанкционированного копирования и распространения цифровых объектов в 2 раза. В среднем компания теряет 240 000 рублей в месяц из-за несанкционированного копирования и распространения цифровых объектов компании. Рассчитаем, сколько проект приносит прибыли в месяц:

$$P = \frac{240000}{2} - 58000 = 62000$$

где P — это прибыль. Итого срок окупаемости проекта составляет пять месяцев.

## ЗАКЛЮЧЕНИЕ

После проделанной работы можно подвести следующие итоги:

- а) были описаны общие сведения о стеганографических методах сокрытия информации;
- б) был описан алгоритм сокрытия информации методом наименее значимого бита;
- в) была написана реализация алгоритма сокрытия информации методом наименее значимого бита с использования языка программирования Python;
- г) был проведен анализ разобранного алгоритма, найдены и продемонстрированы его уязвимости;
- д) были описаны и проанализированы алгоритмы сокрытия в спектральной области;
- е) был проведен анализ алгоритмов сокрытия в спектральной области, найдены и продемонстрированы их уязвимости;
- ж) была написана реализация алгоритмов сокрытия в спектральной области на языке Python;
- и) были реализованы атаки на описанные алгоритмы;
- к) были приведены рекомендации по защите от реализованных атак.

Из высказанного можно сделать вывод о том, что мной успешно были рассмотрены основные стеганографические алгоритмы, определены области их применения, разобраны их недостатки и достоинства. Я научился проводить анализ стеганографических алгоритмов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конахович, Г. Ф. Компьютерная стеганография. Теория и практика / Г. Ф. Конахович, А. Ю. Пузыренко. — МК-Пресс, 2006.
2. Грибунин, В. Г. Цифровая стеганография / В. Г. Грибунин, И. В. Туринцев, И. Н. Оков. — Солон-пресс, 2020.
3. Национальная электронная библиотека им. Н.Э. Баумана. Стеганография. — <https://ru.bmstu.wiki/Стеганография>. — 2021.
4. Завьялов, С. В. Стеганографические методы защиты информации / С. В. Завьялов, Ю. В. Ветров. — Изд-во Политехн. ун-та, 2012.
5. Лучано, Р. Python. К вершинам мастерства / Р. Лучано. — ДМК Пресс, 2016.
6. Маккинни, У. Python и анализ данных / У. Маккинни. — ДМК Пресс, 2015.
7. Kaggle. Steganalysis : Complete Understanding and Model. — <https://www.kaggle.com/tanulsingh077/steganalysis-complete-understanding-and-model>. — 2021.
8. Lutz, M. Learning Python / M. Lutz. — O'Reilly, 2013.
9. Mahajan, M. Secure Steganography in Colored Images: A Novel Approach to Information Security / M. Mahajan, S. Bhushan. — LAP LAMBERT Academic Publishing, 2013.
10. Sattar, I. A. Steganography In Spatial Domain / I. A. Sattar. — LAP LAMBERT Academic Publishing, 2018.

# ПРИЛОЖЕНИЕ А

## РЕАЛИЗАЦИЯ МЕТОДА БЕНГАМА-МЕМОНА-ЭО-ЮНГА НА

### PYTHON

```
1 import numpy as np
2 import jpegio as jio
3 import random
4
5
6 class BMYY():
7     """
8     Реализация метода БенгамаМемонаЭоЮнга---.
9     """
10
11    def __init__(self, file_name: str, message: bytes = None,
12                 seed: int = 0) -> None:
13        """
14        Принимает на вход путь до файла с изображением file_path,
15        байтовое сообщение message и попрождающий элемент seed,
16        используемый для инициализации ГПСЧ. Возвращает простой
17        в использовании JPEG кодер.
18        """
19        self._file_name = file_name
20        # Считываем ДКП коэффициенты
21        self._dct = jio.read(self._file_name)
22        # Считываем канал яркости.
23        self._container = self._dct.coef_arrays[0]
24
25        if message is None:
26            self.message = []
27
28        else:
29            self.message = message
30
31        # Сохраняем seed
32        self.seed = seed
33        # Коэффициенты для встраивания
34        self._stego_coef = [(i, j) for i in range(8) for j in range(8)
35                            if i + j < 5 and (i, j) != (0, 0)]
36
37        # Высокочастотные коэффициенты
38        self._high_coef = np.array(
39            [i + j > 9 for i in range(8) for j in range(8)])
40        .reshape(8, 8)
41
42        # Низкочастотные коэффициенты
```

```

43     self._low_coef = np.array(
44         [i + j < 5 for i in range(8) for j in range(8)])
45     ).reshape(8, 8)
46
47     # Сохраняем порог различия
48     self._P = 3
49     # Сохраняем порог яркости
50     self._Pl = 210
51     # Сохраняем порог монотонности
52     self._Ph = 40
53
54     def _is_suitable_block(self, block: np.array) -> bool:
55         # Проверяем блок на порог яркости и монотонности
56         l = np.absolute(block[self._low_coef]).sum()
57         h = np.absolute(block[self._high_coef]).sum()
58         # return True
59         return l >= self._Pl and h <= self._Ph
60
61     def _encode_block(self, block: np.array, bit: bool) -> np.array:
62         """
63             В данном блоке block кодирует bit за счет
64             изменения соотношения между тремя псевдослучайными
65             элементами.
66         """
67
68         # С помощью ГПСЧ выбираем случайные элементы блока
69         k1, k2, k3 = random.sample(self._stego_coef, 3)
70
71         # Кодируем ноль, устанавливая block[k3] минимальным
72         # из трех элементов так, чтобы это соотношение
73         # сохранилось после квантования коэффициентов
74         if bit == False:
75             m = min(block[k1], block[k2])
76             block[k3] = m - self._P / 2
77
78             if block[k1] == m:
79                 block[k1] += self._P / 2
80
81         else:
82             block[k2] += self._P / 2
83
84         # Кодируем единицу, устанавливая block[k3] максимальным
85         # из трех элементов так, чтобы это соотношение
86         # сохранилось после квантования коэффициентов
87         else:
88             m = max(block[k1], block[k2])
89             block[k3] = m + self._P / 2

```

```

89             if block[k1] == m:
90                 block[k1] -= self._P / 2
91
92             else:
93                 block[k2] -= self._P / 2
94
95
96         return block
97
98     def _decode_block(self, block: np.array) -> bool:
99         """
100         Для данного блока block декодирует
101         bit, закодированный с помощью соотношения
102         между тремя псевдослучайными элементами
103         """
104
105         # С помощью ГПСЧ выбираем случайные элементы блока
106         k1, k2, k3 = random.sample(self._stego_coef, 3)
107
108         # Находим максимум разницы между третьим
109         # и двумя остальными элементами
110         M = max(block[k1], block[k2], block[k3])
111
112         if block[k3] == M:
113             return True
114
115         else:
116             return False
117
118     def _blockshaped(self, arr: np.array, nrows: int, ncols: int) ->
119     np.array:
120         """
121         Возвращает массив формы (n, nrows, ncols), где
122         n * nrows * ncols = arr.size
123
124         Если массив – это матрица, тогда возвращает массив,
125         выглядящий как разбиение этой матрицы на подматрицы.
126         """
127
128         h, w = arr.shape
129         return (arr.reshape(h//nrows, nrows, -1, ncols)
130                 .swapaxes(1, 2)
131                 .reshape(-1, nrows, ncols))
132
133     def _unblockshaped(self, arr: np.array, h: int, w: int) -> np.array:
134         """
135         Возвращает матрицу формы (h, w), где
136         h * w = arr.size

```

```

134     Если матрица формы  $(n, \text{nrows}, \text{ncols})$ , где  $n$  – это подматрицы
135     формы  $(\text{nrows}, \text{ncols})$ , тогда возвращает матрицу, составленную
136     из этих подматриц.
137     """
138     n, nrows, ncols = arr.shape
139     return (arr.reshape(h//nrows, -1, nrows, ncols)
140             .swapaxes(1, 2)
141             .reshape(h, w))
142
143     def encode(self) -> bytes:
144         """
145         Кодирует сообщение в контейнер и возвращает позиции подходящих блоков.
146         """
147         # Инициализируем ГПСЧ
148         random.seed(self.seed)
149         # Разбиваем массив ДКП коэффициентов на блоки
150         blocks = self._blockshaped(self._container, 8, 8)
151         # Находим положение подходящих блоков
152         mask = np.array([self._is_suitable_block(block) for block in
153                         blocks])
154         # В подходящих блоках меняем соотношения коэффициентов
155         suitable_blocks = blocks[mask]
156         # Преобразуем сообщение в бинарный вид
157         np_message = np.unpackbits(np.frombuffer(
158             self.message, dtype=np.uint8)).ravel()
159         # Находим длину сообщения
160         n = len(np_message)
161         # Кодируем сообщение
162         encoded_blocks = []
163
164         for i in range(n):
165             encoded_blocks.append(self._encode_block(
166                 suitable_blocks[i], np_message[i]))
167
168         # Перезаписываем подходящие блоки
169         suitable_blocks[:n] = np.array(encoded_blocks)
170         blocks[mask] = suitable_blocks
171         # Соединяем блоки обратно в контейнер
172         self._container = self._unblockshaped(blocks,
173                                             *self._container.shape)
174         # Сохраняем информацию в изображение
175         self._dct.coef_arrays[0].ravel()[:] = self._container.ravel()
176         # Возвращаем позиции встраивания
177         return np.packbits(mask).tobytes()

    def decode(self, positions: bytes) -> bytes:

```

```

178     """
179     Декодирует сообщение из контейнера
180     """
181     # Инициализируем ГПСЧ
182     random.seed(self.seed)
183     # Разбиваем массив ДКП коэффициентов на блоки
184     blocks = self._blockshaped(self._container, 8, 8)
185     message = []
186     # Находим позиции подходящих блоков
187     mask = np.unpackbits(np.frombuffer(positions,
188                          np.uint8)).astype(bool)
189     # Находим подходящие блоки
190     suitable_blocks = blocks[mask[:len(blocks)]]]
191
192     # Декодируем сообщение
193     for block in suitable_blocks:
194         # Декодируем бит
195         bit = self._decode_block(block)
196         message.append(bit)
197
198     # Из бит собираем исходное сообщение
199     message = np.packbits(message)
200     # Преобразуем его в байты
201     return message.tobytes()
202
203     def save(self) -> None:
204         """
205         Перезаписывает исходный файл
206         новый контейнером.
207         """
208         jio.write(self._dct, self._file_name)
209
210     def save_as(self, file_name: str) -> None:
211         """
212         Сохраняет контейнер в файл,
213         заданный параметром file_name.
214         """
215
216
217     def main() -> None:
218         """
219         Проверяет работоспособность алгоритма.
220         """
221         # Переводим сообщение в байтовую форму
222         message = ("Hello, stegoworld!").encode()

```

```
223 # Запоминаем размер бинарного сообщения
224 size = len(message)
225 # Создаем кодер
226 jpg = BMYY("Images/Lenna.jpg", message)
227 # Скрываем сообщение, запоминаем позиции
228 # блоков, в которые встроены биты
229 positions = jpg.encode()
230 jpg.save_as("Images/BMYY_Lenna.jpg")
231 jpg = BMYY("Images/BMYY_Lenna.jpg")
232 decoded = jpg.decode(positions)[:size]
233 # Убеждаемся, что метод работает
234 print(decoded)
235
236
237 if __name__ == "__main__":
238     main()
```