

СОДЕРЖАНИЕ

Обозначения и сокращения	7
Введение	8
1 Общие положения и описание стеганографических методов защиты информации	9
1.1 Основные понятия	9
1.2 Цифровые водяные знаки и цифровые отпечатки	10
1.3 Обобщенные стеганографические методы	12
1.4 Стегоанализ	12
2 Скрытия методом наименее значимого бита	14
2.1 Общие сведения	14
2.2 Алгоритм	15
2.3 Формат файла PNG	17
2.4 Реализация LSB для контейнера PNG	20
3 Скрытие в спектральной области	24
3.1 Общие сведения	24
3.2 Дискретное косинусное преобразование	24
3.3 JPEG	25
3.4 JSteg	31
3.5 Метод относительной замены величин коэффициентов ДКП ..	35
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	43
5.1 Постановка задачи	43
5.2 Оценка стоимости объектов интеллектуальной собственности ..	43
5.3 Оценка стоимости разработки	47
5.4 Оценка стоимости использования оборудования и сопровождения системы	47

5.5 Экономическое обоснование проекта	48
Заключение	49
Список использованных источников	50
Приложение А Реализация метода Бенгмана-Мемона-Эо-Юнга на Python	51

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 надежность понимается устойчивость к различным деформациям контейнера. По отношению к этой характеристики 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. Так же 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 скрытия 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. Так же сообщение разрушается в результате сжатия с потерями. Все это говорит о том, что метод обладает низкой робастостью.
- в) Факт сокрытия изображения легко обнаруживает методами стеганализа.

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

```

34     self._from_elements(elements)
35
36     def decode(self):
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
56     @abstractmethod
57     def _to_elements(self):
58         """
59             Преобразует контейнер в последовательность элементов
60         """
61
62         pass
63
64     @abstractmethod
65     def _from_elements(self, elements):
66         """
67             Собирает контейнер из элементов
68         """
69
70         pass

```

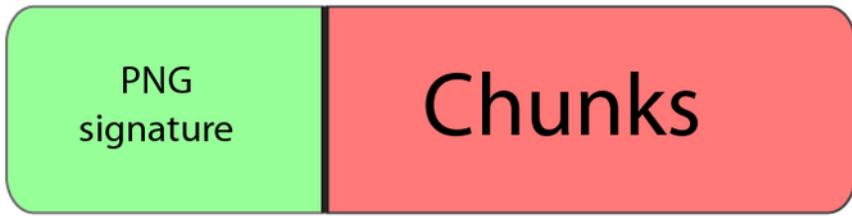
Как видно, в методах нет циклов **for**. Они скрыты за интерфейсом библиотеки `pintpy`. Интерфейс библиотеки `pintpy` позволяет нам применять операции к матрицам, из-за чего код выглядит лаконичнее. К тому же библиотека написана на языке C, поэтому ее код работает очень быстро.

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 передает 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 библиотека 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:

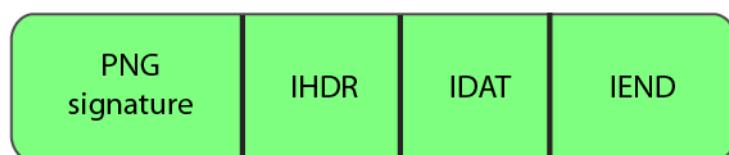
- а) IHDR — заголовочный блок, содержащий основную информацию об изображении.
- б) PLTE — палитра изображения.
- в) IDAT — содержит непосредственно изображение. В любом PNG файле должно быть не менее одного такого блока.
- г) IEND — завершающих чанк. Должен находиться в самом конце файла.

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

```

43     заданный параметром file_name.
44     """
45     image = Image.fromarray(self.container)
46     image.save(file_name)
47
48
49 def main():
50     # Считываем сообщение.
51     with open(”Messages/Alice in wonderland.txt”, ”rb”) as f:
52         message = f.read()
53
54     # Запоминаем длину сообщения.
55     size = len(message)
56     # Кодируем сообщение.
57     png = PNG(”Images/Lenna.png”, message)
58     png.encode()
59     png.save_as(”Images/LSB_Lenna.png”)
60     # Переоткрываем изображение.
61     png = PNG(”Images/LSB_Lenna.png”)
62     # Декодируем сообщение.
63     decoded = png.decode()
64
65     # Проверяем, что сообщения до и после совпадают.
66     new_message = decoded[:size]
67     print(f”{new_message == message}”)
68
69
70 if __name__ == ”__main__”:
71     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 Общие сведения

3.2 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 любое дискретное линейное преобразование, ДКП можно представить в виде матрицы. Будучи ортоганальным преобразованием, обратное к ДКП преобразование задает 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 использующее только действительные числа.

Существует 8 стандартных типов ДКП, однако наиболее употребимым 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 формуле 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 информации, 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.

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

Именно такое преобразование используется при компрессии в JPEG.

3.3 JPEG

JPEG является широко используемым методом сжатия с потерями для цифровых изображений. Степень сжатия регулируется, что позволяет выбирать между качеством и размером изображения. JPEG наиболее широко используемый стандарт сжатия изображений в мире и наиболее используемый формат 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 сделана визуализация преобразования ДКП.

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



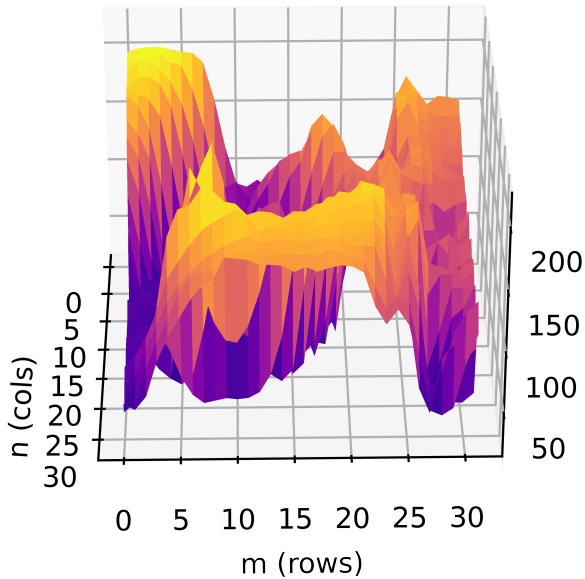
Рисунок 3.1 — Выбираем область

Сначала рассмотрим матрицу пикселей как двумерную дискретную функцию. Расположим координаты так, чтобы в левом верхнем углу располагался 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.

Рисунок 3.2 — Визуализация пикселей

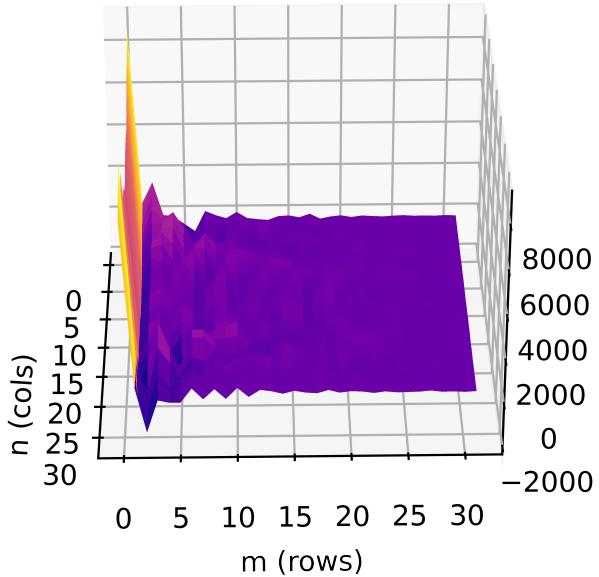


Остальные коэффициенты называются АС-коэффициентами. Матрица показана на рисунке 3.4.

DC-коэффициент блока равен среднему всех пикселей в блоке, взято-му с определенным коэффициентом. Удаляя все коэффициенты, кроме DC, мы можем аппроксимировать блок пикселей их средним арифметическим. Чем дальше коэффициент располагается от DC, тем меньше психовизуаль-ной 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.

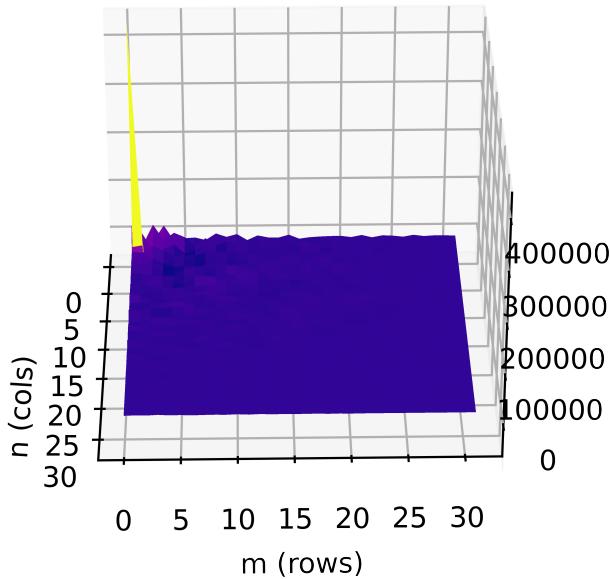
Алгоритм сжатия JPEG работает с каждым каналом отдельно, поэто-му для простоты рассмотрим работу JPEG на изображении в режиме града-ции серого. В самом начале своей работы алгоритм разбивает изображение на блоки 8x8 пикселей. К каждому блоку применяется ДКП преобразова-ние, что равносильно разложению исходной матрицы по базису, состояще-му из 64 функций. Эти 64 функции показаны на рисунке 3.5. Из этого

Рисунок 3.3 — После применения ДКП к строкам матрицы



рисунка видно, что помере отдаления от левого верхнего которая функции становятся все более рельефными, что объясняет, почему они несут наиболее мелкие детали изображения. Так же видно, что функция, соответствующая DC-коэффициенту, представлена плоскостью. Очевидно, что лучшим константным приближением функции является ее математическое ожидание. После применения ДКП преобразования к блоку матрице 8×8 получается другая матрица той же размерности. В соответствии с вышесказанным 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 мат-

Рисунок 3.4 — После применения ДКП к столбцам матрицы

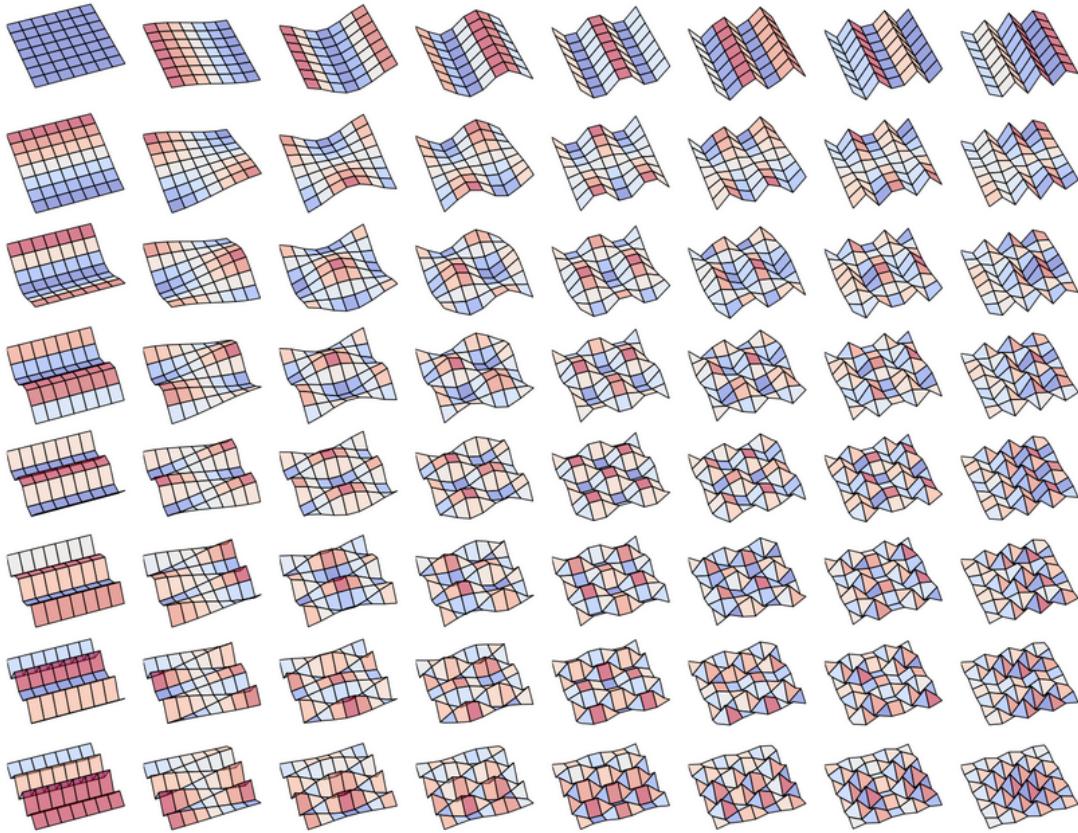


риц, одна из таких матриц представлена в формуле 3.5.

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

Это матрица для 50% качества, указанная в исходном стандарте JPEG. Каждый элемент ДКП матрицы делится на коэффициент 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.

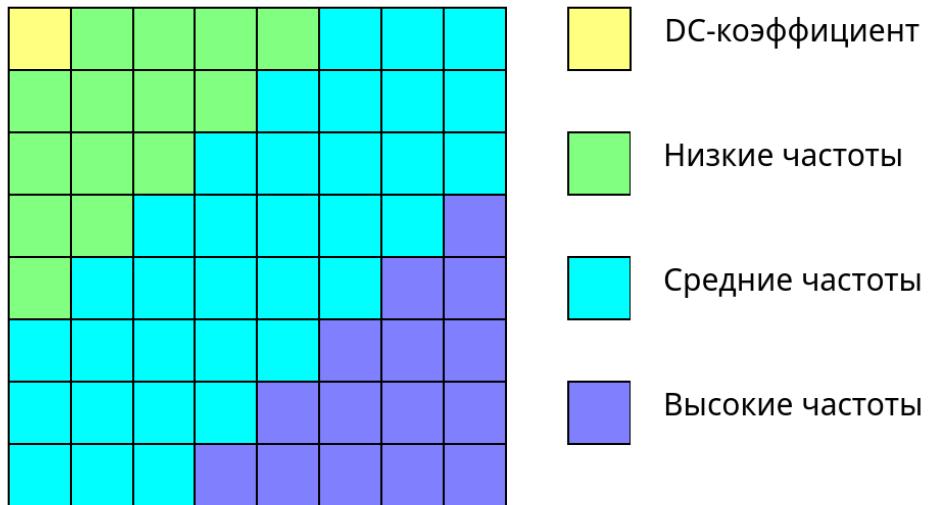
Рисунок 3.5 — Базис ДКП



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



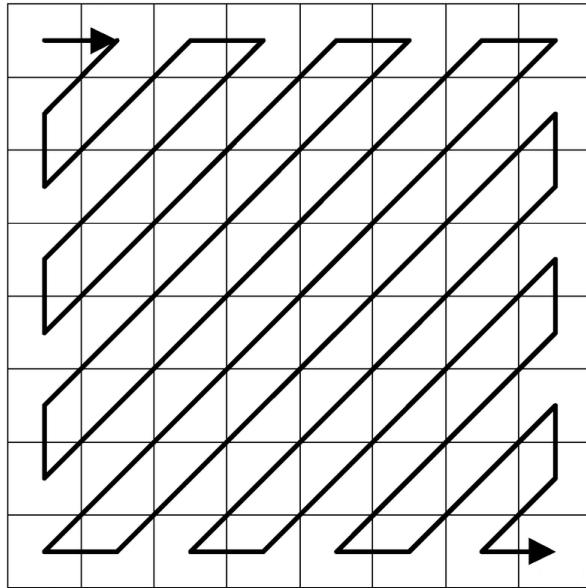
вов и количество его повторов. После этого к DC и AC коэффициентам применяется энтропийное кодирование с помощью алгоритма Хаффмана.

Мы разобрали работу алгоритма для одноканального изображения. В RGB изображениях такой алгоритм применяется к каждому каналу отдельно. Так же 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, C_B и C_R отвечают за синюю и красную компоненты. После преобразования в $Y\text{C}_\text{B}\text{C}_\text{R}$ над C_B и C_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

3.4 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 коэффициенты

Рисунок 3.7 — Зигзагообразный порядок



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:

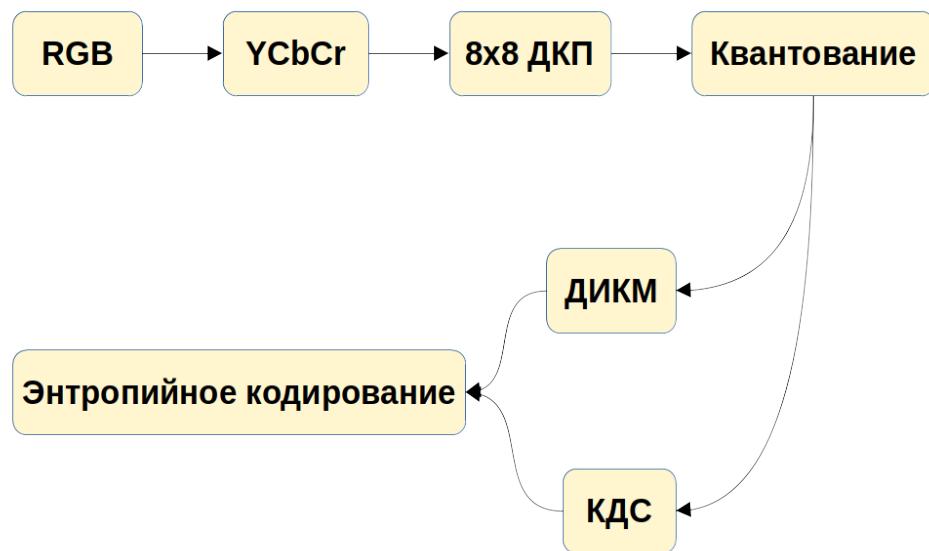
- а) Низкая вычислительная сложность.
- б) Алгоритм обеспечивает большую вместимость стегосообщений: стегосообщение может занимать до 13% объема контейнера.
- в) Изменения, вносимые в контейнер, незаметны для человеческого глаза.

Но у метода так же 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 соображения становится ясно, что метод не устойчив к сжатию. Это так же 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
5 from lsb import LSB
6
7
8 class JSteg(LSB):
9     def __init__(self, file_name: str, message: str = None):
10         """
11             Возвращает простой JSteg кодер,
12             принимает на вход имя файла и сообщение.
13         """
14         self.file_name = file_name
15         # Считываем все коэффициенты ДКП
```

```

16     self.dct = jio.read(self.file_name)
17     # Оставляем только Cb канал.
18     # Коэффициенты упорядочены в зигзагообразном порядке
19     container = self.dct.coef_arrays[1]
20     super().__init__(container, message)
21
22     def _to_elements(self):
23         """
24             Возвращает репрезентацию контейнера
25             как последовательности элементов.
26         """
27         return self.container.ravel()[:]
28
29     def _from_elements(self, elements):
30         """
31             Строит контейнер по последовательности
32             элементов.
33         """
34         self.dct.coef_arrays[1].ravel()[:] = elements
35
36     def save(self):
37         """
38             Перезаписывает исходный файл
39             новым контейнером.
40         """
41         jio.write(self.dct, self.file_name)
42
43     def save_as(self, file_name):
44         """
45             Сохраняет контейнер в файл,
46             заданный параметром file_name.
47         """
48         jio.write(self.dct, file_name)
49
50
51     def main():
52         # Считываем сообщение.
53         with open("Messages/Alice in wonderland.txt", "rb") as f:
54             message = f.read()[:80000]
55
56         # Запоминаем длину сообщения.
57         size = len(message)
58         # Кодируем сообщение.
59         jsteg = JSteg("Images/Lenna.jpg", message)
60         jsteg.encode()
61         jsteg.save_as("Images/JSteg_Lenna.jpg")

```

```

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

```

3.5 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 так же есть и серьезные недостатки:

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

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

- 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.

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;

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 модуляция 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. Декодирование происходит симметричным образом. 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 # Читаем синий канал оригинала
5 blue_original = cv2.imread("Images/Lenna.png", 0)
6 # Читаем синий канал модифицированного сообщения
7 blue_stego = cv2.imread("Images/LSB_Lenna.png", 0)
8
9 # Получим только последний бит.
10 # Для контрастности умножим полученное значение на 255,
11 # таким образом в матрицу будут лишь значения 0 и 255.
12 bw_original = (blue_original & 1) * 255
13 bw_stego = (blue_stego & 1) * 255
14
15 # Сохраним полученные изображения в градации серого.
```

```
16 | cv2.imwrite("Images/BW_Lenna.png", bw_original)
17 | cv2.imwrite("Images/BW_LSB_Lenna.png", bw_stego)
```

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. Так же из изображения можно увидеть примерную длину сообщения.

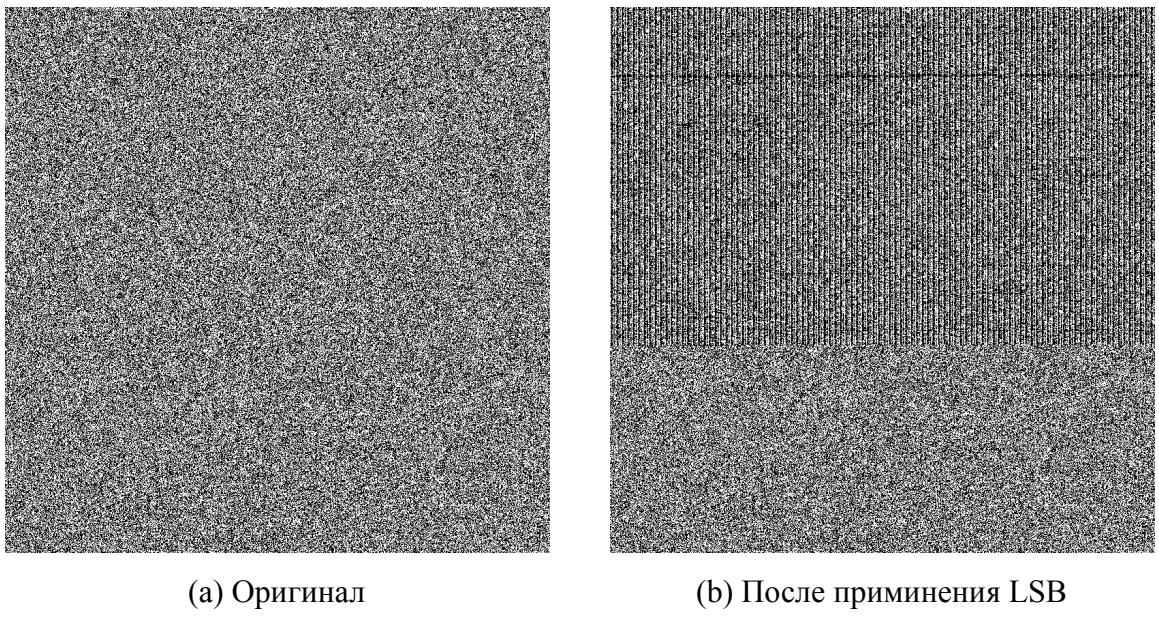


Рисунок 4.1 — Наименее значащий бит до и после LSB

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 занимать 0 и другие 50% будет занимать 1. При этом два соседних наименее значимых бита не будут скорелированы. Однако в реальных изображениях это не так. В реальном изображении соседние пиксели скореллированы, 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 # Считаем синий канал оригинала.
5 blue_original = cv2.imread("Images/Lenna.png", 0)
6
7 # Получим только последний бит.
8 bin_original = (blue_original & 1)
9
10 # Преобразуем матрицу бит в массив бит.
11 bin_original = bin_original.ravel()[:]
12
13 # Для сравнения сгенерируем псевдослучайную
14 # равномерно распределенную последовательность
15 # бит, чтобы смоделировать стегосообщение.
16 bin_stego = np.random.randint(2, size=len(bin_original))
17
18 # Сравним 2 соседних бита у оригинального изображения
19 # и промоделированного.
20 original_comparison = (bin_original[1:] == bin_original[:-1])
21 stego_comparison = (bin_stego[1:] == bin_stego[:-1])
22
23 # Напишем функцию, печатающую статистику по нашему распределению
24 def pretty_stat(X, name):
25     size = len(X)
26     # Посчитаем частоты элементов последовательности
27     comparison = dict(zip(*np.unique(X, return_counts=True)))
28     # Выведем статистику на экран
29     stat = f"{name}:\n"
30     for key, value in comparison.items():
31         stat += f"P({key}) = {value / size:.2f}, "
32
33     print(stat[:-2])
34
35 # Посмотрим на получившееся распределение
36 pretty_stat(original_comparison, "Original")
37 pretty_stat(stego_comparison, "Stego")
```

Вывод программы следующий: “Original: P(False) = 0.46, P(True) = 0.54; Stego: P(False)=0.5, P(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 скорелированы. 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 χ^2) — это критерий принадлежности наблюдаемой выборки x_1, x_2, \dots, x_n теоретическому закону распределения $F(x, \theta)$, где θ — это известный параметр распределения. Процедура проверки гипотез с использованием критерия χ^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 χ^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 ν — 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 from enum import unique
2 import jpegio as jio
3 import scipy.stats
4 import numpy as np
5
6 def chi_attack(file_name: str):
7     # Считываем ДКП коэффициенты
8     dct = jio.read(file_name)
9     # Выбираем синий канал, в который спрятано сообщение.
10    # Здесь важно, что сообщение представляет собой шум.
11    container = dct.coef_arrays[1].ravel()[:]
12    # Строим гистограмму
13    unique, counts = np.unique(container, return_counts=True)
14    hist = dict(zip(unique, counts))
15    # Ищем соседние пары
16    unique.sort()
17    pairs = [(x, y) for (x, y) in zip(unique[:-1], unique[1:]) if x ^ y == 1]
18    # Строим наблюдаемое и ожидаемое распределения
19    observed = [hist[x] for (x, _) in pairs]
20    expected = [(hist[x] + hist[y]) / 2 for (x, y) in pairs]
21    # Считаем степень сходства
22    _, p = scipy.stats.chisquare(observed, f_exp=expected)

```

```
23     print(f" {p:.2} ")  
24  
25  
26 if __name__ == "__main__":  
27     # Пустой стегоконтейнер  
28     chi_attack("Images/Lenna.jpg")  
29     # Заполненный шумом стегоконтейнер  
30     chi_attack("Images/JSteg_Lenna.jpg")
```

Программа выводит 0.0 для пустого контейнера и 0.99 для заполненного.

4.5 Методы противодействия

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

- а) 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 трудоемкость 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 — привиденные затраты на правовую охрану объектов интеллектуальной собственности, руб.;
- в) C_m — привиденные 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 α_i , но 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 производственного задания, 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 = $9000 * 0.3 = 27000$

Общие затраты на разработку составят:

$$Z = Z_{\text{прям}} + Z_{\text{пр}} = 191839.3 + 27000 = 218839.3$$

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 218 839.3 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:

$$P = \frac{240000}{2} - 52000 = 68000$$

где P — это прибыль. Итого срок окупаемости проекта составляет три месяца.

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;

Из вышесказанного 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. Тцирнцев, И. Н. Оков. — Солон-пресс, 2020.
3. *Национальная электронная библиотека им. Н.Э. Баумана. Стеганография.* — <https://ru.bmstu.wiki/Стеганография>. — 2021.

ПРИЛОЖЕНИЕ А

РЕАЛИЗАЦИЯ МЕТОДА БЕНГМАНА-МЕМОНА-ЭО-ЮНГА НА

PYTHON

```
1 from hashlib import new
2 from os import nice
3 from typing import Container
4 import cv2
5 import numpy as np
6 import jpegio as jio
7 import cv2
8 import itertools
9 import random
10
11 from numpy.core.shape_base import block
12
13
14 class DCT():
15     def __init__(self, file_name, message=None, seed=0):
16         """
17             Принимает на вход путь до файла с изображением file_path,
18             байтовое сообщение message и попрождающий элемент seed,
19             используемый для инициализации ГПСЧ. Возвращает простой
20             в использовании JPEG кодер.
21         """
22         self.file_name = file_name
23         # Считываем ДКП коэффициенты
24         self.dct = jio.read(self.file_name)
25         # Считываем канал яркости.
26         self.container = self.dct.coef_arrays[0]
27
28         if message is None:
29             self.message = []
30         else:
31             self.message = message
32
33         # Сохраняем seed
34         self.seed = seed
35         # Коэффициенты для встраивания
36         self.stego_coef = [(i, j) for i in range(8) for j in range(8)
37                            if i + j < 5 and (i, j) != (0, 0)]
38
39         # Высокочастотные коэффициенты
40         self.high_coef = np.array(
41             [i + j > 9 for i in range(8) for j in range(8)])
42         .reshape(8, 8)
```

```

43
44     # Низкочастотные коэффициенты
45     self.low_coef = np.array(
46         [i + j < 5 for i in range(8) for j in range(8)])
47     ).reshape(8, 8)
48
49     # Сохраняем порог различия
50     self.P = 3
51     # Сохраняем порог яркости
52     self.Pl = 210
53     # Сохраняем порог монотонности
54     self.Ph = 40
55     self.x = 0
56
57     def _check_block(self, block):
58         # Проверяем блок на порог яркости и монотонности
59         l = np.absolute(block[self.low_coef]).sum()
60         h = np.absolute(block[self.high_coef]).sum()
61         # return True
62         return l >= self.Pl and h <= self.Ph
63
64     def _encode_block(self, block, bit):
65         """
66             В данном блоке block кодирует bit за счет
67             изменения соотношения между тремя псевдослучайными
68             элементами.
69         """
70
71         # С помощью ГПСЧ выбираем случайные элементы блока
72         k1, k2, k3 = random.sample(self.stego_coef, 3)
73
74         # Кодируем ноль, устанавливая block[k3] минимальным
75         # из трех элементов так, чтобы это соотношение
76         # сохранилось после квантования коэффициентов
77         if bit == False:
78             m = min(block[k1], block[k2])
79             block[k3] = m - self.P / 2
80
81             if block[k1] == m:
82                 block[k1] += self.P / 2
83             else:
84                 block[k2] += self.P / 2
85
86         # Кодируем единицу, устанавливая block[k3] максимальным
87         # из трех элементов так, чтобы это соотношение
88         # сохранилось после квантования коэффициентов
89         else:

```

```

89         m = max(block[k1], block[k2])
90         block[k3] = m + self.P / 2
91
92         if block[k1] == m:
93             block[k1] -= self.P / 2
94         else:
95             block[k2] -= self.P / 2
96
97     return block
98
99 def _decode_block(self, block):
100    """
101    Для данного блока block декодирует
102    bit, закодированный с помощью соотношения
103    между тремя псевдослучайными элементами
104    """
105
106    # С помощью ГПСЧ выбираем случайные элементы блока
107    k1, k2, k3 = random.sample(self.stego_coef, 3)
108
109    # Находим минимум и максимум разницы между третьим
110    # и двумя остальными элементами
111    M = max(block[k1], block[k2], block[k3])
112    m = min(block[k1], block[k2], block[k3])
113
114    if block[k3] == M:
115        return 1
116
117    if block[k3] == m:
118        return 0
119    else:
120        return -1
121
122 def encode(self):
123    """
124    Кодирует сообщение в контейнер
125    """
126
127    # Инициализируем ГПСЧ
128    random.seed(self.seed)
129
130    # Разбиваем массив ДКП коэффициентов на блоки
131    blocks = self._blockshaped(self.container, 8, 8)
132
133    # Находим положение подходящих блоков
134    mask = np.array([self._check_block(block) for block in blocks])
135
136    # В подходящих блоках меняем соотношения коэффициентов
137    nice_blocks = blocks[mask]
138    print(len(nice_blocks))
139
140    # Преобразуем сообщение в бинарный вид

```

```

135     np_message = np.unpackbits(np.frombuffer(
136         self.message, dtype=np.uint8)).ravel()
137     # Находим длину сообщения
138     n = len(np_message)
139     # Кодируем сообщение
140     new_nice_blocks = []
141     for i in range(n):
142         new_nice_blocks.append(self._encode_block(
143             nice_blocks[i], np_message[i]))
144     # Перезаписываем подходящие блоки
145     nice_blocks[:n] = np.array(new_nice_blocks)
146     blocks[mask] = nice_blocks
147     # Соединяем блоки обратно в контейнер
148     self.container = self._blockshaped(blocks, *self.container.shape)
149     # Сохраняем информацию в изображение
150     self.dct.coef_arrays[0].ravel()[:] = self.container.ravel()
151     self.x = 0
152
153 def decode(self):
154     """
155     Декодирует сообщение из контейнера
156     """
157     # Инициализируем ГПСЧ
158     random.seed(self.seed)
159     # Разбиваем массив ДКП коэффициентов на блоки
160     blocks = self._blockshaped(self.container, 8, 8)
161     message = []
162     # Находим позиции подходящих блоков
163     mask = np.array([self._check_block(block) for block in blocks])
164     # Находим подходящие блоки
165     nice_blocks = blocks[mask]
166     print(len(nice_blocks))
167     # Декодируем сообщение
168     for block in nice_blocks:
169         # Пробуем декодировать бит
170         bit = self._decode_block(block)
171         # -1 сигнализирует о том,
172         # что декодирование не удалось
173         if bit != -1:
174             message.append(bit)
175     # Из бит собираем исходное сообщение
176     message = np.packbits(message)
177     # Преобразуем его в байты
178     return message.tobytes()
179
180 def save(self):

```

```

181     """
182     Перезаписывает исходный файл
183     новый контейнером.
184     """
185     jio.write(self.dct, self.file_name)
186
187     def save_as(self, file_name):
188         """
189         Сохраняет контейнер в файл,
190         заданный параметром file_name.
191         """
192         jio.write(self.dct, file_name)
193
194     def _blockshaped(self, arr, nrows, ncols):
195         """
196         Возвращает массив формы (n, nrows, ncols), где
197          $n * nrows * ncols = arr.size$ 
198
199         Если массив – это матрица, тогда возвращает массив,
200         выглядящий как разбиение этой матрицы на подматрицы.
201         """
202         h, w = arr.shape
203         return (arr.reshape(h//nrows, nrows, -1, ncols)
204                 .swapaxes(1, 2)
205                 .reshape(-1, nrows, ncols))
206
207     def _unblockshaped(self, arr, h, w):
208         """
209         Возвращает матрицу формы (h, w), где
210          $h * w = arr.size$ 
211
212         Если матрица формы (n, nrows, ncols), где n – это подматрицы
213         формы (nrows, ncols), тогда возвращает матрицу, составленную
214         из этих подматриц.
215         """
216
217         n, nrows, ncols = arr.shape
218         return (arr.reshape(h//nrows, -1, nrows, ncols)
219                 .swapaxes(1, 2)
220                 .reshape(h, w))
221
222     def main():
223         with open("Messages/A Sound of Thunder.txt", "rb") as f:
224             message = f.read()[:100]
225
226             message = ("Hello").encode()

```

```
227     size = len(message)
228     jpg = DCT("Images/Lenna.jpg", message)
229     jpg.encode()
230     jpg.save_as("Images/DCT_Lenna.jpg")
231     jpg = DCT("Images/DCT_Lenna.jpg")
232     decoded = jpg.decode()[:size]
233
234     print(decoded)
235
236
237 if __name__ == "__main__":
238     main()
```