

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ярославский государственный университет им. П.Г. Демидова»
Кафедра компьютерной безопасности и математических
методов обработки информации

Сдано на кафедру

« ____ » _____ 20__ г.

Заведующий кафедрой

д. ф.-м. н., профессор

_____ Дурнев В.Г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
Исследование методов внедрения
цифровых водяных знаков на основе
вейвлет-преобразований

специальность 10.05.01 Компьютерная безопасность

Научный руководитель
доцент кафедры КБиММОИ

Мурин Дмитрий Михайлович

(подпись)

« ____ » _____ 20__ г.

Студент группы КБ61-СО
Шарунцова Анна Александровна

(подпись)

« ____ » _____ 20__ г.

Ярославль 2020

Реферат

Объем 66 с., 3 гл., 8 рис., 1 табл., 8 источников, 10 прил.

Стеганография, вейвлет-преобразования, преобразования Хаара, цифровой водяной знак, внедрение цифровых водяных знаков, робастность.

Объектом исследования являются методы внедрения цифровых водяных знаков в изображения с помощью вейвлет-преобразований.

Цель работы – реализация и оптимизация алгоритма встраивания цифрового водяного знака в изображение с помощью вейвлет-преобразований.

В главе 1 даны определения некоторых понятий стеганографии, а также рассказано о возможностях применения цифровой стеганографии в современном мире. Глава 2 представляет собой теоретическую сводку о вейвлет-преобразованиях, в частности, преобразованиях Хаара. Глава 3 описывает алгоритм И. Р. Ким по встраиванию цифрового водяного знака в изображение с помощью вейвлет-преобразований. Также в ней проведен анализ указанного алгоритма и предложены способы оптимизации исходя из основных выявленных недостатков алгоритма.

Работа представляет ценность в области защиты авторских прав и может быть использована для защиты изображений от незаконного использования.

Оглавление

Реферат	2
Оглавление	3
Введение	5
1. Стеганография	7
1.1 Виды стеганографии	7
1.1.1. Классическая стеганография	7
1.1.2. Компьютерная стеганография	8
1.1.3 Цифровая стеганография	10
1.2 Задачи цифровой стеганографии	10
1.3 Встраивание информации в изображения	12
1.3.1 Основные понятия	12
1.3.2 Физиологические особенности человеческого восприятия	13
1.3.3 Классификация методов встраивания ЦВЗ в изображения	15
1.3.4 Атаки на стеганографические системы	16
1.3.5 Оценка качества стеганосистемы	21
2. Вейвлет-преобразования	23
2.1 Определение вейвлет-преобразования	23
2.2 Применение вейвлет-преобразования	26
2.3 Примеры наиболее часто используемых вейвлетов	27
2.4 Вейвлет-преобразование Хаара	28
2.4.1 Преобразование Хаара для одномерного сигнала	29
2.4.2 Преобразование Хаара для двумерного сигнала	29
3. Алгоритм внедрения цифрового водяного знака в изображение с помощью вейвлет-преобразований	31
3.1 Алгоритм И.Р. Ким	31
3.1.1 Встраивание ЦВЗ	31
3.1.2 Извлечение ЦВЗ	34
3.2 Программная реализация алгоритма И.Р. Ким	34
3.2.1 Хранение и обработка изображений	34
3.2.2 Вейвлет-преобразование	35

3.2.3 Алгоритмы встраивания и извлечения ЦВЗ.....	36
3.3 Анализ алгоритма	37
3.3.1 Анализ быстродействия и эффективности	37
3.3.2 Анализ на робастность	38
3.4 Оптимизация алгоритма	39
3.4.1 Встраивание ЦВЗ в несколько цветовых компонент	40
3.4.2 Предварительный расчет максимально возможного и оптимального размера ЦВЗ	40
3.4.3 Вывод предупреждения о невозможности встроить ЦВЗ целиком	41
3.4.4 Алгоритм определения процента совпадения восстановленного ЦВЗ с исходным в случае потери данных	41
3.4.5 Восстановление изображения после кадрирования для дальнейшего извлечения ЦВЗ.....	41
Заключение	43
Список литературы	44
Приложение А	45
Приложение Б.....	46
Приложение В.....	49
Приложение Г	51
Приложение Д	54
Приложение Е.....	57
Приложение Ж.....	63
Приложение З	64
Приложение И	65
Приложение К.....	66

Введение

В современном мире, благодаря активному развитию технологий, передача и дублирование данных осуществляется достаточно просто. В связи с этим остро стоит вопрос о защите интеллектуальной собственности. Нередко возникают ситуации незаконного копирования и использования произведений интеллектуальной собственности. В таких случаях автор вынужден обращаться в суд и доказывать свое исключительное право на собственное произведение. Но как это сделать?

Существует такое понятие, как презумпция авторства, которое означает, что лицо, указанное в качестве автора на оригинале или экземпляре произведения, считается его автором, если не доказано иное. Однако в случае с цифровым контентом это можно легко подделать. Можно назначить экспертизу по установлению авторства, но это долгая, сложная, а главное, дорогая процедура.

Каждый автор желает защитить свои произведения. К сожалению, в нашей стране нет такого понятия, как государственная регистрация авторского права, поэтому люди прибегают к различным способам «собственной» регистрации. Например, рукописи можно отправить по почте самому себе и сохранить, не вскрывая конверта, на котором будет зарегистрирована дата отправления. Тогда в случае необходимости доказать свое авторство, достаточно распечатать конверт. Многие интернет-магазины по продаже стоковых фотографий во избежание копирования и незаконного использования накладывают поверх полупрозрачную надпись с названием магазина размером почти во всю фотографию. Это эффективно, но визуально портит продукт.

Что если «подписывать» фотографию таким образом, чтобы это не было заметно глазу, но при этом было очень сложно удалить, не прибегая к сильным искажениям фотографии? Эту проблему решает один из способов современной цифровой стеганографии – цифровой водяной знак (ЦВЗ). Существует множество различных схем по встраиванию ЦВЗ, специализированных для определенных нужд.

В данной работе будут рассмотрены основные виды ЦВЗ и возможные способы их реализации. Также будет описан один из алгоритмов по встраиванию ЦВЗ в изображения с целью защиты авторских прав и его программная реализация. В основе алгоритма лежит вейвлет-анализ, который является на сегодняшний день одной из самых перспективных технологий анализа данных. Его инструменты находят применение в самых различных сферах интеллектуальной деятельности,

таких как фильтрация и предварительная обработка данных, анализ состояния и прогнозирования ситуации на фондовых рынках, распознавание образов. Также он используется при обработке и синтезе различных сигналов, например речевых, медицинских, для решения задач сжатия и обработки изображений, при обучении нейросетей и во многих других случаях.

1. Стеганография

Стеганография (от греч. $\sigma\tau\epsilon\upsilon\alpha\nu\acute{o}\varsigma$ «скрытый» + $\gamma\rho\acute{\alpha}\phi\omega$ «пишу»; букв. «тайнопись») — способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи (хранения).

В отличие от криптографии, которая скрывает содержимое тайного сообщения, стеганография скрывает сам факт его существования. Преимущество стеганографии над чистой криптографией состоит в том, что сообщения не привлекают к себе внимания. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Первая запись об использовании стеганографии встречается в трактате Геродота «История», относящегося к 440 году до н. э. В трактате были описаны два метода скрытия информации. Демарат, царь Спарты в 515—491 годах до н.э., отправил предупреждение о предстоящем нападении на Грецию, записав его на деревянную подложку восковой таблички до нанесения воска. Второй способ заключался в следующем: на обритую голову раба записывалось необходимое сообщение, а когда его волосы отрастали, он отправлялся к адресату, который вновь брил его голову и считывал доставленное сообщение.

Существует версия, что древние шумеры одними из первых использовали стеганографию, так как было найдено множество глиняных клинописных табличек, в которых одна запись покрывалась слоем глины, а на втором слое писалась другая. Однако противники этой версии считают, что это было вовсе не попыткой скрытия информации, а всего лишь практической потребностью.

1.1 Виды стеганографии

В настоящее время разделяют три вида стеганографии: классическая, компьютерная и цифровая. Рассмотрим каждый из них подробнее.

1.1.1. Классическая стеганография

Одним из наиболее распространённых методов классической стеганографии является использование симпатических (невидимых) чернил. Текст, записанный такими чернилами, проявляется только при определённых условиях (нагрев, освещение, химический проявитель и т. д.). Изобретённые ещё в I веке н. э. Филоном Александрийским, они

продолжали использоваться как в средневековье, так и в новейшее время, например, в письмах русских революционеров из тюрем.

Существуют также чернила с химически нестабильным пигментом. Текст, написанный этими чернилами, выглядит как текст, написанный обычной ручкой, но через определённое время нестабильный пигмент разлагается, и от текста не остаётся и следа. Хотя при использовании обычной шариковой ручки текст можно восстановить по деформации бумаги, этот недостаток можно устранить с помощью мягкого пишущего узла, наподобие фломастера.

Во время Второй мировой войны активно использовались микроточки микроскопические фотоснимки, вклеиваемые в текст писем.

Другие примеры классической стеганографии:

1. запись на боковой стороне колоды карт, расположенных в условленном порядке;
2. запись внутри варёного яйца;
3. «жаргонные шифры», где слова имеют другое обусловленное значение;
4. трафареты, которые, будучи положенными на текст, оставляют видимыми только значащие буквы;
5. геометрическая форма — метод, в котором отправитель старается скрыть ценную информацию, поместив её в сообщение так, чтобы важные слова расположились в нужных местах или в узлах пересечения геометрического рисунка;
6. семаграммы — секретные сообщения, в которых в качестве шифра используются различные знаки, за исключением букв и цифр;
7. узелки на нитках и т. д.

1.1.2. Компьютерная стеганография

Компьютерная стеганография — направление классической стеганографии, основанное на особенностях компьютерной платформы. Приведём некоторые примеры:

- *Использование зарезервированных полей компьютерных форматов файлов.* Суть метода состоит в том, что часть поля расширений, не заполненная информацией о расширении, по умолчанию заполняется нулями. Соответственно мы можем использовать эту «нулевую» часть для записи своих данных. Недостатком этого метода является низкая степень скрытности и малый объём передаваемой информации.

- *Соккрытие информации в неиспользуемых местах гибких дисков.* При использовании этого метода информация записывается в неиспользуемые части диска, к примеру, на нулевую дорожку. Недостатки: маленькая производительность, передача небольших по объёму сообщений.
- *Использование особых свойств полей форматов, которые не отображаются на экране.* Этот метод основан на специальных «невидимых» полях для получения сносков, указателей. К примеру, написание чёрным шрифтом на чёрном фоне. Недостатки: маленькая производительность, небольшой объём передаваемой информации.
- *Использование особенностей файловых систем.* При хранении на жёстком диске файл всегда (не считая некоторых файловых систем, например, ReiserFS) занимает целое число кластеров (минимальных адресуемых объёмов информации). К примеру, в ранее широко используемой файловой системе FAT32 (использовалась в Windows98/Me/2000) стандартный размер кластера — 4 КБ. Соответственно для хранения 1 КБ информации на диске выделяется 4 КБ памяти, из которых 1 КБ нужен для хранения сохраняемого файла, а остальные 3 ни на что не используются — соответственно их можно использовать для хранения информации. Недостаток данного метода: лёгкость обнаружения.

В последнее время приобрели популярность методы, когда скрытая информация передаётся через компьютерные сети с использованием особенностей работы протоколов передачи данных. Эта разновидность компьютерной стеганографии получила название **«сетевая стеганография»**. Типичные методы сетевой стеганографии включают изменение свойств одного из сетевых протоколов. Кроме того, может использоваться взаимосвязь между двумя или более различными протоколами с целью более надёжного сокрытия передачи секретного сообщения. Сетевая стеганография охватывает широкий спектр методов, в частности:

- *WLAN-стеганография* основывается на методах, которые используются для передачи стеганограмм в беспроводных сетях (Wireless Local Area Networks). Практический пример WLAN-стеганографии — система HICCUPS (Hidden Communication System for Corrupted Networks). HICCUPS использует несовершенства среды передачи — шумы и помехи, которые являются естественными причинами искажения данных.
- *LACK-стеганография* — сокрытие сообщений во время разговоров с использованием IP-телефонии. Например: использование пакетов, которые задерживаются или намеренно повреждаются и игнорируются приемником (этот метод называют LACK — Lost

Audio Packets Steganography) или сокрытие информации в полях заголовка, которые не используются.

Принцип функционирования LACK выглядит следующим образом. Передатчик выбирает один из пакетов голосового потока, и его полезная нагрузка заменяется битами секретного сообщения — стеганограммой, которая встраивается в один из пакетов. Затем выбранный пакет намеренно задерживается. Каждый раз, когда чрезмерно задержанный пакет достигает получателя, незнакомого со стеганографической процедурой, он отбрасывается. Однако, если получатель знает о скрытой связи, то вместо удаления полученных RTP-пакетов он извлекает скрытую информацию.

1.1.3 Цифровая стеганография

Цифровая стеганография — направление классической стеганографии, основанное на сокрытии или внедрении дополнительной информации в цифровые объекты, вызывая при этом некоторые искажения этих объектов. Но, как правило, данные объекты являются мультимедиа-объектами (изображения, видео, аудио, текстуры 3D-объектов) и внесение искажений, которые находятся ниже порога чувствительности среднестатистического человека, не приводит к заметным изменениям этих объектов. Кроме того, в оцифрованных объектах, изначально имеющих аналоговую природу, всегда присутствует шум квантования; далее, при воспроизведении этих объектов появляется дополнительный аналоговый шум и нелинейные искажения аппаратуры, все это способствует большей незаметности сокрытой информации.

В связи со спецификой данной работы, наибольший интерес представляет цифровая стеганография, а именно сокрытие данных в изображениях. Но, прежде чем рассмотреть подробно технологию сокрытия данных в изображении, стоит сказать несколько слов о различных сферах применения цифровой стеганографии.

1.2 Задачи цифровой стеганографии

Рассмотрим основные задачи, решаемые современной цифровой стеганографией.

1. **Защита конфиденциальной информации от несанкционированного доступа.** Это область использования цифровой стеганографии является наиболее эффективной при решении проблем защиты конфиденциальной информации. Встраивание скрытой информации происходит в общедоступную

мультимедийную информацию. Так, например, объем секретного сообщения в звуковых и графических файлах может составлять до 25 - 30 % от размера файла. Причем, аудиовизуальные изменения таковы, что не обнаруживаются при прослушивании и просмотре файлов большинством людей, даже если факт сокрытия известен. Такой способ сокрытия секретной информации используется в военной сфере, а также в случаях, когда нельзя использовать криптографию.

2. **Преодоление систем мониторинга и управления сетевыми ресурсами.** Стеганографические методы позволяют противостоять попыткам контроля над информационным пространством при прохождении информации через серверы управления локальных и глобальных вычислительных сетей.

Например, существует утилита Camera/Shy, которая работает на основе браузера Internet Explorer, не оставляя в нем истории деятельности. Используя стеганографическую технику LSB и алгоритм шифрования AES с 256-разрядным ключом, она функционирует очень быстро и позволяет скрывать сообщения в gif-файлах. Кроме того, эта программа способна также автоматически сканировать HTML-страницы на наличие графических изображений со скрытой информацией. По заявлению авторов, эта программа была создана “для обхода национальных межсетевых экранов, что дает возможность безопасно обмениваться любым цифровым контентом через Интернет”.

3. **Камуфлирование программного обеспечения (ПО).** Применяется в тех случаях, когда использование ПО незарегистрированными пользователями является нежелательным. ПО может быть закомуфлировано под стандартные универсальные программные продукты (например, текстовые редакторы) или скрыто в файлах мультимедиа и использоваться только лицами, имеющими на это права. Таким образом, обеспечивается многоуровневый санкционированный доступ к ПО.
4. **Защита авторских прав.** Одним из наиболее перспективных направлений цифровой стеганографии является технология использования цифровых водяных знаков – в данном случае, создание невидимых глазу знаков защиты авторских прав на графические и аудио файлы. Такие ЦВЗ, помещенные в файл, могут быть распознаны специальными программами, которые извлекут из файла много полезной информации: когда создан файл, кто владеет авторскими правами, как вступить в контакт с автором и т.д.

Данная дипломная работа посвящена исследованию способов сокрытия данных в изображении с целью защиты авторских прав. Дадим

определения используемым понятиям и рассмотрим возможные способы сокрытия данных в изображении и существующие атаки на них.

1.3 Встраивание информации в изображения

1.3.1 Основные понятия

Современные компьютерные системы обеспечивают доступность и простоту дублирования фото, аудио, видео и других типов данных. При этом сами данные могут являться цифровой формой представления художественного, литературного, музыкального или иного произведения, на которое распространяется авторское право. В этом случае, для защиты авторского права, наносится специальная метка, которая остается невидимой для глаз, но распознается специальным программным обеспечением (ПО). Такой меткой могут быть цифровой водяной знак и цифровой отпечаток пальца.

Цифровой водяной знак (ЦВЗ) – невидимая информация, маркирующая изображение со специальной целью. В частности, ЦВЗ может играть роль метки авторского права, которая встраивается для фиксации владельца изображения, цифровой подписи и др.

Стойкий (робастный) цифровой водяной знак – ЦВЗ, который нельзя удалить или модифицировать даже автору без заметного искажения изображения-носителя.

Цифровой отпечаток пальца, или идентификационный номер, – метка, подобная серийному номеру, цель которого идентифицировать случаи нарушения лицензионных соглашений.

Слепое встраивание информации – встраивание информации, которое не требует наличия ни исходного изображения, ни самого ЦВЗ для извлечения на стороне получателя.

Схемы встраивания ЦВЗ основаны на том, что зрение человека не чувствительно к малым амплитудным изменениям, а также к малым изменениям во временной или в частотной областях.

В качестве входных данных в любой схеме встраивания ЦВЗ выступают оригинальное изображение (контейнер), встраиваемая информация (ЦВЗ) и, опционально, секретный ключ, ограничивающий доступ к скрытой информации. Встраиваемая информация может быть представлена в виде бинарной последовательности, текста, двумерного объекта, цифрового изображения.

Выходным результатом будет маркированное изображение (стеганоконтейнер), иначе говоря, модифицированное исходное изображение.

Встраиваемые ЦВЗ могут быть трех типов:

- робастные,
- хрупкие,
- полухрупкие.

Под робастностью, как уже было сказано выше, понимается устойчивость ЦВЗ к различным воздействиям на стеганоконтейнер. Робастные ЦВЗ могут быть трех типов. Это ЦВЗ, которые могут быть обнаружены всеми желающими, обнаружены хотя бы одной стороной, либо это могут быть ЦВЗ, которые трудно модифицировать или извлечь из контента (контейнера).

Хрупкие ЦВЗ разрушаются при незначительной модификации заполненного контейнера и применяются для аутентификации сигналов (изображений). Отличие от средств электронной цифровой подписи заключается в том, хрупкие ЦВЗ все же допускают некоторую модификацию контейнера. Это важно для защиты мультимедийной информации, так как законный пользователь может, например, пожелать сжать изображение. Другое отличие заключается в том, хрупкие ЦВЗ должны не только отразить факт модификации контейнера, но также вид и местоположение модификации.

Полухрупкие ЦВЗ устойчивы по отношению к одним воздействиям и неустойчивы по отношению к другим. Полухрупкие ЦВЗ специально проектируются так, чтобы быть неустойчивыми по отношению к определенному виду операций. Например, они могут позволять сжатие изображения, но запрещать вырезку из него или вставку в него фрагмента.

1.3.2 Физиологические особенности человеческого восприятия

При разработке алгоритмов встраивания ЦВЗ необходимо учитывать особенности зрительной системы человека (ЗСЧ), поскольку незаметность внедренных сообщений является одним из главных требований к любой стеганосистеме. Особенности восприятия зрительной систем подразделяются на две группы: низкоуровневые (физиологические) и высокоуровневые (психофизиологические).

Среди низкоуровневых особенностей человеческого восприятия видеоданных выделяют три наиболее важных свойства, влияющих на заметность постороннего шума в изображении:

- чувствительность зрения к изменению контрастности (яркости) изображения;
- частотную чувствительность;
- эффект маскирования сигнала изображения.

Чувствительность к изменению яркости можно определить следующим образом. — Испытуемому показывают некоторую однотонную картинку. После того как глаз адаптировался к ее освещенности I , постепенно изменяют яркость. Изменение освещенности ΔI продолжают до тех пор, пока оно не будет обнаружено. На большом диапазоне изменений яркости в области средних значений контраст примерно постоянен, тогда как для малых и больших яркостей значение порога возрастает.

Частотная чувствительность зрительной системы проявляется в том, что человек гораздо более восприимчив к низкочастотному (НЧ), чем к высокочастотному (ВЧ) шуму. Это связано с неравномерностью амплитудно-частотной характеристики зрительной системы человека. Экспериментально ее можно определить при помощи того же опыта, что и при яркостной чувствительности. Но на этот раз в центральном квадрате изменяются пространственные частоты до тех пор, пока изменения не станут заметными.

Эффект маскирования сигнала изображения заключается в следующем. Фоторецепторы сетчатки человеческого глаза, отвечающие за цветное зрение, так называемые колбочки, делятся на три вида, каждый из которых чувствителен к определенным длинам световых волн (коротких, средних и длинных). Поэтому зрительная информация воспринимается в виде трех составляющих, возбуждающих нервные окончания в глазу через определенные подканалы.

Каждая из таких составляющих отличается по частотным характеристикам, а также имеет различную пространственную ориентацию. При рассмотрении изображений, сохраненных в цветовой модели RGB, зрительная система человека (ЗСЧ) наименее восприимчива к информации из синего цветового канала. В красном канале человеческий глаз воспринимает семь бит из восьми, в зеленом — из восьми и лишь в синем канале из восьми бит воспринимается всего четыре бита.

Психофизиологические свойства человеческого зрения проявляются после обработки поступившей от зрительного анализатора первичной информации. Они направлены на «подстройку» информации под изображение. ЗСЧ чувствительна к высококонтрастным участкам цифрового изображения, к размерам, форме, местоположению и цвету информационных фрагментов. Однако при разработке стегосистем психофизиологические свойства ЗСЧ в настоящее время практически не учитываются.

1.3.3 Классификация методов встраивания ЦВЗ в изображения

Для методов встраивания водяных знаков может использоваться широкий диапазон преобразований в любой области. Перед внедрением или извлечением водяного знака исходные данные могут быть преобразованы в пространственную область методом Фурье или дискретного косинусного преобразования, вейвлет-преобразования или даже в область фрактального кодирования, что позволяет использовать конкретные преобразования. В этих областях могут быть произведены следующие преобразования:

- модификация наименее значимого бита,
- добавление шума,
- изменение порядка коэффициентов,
- удаление коэффициента,
- деформация части данных и блоков на основании их сходства.

Кроме того, если учитывать визуальную модель человека, то визуальное воздействие изменений на человека нужно свести к минимуму. В таком случае модификации могут быть адаптированы с учетом дальнейшей обработки или используемого формата сжатия данных.

Методы встраивания ЦВЗ можно классифицировать по различным критериям, в том числе, по способу их внедрения, по способу извлечения, а также по области встраивания (рис. 1).



Рис. 1 Классификация методов встраивания ЦВЗ

По способу извлечения скрытой информации из контейнера алгоритмы маркировки изображений можно разделить на три группы:

- алгоритмы, выполняющие поиск секретного сообщения без исходного промаркированного изображения (слепые алгоритмы);
- алгоритмы, требующие наличия исходного изображения;
- алгоритмы, выполняющие поиск сообщений с использованием фрагмента оригинала контейнера.

По способу внедрения ЦВЗ существующие методы можно разделить на следующие группы:

- линейные, основанные на линейной модификации изображения;
- нелинейные, использующие векторное или скалярное преобразование;
- другие (в том числе использующие фрактальные преобразования).

По области встраивания скрытых данных существующие методы можно разделить на две группы:

- пространственные, основанные на внедрении бит ЦВЗ в результате изменений яркостных или цветовых составляющих изображения;
- спектральные, основанные на декомпозиции маркируемых областей изображения.

1.3.4 Атаки на стеганографические системы

Рассмотрим классификацию атак на системы цифровых водяных знаков.

Классификация предназначена для того, чтобы выделить основные типы атак, а также проанализировать и определить критерии уязвимости и устойчивости ЦВЗ к деструктивным воздействиям, ведь именно список атак определяет основу для оценки стойкости ЦВЗ.

Выделяют следующие виды атак (рис. 2).

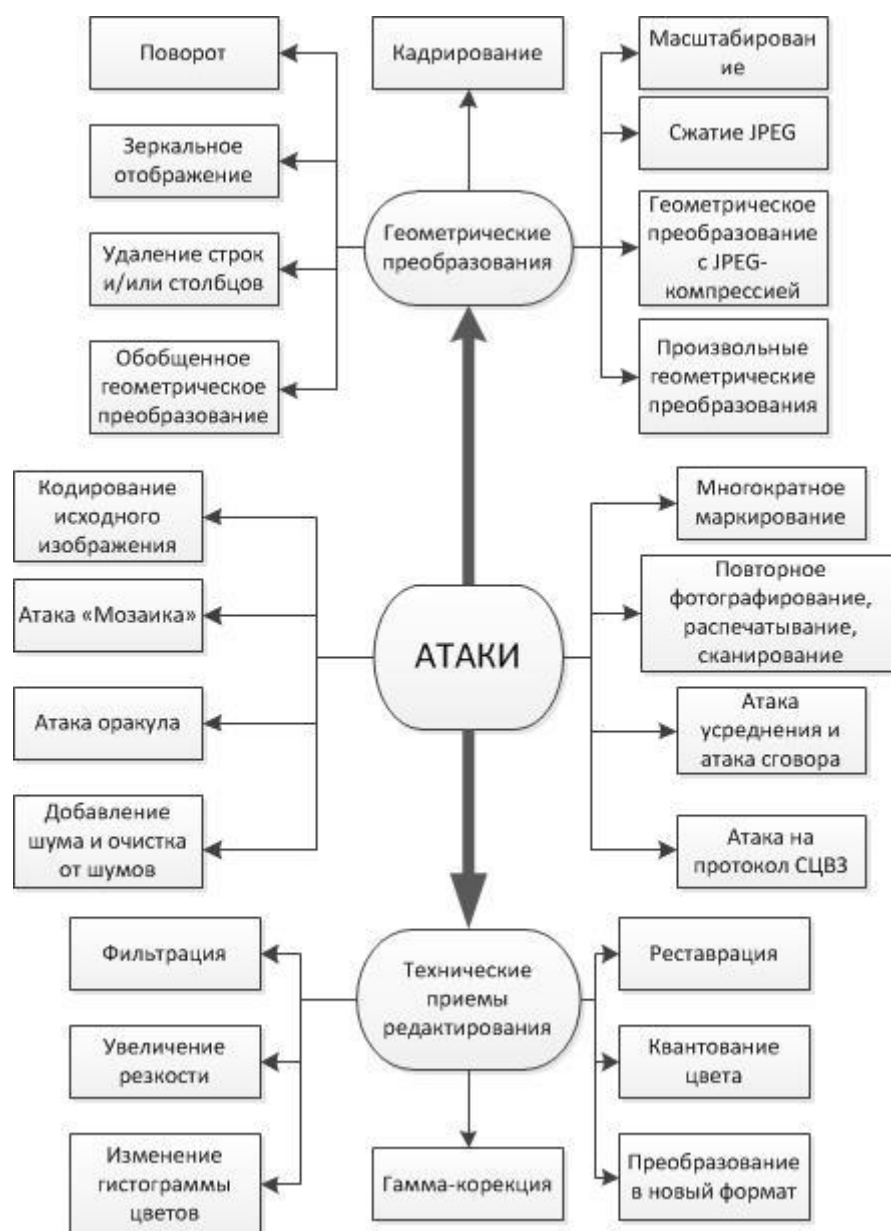


Рис. 2 Виды атак на системы ЦВЗ

Геометрические преобразования.

Зеркальное отображение. Большинство компьютерных изображений можно зеркально отобразить относительно вертикальной или горизонтальной оси. Однако немногие системы ЦВЗ могут сохранить

внедренный знак после данного преобразования. При этом основной проблемой является рассинхронизация стеганодекодера.

Поворот. Поворот изображения на небольшой угол часто применяется к отсканированному изображению, чтобы выровнять картинку по горизонтали или вертикали, но может применяться и для того, чтобы не обнаруживался водяной знак. Обычно поворот совмещается с кадрированием.

Кадрирование (обрезка и наращивание изображения). В некоторых случаях нарушители заинтересованы «центральной» частью материала, защищенного авторским правом. Тогда они вырезают центральный сегмент изображения. Однако рассеивание (размножение) ЦВЗ по всей площади изображения предотвращает вырезание встроенного знака.

Масштабирование. Масштабирование применяют, когда цифровое изображение с высоким расширением используется для электронных приложений, таких как публикации в Интернете или отправка по электронной почте. Масштабирование бывает пропорциональное и непропорциональное. Под пропорциональным масштабированием понимают такое, при котором коэффициенты масштабирования по горизонтали и вертикали одинаковы. Непропорциональное масштабирование использует различные коэффициенты по горизонтали и вертикали. Достаточно часто методы ЦВЗ устойчивы только к пропорциональному масштабированию.

Удаление строк и/или столбцов. Удаление нескольких строк или столбцов изображения, выбранных псевдослучайным образом из всей картинки, считается эффективной атакой против внедрения ЦВЗ.

Сжатие JPEG. В настоящее время JPEG — один из широко используемых алгоритмов сжатия изображения, поэтому любая система ЦВЗ должна быть устойчива к сжатию. Важным является показатель уровня сжатия, рекомендуется проверять устойчивость к JPEG-сжатию до 70 %.

Обобщенное геометрическое преобразование. Обобщенное геометрическое преобразование — это комбинация непропорционального масштабирования, поворота и обрезания.

Произвольные геометрические искажения. Программные инструментальные средства (например, StirMark) используют различные комбинации геометрических искажений для оценки устойчивости систем ЦВЗ к атакам.

Геометрические искажения вместе с JPEG-компрессией. Следует отдельно выделить комбинацию геометрического преобразования и сжатия JPEG, так как это очень распространенная операция при редактировании цифровых фотографий. Однако исчерпывающий тест на устойчивость к атакам должен включать и обратное к JPEG-сжатию преобразование, так как подобное может использоваться злоумышленником.

Технические приемы редактирования.

Преобразование в новый формат. Для надежного скрывания водяных знаков необходимо, чтобы методы внедрения были инвариантны (устойчивы) относительно множества методов преобразования цифрового образа в новый формат файла.

Фильтрация. Фильтрация включает линейные и нелинейные фильтры, применяемые с целью редактирования изображения. Часто используют медианный и гауссовский фильтры. Фильтрацией посредством сглаживания образа можно удалить ЦВЗ. Современные системы маркировки не позволяют отфильтровать ЦВЗ без значительных повреждений самого образа.

Увеличение резкости. Функция увеличения резкости принадлежит к стандартным возможностям программного обеспечения для обработки изображений. Это преобразование эффективно определяет шумы в высоких частотах, вводимые программами внедрения ЦВЗ, и поэтому может быть использовано для атак на системы ЦВЗ.

Изменение гистограммы цветов. Указанная атака включает увеличение (вытягивание) или выравнивание гистограммы с целью компенсации уровней цвета или изменения контрастности.

Гамма-коррекция — часто используемая операция для улучшения цветовой схемы изображений или адаптации изображений под дисплей, например, после сканирования.

Квантование цвета применяется при конвертации изображения в формат графического обмена GIF (Graphics Interchange Format), который используется для публикаций в Интернете. Квантование цвета сопровождается сглаживанием переходов и изменением ошибки квантования.

Реставрация обычно используется для снижения эффектов от специфических процессов деградации «бумажной копии».

Добавление шума и очистка от шумов. Многие ЦВЗ эффективно противостоят добавлению помех (аддитивный шум или некоррелированная мультипликативная помеха) в изображения. Этот вид преобразований широко рассмотрен в теории связи и теории обработки сигналов и разработаны алгоритмы защиты от шума. Однако важным является допустимый уровень шума относительно уровня сигнала самого маркированного изображения.

Повторное фотографирование, распечатывание, сканирование. Этот процесс вводит геометрические искажения, такие же, как шумоподобные искажения.

Атака «Мозаика». При этой атаке картинка разбивается на фрагменты, которые являются отдельными, но состыкованными в единое целое. Такие сегментированные изображения могут использоваться при оформлении Интернет-сайтов. Если злоумышленнику удастся разбить маркированное изображение на немаркированные фрагменты, то он сможет обмануть автоматическую систему поиска в Интернете произведений с внедренными ЦВЗ.

Атака усреднения и атака сговора. Имея несколько копий одной и той же картинки, но с разными знаками, можно удалить водяные знаки путем усреднения этих изображений (атака усреднения) или путем разделения всех копий изображения на небольшие части и затем составления оригинальной картинки, но из соответствующих частей различных копий (атака сговора).

Многократное маркирование. При этой атаке в контейнер добавляются несколько различных ЦВЗ. Однако современные разработки (например, P1свигеМагс) откажутся от выполнения добавления водяного знака, если другой уже внедрен. Следовательно, для избыточного маркирования атакующему нужен специальный доступ к маркирующей программе, например, дизассемблированный исходный код программного обеспечения. Водяной знак владельца должен оставаться даже после нанесения многих фальшивых ЦВЗ.

Атака оракула. Когда доступен открытый стеганодетектор, атакующий может удалить метку, последовательно внося небольшие изменения в изображение до тех пор, пока стеганодетектор еще определяет наличие ЦВЗ.

Атаки на протокол систем ЦВЗ. Указанные атаки направлены против функционирования самого протокола выработки и проверки ЦВЗ. Одной из таких атак является атака, основанная на инверсии последовательности действий при внедрении метки, в результате чего в

случае необратимости ЦВЗ злоумышленнику удастся промаркировать уже защищенное изображение. При разработке всей системы необходимо анализировать слабости не только ЦВЗ, но и стеганографические протоколы взаимодействия участников коммуникационного процесса.

Копирование (кража) исходного изображения. Атакующий с целью компрометации ЦВЗ может попытаться получить доступ к исходному немаркированному изображению. Рассмотренный список атак является основой для оценки стойкости ЦВЗ. При этом при разработке ЦВЗ необходимо тестировать систему не только для основных видов атак, но и для комбинаций перечисленных атак.

1.3.5 Оценка качества стеганосистемы

Создание и эксплуатация надежного стеганографического средства предусматривает наличие определенного инструментария для его контроля и оценки. Количественное оценивание стойкости стеганографической системы защиты к внешним воздействиям представляет собой достаточно сложную задачу, которая обычно на практике реализуется методами системного анализа, математического моделирования или экспериментального исследования.

Как правило, профессионально разработанная стеганосистема обеспечивает трехуровневую модель защиты информации:

- скрывание самого факта наличия защищаемой информации (первый уровень защиты);
- блокирование несанкционированного доступа к информации, осуществляемое путем избрания соответствующего метода скрывания информации (второй уровень защиты).
- предварительная криптографическая защита (шифрование) скрываемой информации.

Оценка качества основной характеристики стеганосистемы — *уровня скрытости* — обеспечивается путем проведения аналитических исследований (стеганоанализа) и натурных испытаний. Для оценки качества стеганографического скрывания часто используются известные методы из других областей, в первую очередь — криптоанализа.

Поскольку абонент-получатель может восстанавливать скрытую информацию принятого сообщения, то очевидно, что существует некий механизм ее извлечения.

Если нарушитель, выдвигая гипотезы о возможном стеганографическом преобразовании, имеет некоторый инструмент для их проверки, то он имеет шансы подтвердить факт существования скрытой информации, выполнив поиск механизма извлечения секретного сообщения и, наконец, раскрыть содержание сообщения. По этой причине, в первую очередь, для детектирования стеганограмм можно использовать разновидности описанных выше атак на стеганосистему и значительную часть методов криптоанализа.

Достаточно эффективны в некоторых случаях методы оценки уровня скрытости стеганосредств на основании анализа их статистических характеристик. Статистическая теория дает количественные критерии вероятности, что позволяет создавать детекторы, которые будут обнаруживать статистические расхождения между последовательностями. В случае наличия достаточного объема анализируемой информации с достаточно высокой вероятностью можно делать выводы об основных характеристиках последовательности, выделенной для анализа из контейнера.

Для сравнительного оценивания качества стеганографических средств разрабатываются разные показатели, дающие количественные оценки. Больше всего их разработано для стеганометодов, которые работают с изображениями и видео (методов ЦВЗ). Зачастую такие показатели оперируют с изображениями на уровне пикселей, хотя после надлежащей адаптации они применимы и к другим способам описания изображения, а также к аудиоданным.

2. Вейвлет-преобразования

Вейвлет-преобразование в настоящее время имеет большую популярность при обработке различных данных, так как оно устраняет те недостатки, которые присущи преобразованию Фурье. Преобразование Фурье даёт информацию о частотах исследуемого сигнала, но не даёт сведения о локальных особенностях сигнала. Поэтому при использовании преобразования Фурье можно получать информацию либо во временной области, либо в частотной. Вейвлет-преобразование справляется с этой задачей. Дадим математическое определение вейвлет-преобразования.

2.1 Определение вейвлет-преобразования

Традиционно для анализа временных рядов используется преобразование Фурье, дающее разложение исследуемого временного процесса $f(t)$ в ряд по тригонометрическим функциям, или в более общей форме записи

$$f(t) = \sum_{-\infty}^{+\infty} c_n e^{int}. \quad (1)$$

Коэффициенты c_n являются амплитудами гармонических колебаний соответствующей частоты и определяются формулой

$$c_n = (2\pi)^{-1} \int_0^{2\pi} f(t) e^{-int} dt. \quad (2)$$

Множество функций e^{int} образует ортонормированный базис пространства $L^2(0, 2\pi)$, которое является пространством определенных на $(0, 2\pi)$ функций, действительных, ограниченных на $(0, 2\pi)$ и непрерывных всюду за исключением конечного числа точек.

Аппарат Фурье-преобразований дает достаточно простые для расчетов формулы и прозрачную интерпретацию результатов, но имеет некоторые недостатки. Преобразование, например, не отличает сигнал, являющийся суммой двух синусоид, от ситуации последовательного включения синусоид, не дает информации о преимущественном распределении частот во времени, может дать неверные результаты для сигналов с участками резкого изменения. Исследуемые ряды также далеко не всегда удовлетворяют требованию периодичности и более того, как правило, заданы на ограниченном отрезке времени.

Основы вейвлет-анализа были разработаны в середине 80-х годов Гроссманом и Морле как альтернатива преобразованию Фурье для исследования временных (пространственных) рядов с выраженной неоднородностью. В отличие от преобразования Фурье, локализирующего частоты, но не дающего временного разрешения процесса, вейвлет-преобразование, обладающее самонастраивающимся подвижным частотно-временным окном, одинаково хорошо выявляет как низкочастотные, так и высокочастотные характеристики сигнала на разных временных масштабах. По этой причине вейвлет-анализ часто сравнивают с "математическим микроскопом", вскрывающим внутреннюю структуру существенно неоднородных объектов.

Подобно тому, как в основе аппарата преобразований Фурье лежит единственная функция $w(t) = e^{it}$, порождающая ортонормированный базис пространства $L^2(0, 2\pi)$ путем масштабного преобразования, так и вейвлет-преобразование строится на основе единственной базисной функции $\psi(t)$, принадлежащей пространству $L^2(R)$, т.е. всей числовой оси.

В западной литературе за этой функцией закрепилось название "вейвлет", что означает "маленькая волна", в отечественной иногда ее называют "всплеском", отражая в этом названии и локализацию, и осцилляционный характер поведения.

1. При конструировании базисной анализирующей функции $\psi(t)$ должны выполняться следующие необходимые условия.
2. Локализация – вейвлет должен быть локализован вблизи нуля аргумента как во временном, так и в частотном пространстве.
3. Нулевое среднее:

$$\int_{-\infty}^{\infty} \psi(t) dt = 0.$$

4. Как следствие, вейвлет должен быть знакопеременной функцией.
5. Ограниченность:

$$\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty.$$

6. Вейвлет должен быть достаточно быстро убывающей функцией временной (пространственной) переменной.

Базис одномерного дискретного вейвлет-преобразования (ДВП) строится на основе вейвлета $\psi(t)$ посредством операций сдвигов и растяжений вдоль оси t . Вводя аналог синусоидальной частоты и принимая для простоты в качестве ее значений степени двойки, получаем для функций базиса $\psi_{jk}(t) = 2^{j/2}\psi(2^j t - k)$.

Базис нормирован, если вейвлет имеет единичную норму.

Вейвлет называется ортогональным, если семейство $\{\psi_{jk}\}$ представляет ортонормированный базис функционального пространства $L^2(R)$, т.е. $\langle \psi_{jk}, \psi_{lm} \rangle = \delta_{jl}\delta_{km}$. В этом случае любая функция f из $L^2(R)$ может быть представлена в виде ряда

$$f(t) = \sum_{j,k=-\infty}^{+\infty} c_{jk}\psi_{jk}(t), \quad (3)$$

где

$$c_{jk} = 2^{j/2} \int_{-\infty}^{\infty} f(t)\psi(2^j t - k)dt. \quad (4)$$

Непрерывное вейвлет-преобразование (НВП) строится аналогичным образом с помощью непрерывных масштабных преобразований и переносов вейвлета $\psi(t)$ с произвольными значениями масштабного коэффициента a и параметра сдвига b :

$$W(a, b) = |a|^{-1/2} \int_{-\infty}^{\infty} f(t)\psi^*\left(\frac{t-b}{a}\right), \quad (5)$$

где символ $*$ обозначает операцию комплексного сопряжения.

Вейвлет-преобразование обратимо для функций f из $L^2(R)$

$$f(t) = \int_0^{\infty} \int_{-\infty}^{\infty} W(a, b)\psi\left(\frac{t-b}{a}\right)\frac{da db}{a^2} dt. \quad (6)$$

Таким образом, любая функция из $L^2(R)$ может быть представлена суперпозицией масштабных преобразований и сдвигов базисного вейвлета с коэффициентами, зависящими от масштаба (частоты) и параметра сдвига (времени).

Двухпараметрическая функция $W(a,b)$ дает информацию об изменении относительного вклада компонент разного масштаба во времени и называется спектром коэффициентов вейвлет-преобразования.

2.2 Применение вейвлет-преобразования

Перечислим некоторые области, где использование вейвлетов может оказаться (или уже является) весьма перспективным.

Обработка экспериментальных данных. Поскольку вейвлеты появились именно как механизм обработки экспериментальных данных, их применение для решения подобных задач представляется весьма привлекательным до сих пор. Вейвлет-преобразование дает наиболее наглядную и информативную картину результатов эксперимента, позволяет очистить исходные данные от шумов и случайных искажений, и даже "на глаз" подметить некоторые особенности данных и направление их дальнейшей обработки и анализа. Кроме того, вейвлеты хорошо подходят для анализа нестационарных сигналов, возникающих в медицине, анализе фондовых рынков и других областях.

Обработка изображений. Наше зрение устроено так, что мы сосредотачиваем свое внимание на существенных деталях изображения, отсекая ненужное. Используя вейвлет-преобразование, мы можем сгладить или выделить некоторые детали изображения, увеличить или уменьшить его, выделить важные детали и даже повысить его качество.

Сжатие данных. Особенностью ортогонального многомасштабного анализа является то, что для достаточно гладких данных полученные в результате преобразования детали в основном близки по величине к нулю и, следовательно, очень хорошо сжимаются обычными статистическими методами. Огромным достоинством вейвлет-преобразования является то, что оно не вносит дополнительной избыточности в исходные данные, и сигнал может быть полностью восстановлен с использованием тех же самых фильтров. Кроме того, отделение в результате преобразования деталей от основного сигнала позволяет очень просто реализовать сжатие с потерями – достаточно просто отбросить детали на тех масштабах, где они несущественны. Достаточно сказать, что изображение, обработанное вейвлетами, можно сжать в 3-10 раз без существенных потерь информации (а с допустимыми потерями – до 300 раз). В качестве примера отметим, что вейвлет-преобразование положено в основу стандарта сжатия данных MPEG4.

Нейросети и другие механизмы анализа данных. Большие трудности при обучении нейросетей (или настройке других механизмов

анализа данных) создает сильная зашумленность данных или наличие большого числа "особых случаев" (случайные выбросы, пропуски, нелинейные искажения и т.п.). Такие помехи способны скрывать характерные особенности данных или выдавать себя за них и могут сильно ухудшить результаты обучения. Поэтому рекомендуется очистить данные, прежде чем анализировать их. По уже приведенным выше соображениям, а также благодаря наличию быстрых и эффективных алгоритмов реализации, вейвлеты представляются весьма удобным и перспективным механизмом очистки и предварительной обработки данных для использования их в статистических и бизнес-приложениях, системах искусственного интеллекта и т.п.

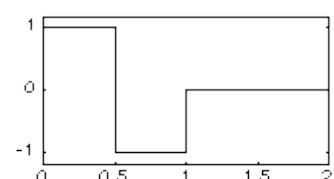
Системы передачи данных и цифровой обработки сигналов. Благодаря высокой эффективности алгоритмов и устойчивости к воздействию помех, вейвлет-преобразование является мощным инструментом в тех областях, где традиционно использовались другие методы анализа данных, например, преобразование Фурье. Возможность применения уже существующих методов обработки результатов преобразования, а также характерные особенности поведения вейвлет-преобразования в частотно-временной области позволяют существенно расширить и дополнить возможности подобных систем.

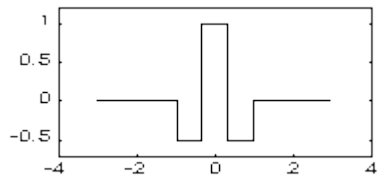
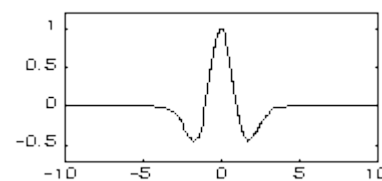
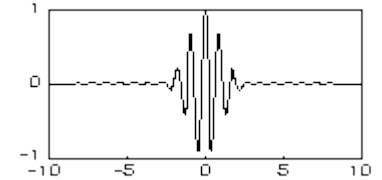
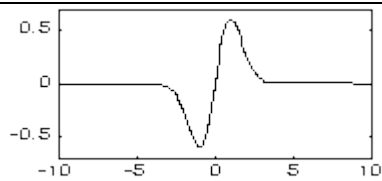
2.3 Примеры наиболее часто используемых вейвлетов

Ниже приведены примеры часто используемых вейвлетов (см. таблицу 1). Выбор того или иного класса анализирующих функций диктуется спецификой задачи, тем, какую информацию нужно извлечь из сигнала. В ряде случаев с помощью различных вейвлетов можно более полно выявить особенности анализируемого сигнала.

Таблица 1

Примеры наиболее часто используемых вейвлетов.

Вейвлет Хаара	
$\psi(t) = \begin{cases} 1, 0 \leq t < \frac{1}{2} \\ -1, \frac{1}{2} \leq t < 1 \\ 0, t < 0, t \geq 1 \end{cases}$	

Вейвлет «Французская шляпа»	
$\psi(t) = \begin{cases} 1, t \leq \frac{1}{2} \\ -\frac{1}{2}, \frac{1}{3} < t \leq 1 \\ 0, t > 1 \end{cases}$	
Вейвлет «Мексиканская шляпа»	
$\psi(t) = (1 - t^2)e^{-t^2/2}$	
Вейвлет Морле	
$\psi(r) = e^{ik_0 r - t^2/2}$	
Wave-вейвлет	
$\psi(t) = te^{-t^2/2}$	

2.4 Вейвлет-преобразование Хаара

Вейвлет Хаара — один из первых и наиболее простых вейвлетов. Он основан на ортогональной системе функций, предложенной венгерским математиком Альфредом Хааром в 1909 году. Вейвлеты Хаара ортогональны, обладают компактным носителем, хорошо локализованы в пространстве, но не являются гладкими.

Родительская (материнская) вейвлет-функция $\psi(t)$, определяющая детали сигнала, задается, как было указано в таблице 1, следующим образом

$$\psi(t) = \begin{cases} 1, 0 \leq t < \frac{1}{2} \\ -1, \frac{1}{2} \leq t < 1 \\ 0, t < 0, t \geq 1 \end{cases} \quad (7)$$

Рассмотрим последовательно, как происходит вейвлет-преобразование Хаара.

Для начала рассмотрим случай одномерного преобразования.

2.4.1 Преобразование Хаара для одномерного сигнала

Пусть имеется одномерный дискретный входной сигнал S . Каждой паре соседних элементов ставятся в соответствие два числа: $a_i = \frac{S_{2i} + S_{2i+1}}{2}$ и $b_i = \frac{S_{2i} - S_{2i+1}}{2}$. Повторяя данную операцию для каждого элемента исходного сигнала, на выходе получают два сигнала, один из которых является аппроксимированной версией входного сигнала — a_i , а второй содержит детализирующую информацию, необходимую для восстановления исходного сигнала. Аналогично, преобразование Хаара может быть применено к полученному сигналу a_i снова и тогда это уже будет преобразование Хаара второго уровня. Применять преобразование к коэффициентам аппроксимации можно до тех пор, пока их не останется меньше двух.

В итоге получается ступенчатое разложение, в котором коэффициенты детализации сохраняются, а коэффициенты аппроксимации раскладываются на коэффициенты следующего уровня (рис. 3).



Рис. 3 Разложение Хаара

Пример: пусть входящий сигнал представляется в виде строки из 8 значений яркости пикселей (220, 211, 212, 218, 217, 214, 210, 202). После применения преобразования Хаара получаются следующие две последовательности $a_1 = (215.5, 215, 215.5, 206)$ и $b_1 = (4.5, -3, 1.5, 4)$. Стоит заметить, что значения b_1 достаточно близки к 0. При необходимости сжать сигнал, некоторыми коэффициентами детализации можно пренебречь. Повторяя операцию, применительно к последовательности a_1 , получаем: (215.25, 210.75) и (0.25, 4.75). На примере преобразования Хаара хорошо видна структура дискретного вейвлет-преобразования сигнала. На каждом шаге преобразования сигнал распадается на две составляющие: приближение с более низким разрешением (аппроксимацию) и детализирующую информацию.

2.4.2 Преобразование Хаара для двумерного сигнала

Двумерное преобразование Хаара — это не что иное, как композиция одномерных преобразований Хаара. Пусть двумерный входной сигнал представляется матрицей S . После применения одномерного

преобразования Хаара к каждой строке матрицы S получают две новые матрицы, строки которых содержат аппроксимированную и детализирующую часть строк исходной матрицы. Аналогично к каждому столбцу полученных матриц применяют одномерное преобразование Хаара и на выходе получают четыре матрицы, одна из которых является аппроксимирующей составляющей исходного сигнала, а три оставшиеся содержат детализирующую информацию — вертикальную, горизонтальную и диагональную.

3. Алгоритм внедрения цифрового водяного знака в изображение с помощью вейвлет-преобразований

В реализованном мною алгоритме И.Р. Кима используется трехуровневое разложение с использованием вейвлет-преобразования Хаара. ЦВЗ встраивается в коэффициенты всех уровней разложения последовательно, начиная с третьего и заканчивая первым. Для встраивания выбираются коэффициенты, превышающие значение определенного порога.

Для встраивания используется аддитивный алгоритм

$$v'_i = v_i + \alpha v_i x_i, \quad (8)$$

где v'_i – измененный коэффициент, v_i – выбранный для внедрения коэффициент, x_i – бит ЦВЗ, α – коэффициент масштабирования, который регулируется для каждого уровня разложения. Для получения изображения, содержащего ЦВЗ, используется обратное 2D-ДВП.

Для извлечения ЦВЗ необходимо исходное изображение. Применяется инверсия формулы внедрения, учитывая адаптивно-уровневый коэффициент масштабирования α ,

$$x'_i = \frac{v'_i - v_i}{\alpha v_i}. \quad (9)$$

Количество встраиваемой информации относительно невелико. В среднем оно составляет примерно 1/120 от размера исходного изображения, однако при расчете возможного количества встраиваемой информации следует учитывать не только размер изображения, но и насыщенность цвета определенных составляющих.

3.1 Алгоритм И.Р. Ким

3.1.1 Встраивание ЦВЗ

Обобщенная блок-схема встраивания данных водяного знака с помощью 2D-ДВП изображена на рис. 4.

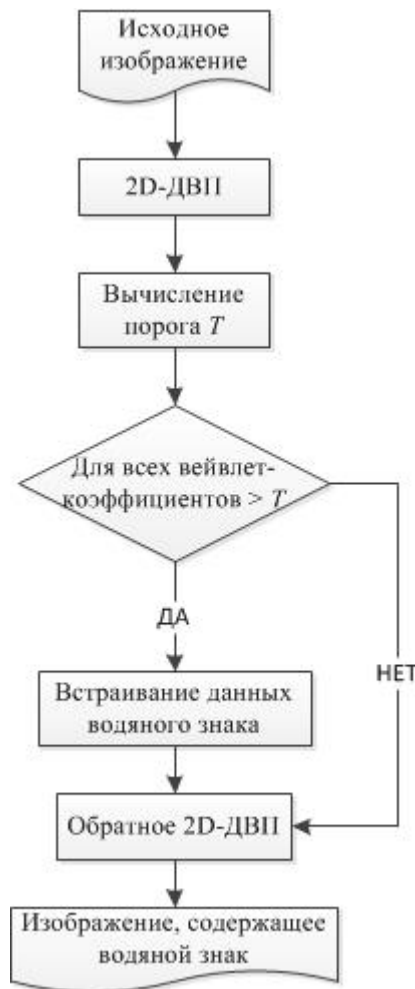


Рис. 4 Обобщенная блок-схема внедрения ЦВЗ

Шаг 1. Импортрование исходного изображения и извлечение его синей составляющей.

Как отмечено в пункте 1.3.2 настоящей работы изменения в синей составляющей наименее заметны человеческому глазу.

Шаг 2. Импортрование ЦВЗ и преобразование его в одномерный массив бит.

Шаг 3. Разложение синей составляющей исходного изображения с помощью вейвлет-преобразования Хаара на 3 уровня и формирование коэффициентов аппроксимации и детализации. В результате получается такая матрица (рис. 5), где LL – коэффициенты аппроксимации, HL – горизонтальные коэффициенты детализации, LH – вертикальные, HH – диагональные, а 1, 2 и 3 – это уровень разложения.

LL3	HL3	HL2	HL1
LH3	HH3		
LH2		HH2	
LH1			HH1

Рис. 5 Таблица коэффициентов, полученных при разложении Хаара до 3 уровня

Шаг 4. Вычисление порогов, в зависимости от которых будут выбраны пригодные для встраивания коэффициенты.

Пороги вычисляются отдельно для каждого уровня. На каждом уровне разложения находится абсолютный максимум для каждого набора коэффициентов, аппроксимации max_a и детализации – горизонтальных max_h^i , вертикальных max_v^i и диагональных max_d^i , где i – уровень разложения. Затем из них берется максимум Max_i . Порог T_i вычисляется по следующей формуле:

$$T_i = 2^{\log_2 Max_i - 1}. \quad (10)$$

Шаг 5. Процесс встраивания ЦВЗ. Начиная с 3-го уровня разложения, перебираются коэффициенты аппроксимации и детализации. Каждый коэффициент сравнивается со значением порога, соответствующего данному уровню. Коэффициенты, превышающие значение порога, изменяются в соответствии с формулой 8. Аналогичные операции производятся на 2 и на 1 уровне. В процессе выполнения этого шага функция периодически проверяет, сколько символов ЦВЗ осталось встроить. Если ЦВЗ встроено полностью, процесс останавливается. Соответственно, ЦВЗ не всегда встраивается во все коэффициенты.

Шаг 6. Восстановление синей составляющей изображения, в которое был встроен ЦВЗ, из измененных коэффициентов с помощью обратного вейвлет-преобразования. Изображение визуально ничем не должно отличаться от исходного.

3.1.2 Извлечение ЦВЗ

Шаг 1. Выполнение шага 1 и шага 3 алгоритма встраивания для изображения со встроенным ЦВЗ. В результате чего получаем коэффициенты аппроксимации и детализации 3 уровней разложения Хаара.

Шаг 2. Выполнение шага 1 и шага 3 алгоритма встраивания для исходного изображения. Это действие необходимо в связи с тем, что алгоритм И.Р. Ким является неслепым, а это значит, что для извлечения ЦВЗ необходимо исходное изображение.

Шаг 3. Вычисление порогов для коэффициентов. Вычисление производится с помощью исходного изображения аналогично шагу 4 предыдущего алгоритма.

Шаг 4. Извлечение ЦВЗ. Для извлечения ЦВЗ производится перебор коэффициентов синей составляющей исходного изображения, начиная с 3 уровня. Если коэффициент больше порога T_i , то находим соответствующий коэффициент в изображении со встроенным ЦВЗ и извлекаем бит ЦВЗ в соответствии с формулой 9.

Шаг 5. Перевод одномерного извлеченного массива бит ЦВЗ к двумерному виду и сохранение его в формате изображения. Для выполнения этого шага необходимо указать размеры исходного ЦВЗ.

3.2 Программная реализация алгоритма И.Р. Ким

Программа, реализующая алгоритм И. Р. Ким, описанный в пункте 2.1, написана на языке C# на платформе .NET Framework. Реализовано 9 классов. Программный код некоторых из них приведен в приложениях А-Ж. Далее следует описание основной функциональности каждого класса.

3.2.1 Хранение и обработка изображений

Для реализации хранения и обработки изображений было написано два класса. Вспомогательный класс DoublePixel (Приложение А) дает возможность оперировать значениями пикселей. Он содержит функции для быстрого доступа к конкретному пикселю, замены его значения и приведения пикселя к формату RGB.

Класс DoubleImage (Приложение Б) был реализован с целью замены стандартного класса для обработки изображений Bitmap. Он дает более

быстрый доступ к отдельным пикселям в отличие от класса `Bitmap` за счет использования класса `DoublePixel`. Класс `DoubleImage` позволяет извлекать отдельные цветовые компоненты по выбору и обновлять их. Этот класс также содержит функцию для преобразования изображения в стандартный формат `Bitmap` для последующего сохранения.

Для работы ЦВЗ оказалось недостаточно реализованных классов для работы с изображением, поэтому потребовалось реализовать класс `Watermark` (Приложение В). Он позволяет преобразовывать изображение в двумерный массив бит и обратно.

3.2.2 Вейвлет-преобразование

Для осуществления вейвлет-преобразования Хаара были реализованы два класса, `Haar` (Приложение Г) и `Wavelet` (Приложение Д).

Класс `Haar` содержит функцию `Transform`, принимающую на вход матрицу вещественных чисел, а также заданную ширину и высоту для преобразования отдельной части матрицы. Функция `Transform` сначала осуществляет преобразования по строкам матрицы, затем аналогично по столбцам. Для этого создается пустая матрица такого же размера для сохранения промежуточных результатов. В ходе преобразований по строкам для каждой строки k берется пара соседних чисел x_i и x_{i+1} , где i пробегает от 0 до $width/2$, где $width$ – это ширина матрицы, с шагом 2, и преобразуется следующим образом:

$$approx = \frac{1}{\sqrt{2}}(x_i + x_{i+1}) \quad (10)$$

$$detal = \frac{1}{\sqrt{2}}(x_i - x_{i+1}) \quad (11)$$

Затем полученные значения аппроксимации *approx* и детализации *detal* сохраняются в промежуточную матрицу в строку k с индексами i и $width/2$, соответственно.

Преобразования по столбцам производятся на промежуточной матрице, полученной в результате преобразований по строкам. Преобразования производятся аналогично описанным выше преобразованиям по строкам. Результат сохраняется в конечную матрицу, которую возвращает функция `Transform`. Если преобразования производятся только на отдельной части матрицы, остальную часть матрицы, полученной в результате преобразований, необходимо заполнить значениями из исходной матрицы. Для этого в классе `Haar` используется вспомогательная функция `FillRestMatrix`.

Класс Haar также содержит функцию Untransform, которая производит обратные преобразования для восстановления исходной матрицы.

Класс Wavelet содержит функцию Transform, которая принимает на вход матрицу вещественных чисел и необходимый уровень разложения. После проверки возможности разложения до указанного уровня запускается функция Transform класса Haar. Размеры изменяемой части матрицы зависят от уровня разложения. Для первого уровня разложения требуется вся матрица, для последующих уровней размеры изменяемой части определяются следующим образом:

$$variableWidth = width / 2^i,$$

$$variableHeight = height / 2^i,$$

где *width* и *height* – исходные размеры изображения, а *i* – текущий уровень разложения.

Класс Wavelet содержит функцию Untransform, которая производит обратное вейвлет-преобразование.

Также класс Wavelet содержит функцию GetCoefficient, которая возвращает матрицу коэффициентов (аппроксимации и детализации – горизонтальных, вертикальных и диагональных) в зависимости от требуемого уровня разложения.

3.2.3 Алгоритмы встраивания и извлечения ЦВЗ

Алгоритмы по встраиванию и извлечению ЦВЗ реализованы в классе JRKimAlgorithm (Приложение Е). Также реализован вспомогательный класс Threshold (Приложение Ж) для подсчета порогов.

Функция KIMembed осуществляет встраивание ЦВЗ в изображение. Она получает на вход исходное изображение и ЦВЗ. Далее извлекается синяя составляющая исходного изображения с помощью функции GetColorComponent класса DoubleImage. Затем вызывается функция Transform класса Wavelet с синей составляющей и значением 3 уровня разложения в виде параметров. Затем все коэффициенты разложения извлекаются из полученной в результате разложения матрицы с помощью функции GetCoefficient класса Wavelet и сохраняются в отдельные двумерные массивы.

Затем необходимо получить значения порогов. Для этого используется функция GetTreshold класса Threshold. Функция GetTreshold

перегружена и может принимать на вход 3 или 4 матрицы в зависимости от уровня разложения, поскольку на 1 и 2 уровнях разложения используются только коэффициенты детализации, а на 3 уровне еще и коэффициенты аппроксимации.

После получения порогов ЦВЗ преобразуется с помощью функции TransformWatermark. Ее назначение поменять все нулевые биты ЦВЗ на значение -1 для осуществления дальнейших арифметических операций.

Затем происходит последовательное встраивание ЦВЗ в коэффициенты, начиная с коэффициентов аппроксимации с 3 уровня разложения и заканчивая коэффициентами детализации 1 уровня разложения.

После чего собирается новая матрица с измененными коэффициентами с помощью функции SetCoefficient класса Wavelet. К ней применяется обратное вейвлет-преобразование Хаара с помощью функции Untransfrom класса Wavelet. В результате получается измененная синяя составляющая, которая несет в себе код ЦВЗ. Далее синяя составляющая исходного изображения заменяется на измененную с помощью функции UpdateColorComponent класса DoubleImage.

Функция KIMextract класса JRKimAlgorithm осуществляет извлечение ЦВЗ с помощью исходного изображения. Ее алгоритм описан в пункте 2.1.2. Данная функция использует аналогичные вспомогательные функции как в функции KIMembed.

3.3 Анализ алгоритма

В этой главе будет приведен анализ алгоритма на быстродействие, эффективность и робастность.

3.3.1 Анализ быстродействия и эффективности

В первую очередь было исследовано быстродействие алгоритма. Внедрение ЦВЗ размером 100x100 пикселей в изображение высокого разрешения размером 2048x2048 пикселей заняло 2,474 секунд. Обработка меньших по размеру изображений происходит быстрее. Соответственно, на внедрение ЦВЗ большего размера уходит больше времени, однако специфика использования алгоритма не имеет особых требований к быстродействию программы, поэтому результат работы около 2 секунд будет считать успешным.

Размер встраиваемого ЦВЗ зависит не только от размера изображения, но и от его цветовой насыщенности. Например, для полностью белого (то есть максимально насыщенного по всем цветовым

составляющим) изображения размером 2048x2048 максимально возможный размер ЦВЗ 443x443. Но для темных изображений такого же размера максимально возможный размер ЦВЗ значительно меньше, что привело к созданию оптимизаций 3.4.2 и 3.4.3.

3.3.2 Анализ на робастность

Анализ на робастность подразумевал под собой изменение изображения, содержащего ЦВЗ внутри себя, и последующую попытку извлечь ЦВЗ, имея исходное изображение.

Попытки применить какое-либо преобразование, а затем сразу же попробовать извлечь ЦВЗ, провалились, потому что алгоритм, имея исходное изображение и поврежденное изображение, содержащее ЦВЗ, сопоставляет пиксели этих двух изображений, предполагая, что они соответствуют друг другу. Соответственно, необходимы определенные дополнительные действия, чтобы можно было оценить робастность данного алгоритма.

Одним из таких действий может быть программная надстройка, которая позволяет определять, какое именно преобразование было сделано, и преобразовать изображение в исходное состояние, насколько это возможно. Одна из оптимизаций данного алгоритма представляет собой возможность восстанавливать исходный размер обрезанного изображения, размещая часть изображения, которая не была утеряна, в нужное положение, чтобы была возможность извлечь ЦВЗ хотя бы частично. Разработка комплексного приложения, позволяющего определять преобразования и восстанавливать исходные изображения, требует гораздо больших трудозатрат.

Второй способ проверить алгоритм на робастность – это изначально вручную восстановить изображение, насколько это возможно, а затем попытаться извлечь ЦВЗ, как это бы было при наличии вышеописанной программной надстройки.

Опишем полученные результаты с использованием данного способа.

Алгоритм робастен к таким геометрическим преобразованиям, как поворот и зеркальное отображение, поскольку при таких преобразованиях информация, которую несут пиксели изображения, не теряется. И при обратном преобразовании мы получаем идентичное исходному изображение.

При кадрировании изображения извлекается только часть ЦВЗ. Причем наибольшую ценность представляет верхняя часть изображения,

поскольку в соответствии с алгоритмом, ЦВЗ встраивается последовательно в коэффициенты аппроксимации 3 уровня, затем в горизонтальные, вертикальные, диагональные коэффициенты детализации 3, 2, и 1 уровней, соответственно. Следовательно, согласно рисунку 4, встраивание начинается сверху изображения, и если ЦВЗ имеет относительно небольшой размер, то он может «уместиться» в верхней части, то есть нижняя может быть обрезана без потери битов ЦВЗ (рис. 6).



Рис. 6 Слева восстановленный ЦВЗ из кадрированного изображения, справа исходный ЦВЗ

Удаление строк и столбцов имеет схожий эффект с кадрированием. Только при этом утерянная часть ЦВЗ будет внутри, а не снаружи (рис.7).



Рис. 7 Слева восстановленный ЦВЗ из изображения с удаленными строками, справа исходный ЦВЗ

При масштабировании качество изображения не меняется, поэтому ЦВЗ восстанавливается идентичный исходному.

Алгоритм не робастен к сжатию, а также к любым цветовым коррекциям таким, как изменение яркости, контрастности, насыщенности. Пример извлечения ЦВЗ после цветокоррекции можно видеть на рис. 8.



Рис. 8 Извлечение ЦВЗ после цветокоррекции

3.4 Оптимизация алгоритма

Несовершенство работы алгоритма, описанное в пункте 3.3.1, и его неустойчивость к некоторым преобразованиям натолкнули на написание

нескольких оптимизаций существующего алгоритма. Мною предложено 5 способов оптимизации алгоритма.

3.4.1 Встраивание ЦВЗ в несколько цветовых компонент

Как было сказано в пункте 3.3.1 количество встраиваемой информации сильно зависит от насыщенности изображения. Поскольку алгоритм предполагает встраивание ЦВЗ только в синюю составляющую, нас интересует насыщенность синего цвета. Однако далеко не всегда изображение содержит много синего цвета. Например, фотография красных роз размером 1000x1000 пикселей по расчетам позволяет встроить ЦВЗ с максимальным размером 30x30 пикселей, что очень мало, потому что даже при кодировании 10-значного номера телефона QR-код получается размером 100x100. Его можно сделать меньше, но с потерей качества.

Соответственно, можно попробовать встраивать ЦВЗ не только в синюю составляющую. Несмотря на физиологические особенности зрительной системы человека, описанные в пункте 1.3.2, по опытам данный алгоритм дает менее заметный результат в изменении красной составляющей, а не зеленой. Следовательно, было принято решение встраивать ЦВЗ в следующем порядке: сначала в синюю составляющую, затем в красную и только потом в зеленую. Код добавленных функций приведен в приложении 3. Теперь в то же самое фото красных роз с использованием всех трех компонент можно встроить ЦВЗ 115x115 пикселей, что в 14,5 раз больше предыдущего результата.

3.4.2 Предварительный расчет максимально возможного и оптимального размера ЦВЗ

Несмотря на оптимизацию, описанную в предыдущем пункте, есть изображения, которые все равно не позволяют встроить большой объем информации. В конце концов, в изображение, полностью залитое черным цветом, что-либо встроить с использованием данного алгоритма, невозможно, поскольку для этого нужны значения пикселей отличные от нуля. Соответственно, чем темнее изображение, тем меньше оно вмещает.

Поэтому в алгоритм был добавлен расчет максимально возможного размера и оптимального размера ЦВЗ (код добавленных функций приведен в приложении И). Максимально возможный размер достигается при использовании всех трех цветовых составляющих. Оптимальный размер считается с учетом встраивания ЦВЗ только в синюю составляющую, как это было в оригинальном алгоритме. Размер считается оптимальным, поскольку изменения в красной и зеленой составляющих все же больше

заметны глазу и в некоторых случаях эти изменения видны невооруженным глазом, что противоречит первоначальной идее использования данного алгоритма для незаметной подписи изображения.

3.4.3 Вывод предупреждения о невозможности встроить ЦВЗ целиком

Данная оптимизация дополняет предыдущую. Дело в том, что исходный алгоритм встраивает ЦВЗ независимо от размера. Но при нехватке необходимых коэффициентов для встраивания он просто прекращает работу. В результате встраивается лишь часть ЦВЗ и информация восстанавливается не полностью. Поэтому была встроена предварительная проверка возможности встроить желаемый ЦВЗ и предупреждения пользователя в противном случае.

3.4.4 Алгоритм определения процента совпадения восстановленного ЦВЗ с исходным в случае потери данных

Как было выяснено в пункте 3.3, есть преобразования, к которым алгоритм робастен, есть те, которые меняют изображение так, что не остается возможности восстановить ЦВЗ, но есть также те, после которых ЦВЗ восстанавливается в виде, близком к первоначальному.

В таких случаях разумно сравнивать два ЦВЗ, исходный и полученный на предмет схожести и выдавать процент совпадения. Для этого был разработан два алгоритма сравнения. Первый сравнивает ЦВЗ побитово. Однако в случаях, когда ЦВЗ в виде QR-кода был получен лишь частично, а остальное заполнено белым, он учитывает белые пиксели, которые совпадают с белыми пикселями в той части исходного ЦВЗ, которая была утеряна в полученном. Процент совпадения получается в таком случае больше, чем он является на самом деле. Поэтому второй алгоритм ищет наибольшие вхождения части поврежденного ЦВЗ в исходном и учитывает только его. Но такой алгоритм даст меньший результат в случае, когда ЦВЗ восстановлено целиком, но с потерями в отдельных битах. Поэтому наилучшим решением является использовать оба этих алгоритма для получения более точного результата. Код обеих функций представлен в приложении К.

3.4.5 Восстановление изображения после кадрирования для дальнейшего извлечения ЦВЗ

Как было упомянуто ранее, одна из оптимизаций – это восстановление изображения после кадрирования. Если обрезать изображение сверху или по бокам, то его пиксели сдвинутся с их исходных позиций. Если при этом пытаться извлечь ЦВЗ из обрезанного

изображения, алгоритм будет сопоставлять друг другу разные пиксели. Соответственно, необходимо прежде восстановить размер и пиксели, несущие информацию «поставить на место» с теми же индексами, как в исходном.

Для этого был реализован алгоритм поиска индексов вхождения в изображение его части. Сложность в том, что ищем мы в исходном изображении часть измененного (с встроенным ЦВЗ), поэтому поиск осуществляется по зеленой компоненте, так как изменения в ней наименее вероятны. Также для уменьшения вероятности сравнения исходного пикселя с пикселем со встроенным битом ЦВЗ, что приведет к неверному результату поиска, сравнение начинается с нижней части изображения, так как встраивание начинается сверху. Сравнение продолжается до половины части измененного изображения. Это дает риск получения неверного результата поиска, в случае если в изображении есть идентичные повторяющиеся части, но уменьшает риск сравнения пикселей, в один из которых был встроен бит ЦВЗ. Поскольку программа предусмотрена для «реальных» изображений, то вероятность того, что в изображении будут две абсолютно одинаковые части, при этом обрезано оно будет именно так, что поиск вхождения части изображения в изображение даст неверный результат достаточно мала.

С указанными ограничениями алгоритма нельзя ожидать корректной работы в всех случаях, однако как было замечено вероятность неправильной работы довольно мала.

Заключение

Мною был реализован алгоритм И.Р. Ким по внедрению цифрового водяного знака в изображение с помощью вейвлет-преобразований. Алгоритм предназначен для незаметного встраивания информации в изображения в целях защиты авторских прав.

В ходе анализа эффективности работы алгоритма и его устойчивости к различным преобразованиям, было выяснено, что он соответствует заявленной цели, но требует доработки для использования в реальной жизни. В связи с этим были предложены и реализованы некоторые способы оптимизации существующего алгоритма для расширения его функционала.

Список литературы

1. Астафьева Н.М. Вейвлет-анализ: основы теории и примеры применения / УФН, 1996. т. 166, № 11. Стр. 1145-1170.
2. Добеши И. Десять лекций по вейвлетам / НИЦ «Регулярная и хаотическая динамика», 2001, 464 стр.
3. Шелухин О.И., Канаев С.Д. Стеганография. Алгоритмы и программная реализация / 2017, 592 стр.
4. Kim J.R., Moon Y.S. A robust wavelet-based digital watermarking using level-adaptive thresholding / Proceedings 1999 International Conference on Image Processing, 1999.
5. Поликар Р. Введение в вейвлет-преобразование / Автор перевода: Грибунин В.Г., 59 стр.
6. Никольский С.М. Курс математического анализа: Учебник для вузов.— 5-е изд., перераб.— М.: Физико-математическая литература, 2000, 592 с.
7. Файфер Л. А. Практическое применение вейвлет-преобразования для исследования нестационарных несинусоидальных сигналов и расчёта мощности / Молодой ученый. 2016, №28 Стр. 200-203.
8. Вейвлет-сжатие: <https://habr.com/>. М.: 2013. 1 веб-страница. URL: <https://habr.com/ru/post/168517/>

Код класса DoublePixel

```
public class DoublePixel
{
    public double Red { get; set; }
    public double Green { get; set; }
    public double Blue { get; set; }

    public DoublePixel()
    {
        Red = 0;
        Green = 0;
        Blue = 0;
    }

    public DoublePixel(DoublePixel p)
    {
        Red = p.Red;
        Green = p.Green;
        Blue = p.Blue;
    }

    public DoublePixel(double new_red, double new_green, double new_blue)
    {
        SetColor(new_red, new_green, new_blue);
    }

    public void SetColor (double new_red, double new_green, double new_blue)
    {
        Red = new_red;
        Green = new_green;
        Blue = new_blue;
    }

    public Color ToNormalColor()
    {
        int normalRed = (int)Red, normal_green = (int)Green, normal_blue = (int)Blue;
        if (normalRed > 255) normalRed = 255;
        if (normal_green > 255) normal_green = 255;
        if (normal_blue > 255) normal_blue = 255;
        if (normalRed < 0) normalRed = 0;
        if (normal_green < 0) normal_green = 0;
        if (normal_blue < 0) normal_blue = 0;
        return Color.FromArgb(normalRed, normal_green, normal_blue);
    }
}
```

Код класса DoubleImage

```

public class DoubleImage
{
    public int Width { get; private set; }
    public int Height { get; private set; }

    public enum ColorComponent
    {
        Red,
        Green,
        Blue
    }

    private DoublePixel[,] _pixels;

    public DoubleImage(int width, int height)
    {
        Width = width;
        Height = height;
        _pixels = new DoublePixel[Width, Height];
        for (int i = 0; i < Width; i++)
            for (int j = 0; j < Height; j++)
                _pixels[i, j] = new DoublePixel();
    }

    public DoubleImage(Bitmap BMImage)
    {
        Width = BMImage.Width;
        Height = BMImage.Height;
        _pixels = new DoublePixel[Height, Width];
        for (int i = 0; i < Height; i++)
            for (int j = 0; j < Width; j++)
            {
                Color c = BMImage.GetPixel(j, i);
                _pixels[i, j] = new DoublePixel(c.R, c.G, c.B);
            }
    }

    public DoublePixel GetPixel (int i, int j)
    {
        return _pixels[i,j];
    }

    public void SetPixel(int i, int j, double r, double g, double b)
    {
        _pixels[i, j].SetColor(r, g, b);
    }

    public double[,] GetColorComponent(ColorComponent component)
    {
        double[,] colorComponent = new double[this.Height, this.Width];
        switch (component)
        {
            case ColorComponent.Red:
                for (int i = 0; i < this.Height; i++)
                    for (int j = 0; j < this.Width; j++)

```

```

        colorComponent[i, j] = this._pixels[i, j].Red;
        break;

    case ColorComponent.Green:
        for (int i = 0; i < this.Height; i++)
            for (int j = 0; j < this.Width; j++)
                colorComponent[i, j] = this._pixels[i, j].Green;
        break;

    case ColorComponent.Blue:
        for (int i = 0; i < this.Height; i++)
            for (int j = 0; j < this.Width; j++)
                colorComponent[i, j] = this._pixels[i, j].Blue;
        break;
    }
    return colorComponent;
}

public DoubleImage UpdateColorComponent(ColorComponent component, double[, ]
colorComponent)
{
    DoubleImage updatedImage = this;
    switch (component)
    {
        case ColorComponent.Red:
            for (int i = 0; i < this.Height; i++)
            {
                for (int j = 0; j < this.Width; j++)
                {
                    DoublePixel initialPixel = this.GetPixel(i, j);
                    updatedImage.SetPixel(i, j, colorComponent[i, j], initialPixel.Green,
initialPixel.Blue);
                }
            }
            break;

        case ColorComponent.Green:
            for (int i = 0; i < this.Height; i++)
            {
                for (int j = 0; j < this.Width; j++)
                {
                    DoublePixel initialPixel = this.GetPixel(i, j);
                    updatedImage.SetPixel(i, j, initialPixel.Red, colorComponent[i, j],
initialPixel.Blue);
                }
            }
            break;

        case ColorComponent.Blue:

            for (int i = 0; i < this.Height; i++)
            {
                for (int j = 0; j < this.Width; j++)
                {
                    DoublePixel initialPixel = this.GetPixel(i, j);
                    updatedImage.SetPixel(i, j, initialPixel.Red, initialPixel.Green,
colorComponent[i, j]);
                }
            }
            break;
    }
    return updatedImage;
}

```

```

    }

    public Bitmap ToBitmap(double colorMult, double colorShift)
    {
        Bitmap BImage = new Bitmap(Width, Height);
        for (int i = 0; i < BImage.Width; i++)
            for (int j = 0; j < BImage.Height; j++)
            {
                DoublePixel P = new DoublePixel(_pixels[j, i]);
                P.Red = P.Red * colorMult + colorShift;
                P.Green = P.Green * colorMult + colorShift;
                P.Blue = P.Blue * colorMult + colorShift;
                BImage.SetPixel(i, j, P.ToNormalColor());
            }
        return BImage;
    }

    public DoubleImage Copy()
    {
        DoubleImage copy = new DoubleImage(Width, Height);
        for (int i = 0; i < Height; i++)
            for (int j = 0; j < Width; j++)
            {
                DoublePixel pixel = this._pixels[i, j];
                copy._pixels[i, j] = new DoublePixel(pixel.Red, pixel.Green, pixel.Blue);
            }
        return copy;
    }
}

```


Код класса Watermark

```

public class Watermark
{
    public int Length { get; private set; }

    private int[] _bits;

    public int this[int index]
    {
        get { return _bits[index]; }
        set { _bits[index] = value; }
    }
    public Watermark(int length)
    {
        Length = length;
        _bits = new int[length];
    }
    public Watermark(DoubleImage image) {
        Length = image.Width * image.Height;
        _bits = new int[Length];

        for(int i=0; i<image.Height; i++)
        {
            for(int j=0; j<image.Width; j++)
            {
                //black
                if (image.GetPixel(i, j).Blue == 0 && image.GetPixel(i, j).Red == 0 &&
image.GetPixel(i, j).Green == 0)
                    _bits[i * image.Width + j]=0;
                //white
                if (image.GetPixel(i, j).Blue == 255 && image.GetPixel(i, j).Red == 255 &&
image.GetPixel(i, j).Green == 255)
                    _bits[i * image.Width + j] = 1;
            }
        }
    }

    public Watermark Append(Watermark addWatermark, int p)
    {
        Watermark newWatermark = this;
        for (int i = p; i < this.Length; i++)
        {
            newWatermark[i] = addWatermark[i];
        }
        return newWatermark;
    }

    public DoubleImage ToDoubleImage(int width, int height)
    {
        DoubleImage image = new DoubleImage(width, height);
        for (int i = 0; i < image.Height; i++)
        {
            for (int j = 0; j < image.Width; j++)
            {

```

```

        int index = i * image.Width + j;

        //black
        if (this[index] == 0)
        {
            image.SetPixel(i, j, 0, 0, 0);
        }

        //white
        if (this[index] == 1)
            image.SetPixel(i, j, 255, 255, 255);
    }
}
return image;
}

public override string ToString()
{
    return string.Join("", _bits);
}
}

```

Код класса Haar

```

public static class Haar
{
    private static double[] c_low = new double[2] { 0.70710678118, 0.70710678118 };
    private static double[] c_high = new double[2] { 0.70710678118, -0.70710678118 };
};

    public static double[,] Transform(double[,] matrix, int variableWidth, int
variableHeight)
    {
        int actualWidth = matrix.GetLength(1);
        int actualHeight = matrix.GetLength(0);

        double[,] intermediateMatrix = new double[variableHeight, variableWidth];
        double[,] resultMatrix = new double[variableHeight, variableWidth];

        //Rows transformation
        for (int i = 0; i < variableHeight; i++)
        {
            for (int j = 0; j < variableWidth / 2; j++)
            {
                double transformNumberLow = 0, transformNumberHigh = 0;

                for (int d = 0; d < waveletOrder; d++)
                {
                    transformNumberLow += c_low[d] * matrix[i, (j * 2 + d)];
                    transformNumberHigh += c_high[d] * matrix[i, (j * 2 + d)];
                }

                intermediateMatrix[i, j] = transformNumberLow; //Low frequency component
                intermediateMatrix[i, variableWidth / 2 + j] = transformNumberHigh; //High
frequency component
            }
        }

        //Column transformation
        for (int i = 0; i < variableWidth; i++)
        {
            for (int j = 0; j < variableHeight / 2; j++)
            {
                double transformNumberLow = 0, transformNumberHigh = 0;

                for (int d = 0; d < waveletOrder; d++)
                {
                    transformNumberLow += c_low[d] * intermediateMatrix[(j * 2 + d), i];
                    transformNumberHigh += c_high[d] * intermediateMatrix[(j * 2 + d), i];
                }

                resultMatrix[j, i] = transformNumberLow; //Low frequency component
                resultMatrix[variableHeight / 2 + j, i] = transformNumberHigh; //High
frequency component
            }
        }

        if(variableWidth!= actualWidth)

```

```

        resultMatrix = FillRestMatrix(matrix, resultMatrix, variableWidth,
variableHeight);

        return resultMatrix;
    }

    public static double[,] Untransform(double[,] matrix, int variableWidth, int
variableHeight)
    {
        double[,] intermediateMatrix = new double[variableHeight, variableWidth];
        double[,] resultMatrix = new double[variableHeight, variableWidth];

        //Column transformation
        for (int i = 0; i < variableWidth; i++)
        {
            for (int j = 0; j < variableHeight / 2; j++)
            {
                double transformedNumberLow = matrix[j, i];
                double transformedNumberHigh = matrix[variableHeight / 2 + j, i];

                double initialNumberLow = (transformedNumberLow + transformedNumberHigh) *
c_low[0];
                double initialNumberHigh = (transformedNumberLow - transformedNumberHigh) *
c_high[0];

                intermediateMatrix[j * 2, i] = initialNumberLow; //Low frequency component
                intermediateMatrix[j * 2 + 1, i] = initialNumberHigh; //High frequency
component
            }
        }

        //Rows transformation
        for (int i = 0; i < variableHeight; i++)
        {
            for (int j = 0; j < variableWidth / 2; j++)
            {
                double transformedNumberLow = intermediateMatrix[i, j];
                double transformedNumberHigh = intermediateMatrix[i, variableWidth / 2 + j];

                double initialNumberLow = (transformedNumberLow + transformedNumberHigh) *
c_low[0];
                double initialNumberHigh = (transformedNumberLow - transformedNumberHigh) *
c_high[0];

                resultMatrix[i, j * 2] = initialNumberLow; //Low frequency component
                resultMatrix[i, j * 2 + 1] = initialNumberHigh; //High frequency component
            }
        }
        int actualWidth = matrix.GetLength(1);
        if (variableWidth != actualWidth)
            resultMatrix = FillRestMatrix(matrix, resultMatrix, variableWidth,
variableHeight);

        return resultMatrix;
    }

    private static double[,] FillRestMatrix(double[,] originalMatrix, double[,]
transfromMatrix, int variableWidth, int variableHeight)
    {
        double[,] resultMatrix = new double[originalMatrix.GetLength(0),
originalMatrix.GetLength(1)];

```

```

    for (int i = 0; i < variableHeight; i++)
    {
        for (int j = 0; j < variableWidth; j++)
            resultMatrix[i, j] = transfromMatrix[i, j];
        for (int j = variableWidth; j < originalMatrix.GetLength(1); j++)
            resultMatrix[i, j] = originalMatrix[i, j];
    }
    for (int i = variableHeight; i < originalMatrix.GetLength(0); i++)
        for (int j = 0; j < originalMatrix.GetLength(1); j++)
            resultMatrix[i, j] = originalMatrix[i, j];
    return resultMatrix;
}
}

```

Код класса Wavelet

```

public static class Wavelet
{
    public enum Coefficients
    {
        Approximation,
        Horizontal,
        Vertical,
        Diagonal
    }
    public static double[,] Transform(double[,] matrix, int decompositionLevel)
    {
        int width = matrix.GetLength(1);
        int height = matrix.GetLength(0);

        CheckDecompositionLevel(decompositionLevel, width, height);

        double[,] resultMatrix = matrix;
        for (int i = 1; i <= decompositionLevel; i++)
        {
            int variableWidth = width / i;
            int variableHeight = height / i;
            resultMatrix = Haar.Transform(resultMatrix, variableWidth, variableHeight);
        }

        return resultMatrix;
    }

    public static double[,] Untransform(double[,] matrix, int decompositionLevel)
    {
        int width = matrix.GetLength(1);
        int height = matrix.GetLength(0);

        CheckDecompositionLevel(decompositionLevel, width, height);

        double[,] resultMatrix = matrix;
        for (int i = decompositionLevel; i >= 1; i--)
        {
            int variableWidth = width / i;
            int variableHeight = height / i;
            resultMatrix = Haar.Untransform(resultMatrix, variableWidth, variableHeight);
        }

        return resultMatrix;
    }

    public static double[,] GetCoefficient(double[,] matrix, Coefficients
    typeOfCoef, int decompositionLevel)
    {
        int width = matrix.GetLength(1);
        int height = matrix.GetLength(0);

        CheckDecompositionLevel(decompositionLevel, width, height);

        int decompositionCoef = Convert.ToInt32(Math.Pow(2, decompositionLevel));
        Range range = GetRange(typeOfCoef, decompositionLevel, width, height);
    }
}

```

```

        double[,] resultCoef = new double[height / decompositionCoef, width /
decompositionCoef];

        for(int i=0; i<resultCoef.GetLength(0); i++)
        {
            for (int j = 0; j < resultCoef.GetLength(1); j++)
                resultCoef[i, j] = matrix[i+ range.Height.StartIndex, j +
range.Width.StartIndex];
        }
        return resultCoef;
    }

    public static double[,] SetCoefficient(double[,] matrix, Coefficients
typeOfCoef, int decompositionLevel, double[,] coefficients)
    {
        int width = matrix.GetLength(1);
        int height = matrix.GetLength(0);

        CheckDecompositionLevel(decompositionLevel, width, height);

        int decompositionCoef = Convert.ToInt32(Math.Pow(2, decompositionLevel));
        Range range = GetRange(typeOfCoef, decompositionLevel, width, height);

        int coefWidth = width / decompositionCoef;
        int coefHeight = height / decompositionCoef;

        for (int i = 0; i < coefHeight; i++)
        {
            for (int j = 0; j < coefWidth; j++)
                matrix[i + range.Height.StartIndex, j + range.Width.StartIndex] =
coefficients[i,j];
        }
        return matrix;
    }

    private static Range GetRange(Coefficients typeOfCoef, int decompositionLevel,
int width, int height)
    {
        int decompositionCoef = Convert.ToInt32(Math.Pow(2, decompositionLevel));
        Range range = new Range();

        switch (typeOfCoef)
        {
            case Coefficients.Approximation:
                range.Width.StartIndex = 0;
                range.Height.StartIndex = 0;
                break;

            case Coefficients.Horizontal:
                range.Width.StartIndex = width / decompositionCoef;
                range.Height.StartIndex = 0;
                break;

            case Coefficients.Vertical:
                range.Width.StartIndex = 0;
                range.Height.StartIndex = height / decompositionCoef;
                break;

            case Coefficients.Diagonal:
                range.Width.StartIndex = width / decompositionCoef;
                range.Height.StartIndex = height / decompositionCoef;

```

```

        break;

    default:
        range.Width.StartIndex = 0;
        range.Height.StartIndex = 0;
        break;
    }

    return range;
}

private static void CheckDecompositionLevel(int decompositionLevel, int width,
int height)
{
    int decompositionCoef = Convert.ToInt32(Math.Pow(2, decompositionLevel));
    if (width % decompositionCoef != 0 || height % decompositionCoef != 0)
        throw new Exception("Width and height should be divided by
2^decompositionLevel.");
}
}

```


Код класса JRKimAlgorithm

```

class JRKimAlgorithm
{
    private double coefApprox = 0.02;
    private double coef3Level = 0.1;
    private double coef2Level = 0.2;
    private double coef1Level = 0.4;
    private DoubleImage.ColorComponent mainColorComponent =
DoubleImage.ColorComponent.Blue;

    public DoubleImage KIMembed(DoubleImage image, Watermark initialWatermark)
    {
        // Get blue component from image
        double[,] blueComponent = image.GetColorComponent(mainColorComponent);

        // Transform blue component with wavelet transformation
        double[,] waveletTransformedComponent = Wavelet.Transform(blueComponent, 3);

        // Get coefficients of transformation
        double[,] approx3 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Approximation, 3);
        double[,] detailHor3 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 3);
        double[,] detailVert3 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Vertical, 3);
        double[,] detailDiag3 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Diagonal, 3);

        double[,] detailHor2 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 2);
        double[,] detailVert2 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Vertical, 2);
        double[,] detailDiag2 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Diagonal, 2);

        double[,] detailHor1 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 1);
        double[,] detailVert1 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Vertical, 1);
        double[,] detailDiag1 = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Diagonal, 1);

        // Get thresholds
        double threshold1 = Threshold.GetThreshold(detailHor1, detailVert1,
detailDiag1);
        double threshold2 = Threshold.GetThreshold(detailHor2, detailVert2,
detailDiag2);
        double threshold3 = Threshold.GetThreshold(detailHor3, detailVert3,
detailDiag3, approx3);

        Watermark watermark = TransformWatermark(initialWatermark);

        // Embed watermark into coefficients
        int p = 0;
        p = EmbedInCoef(coefApprox, ref approx3, threshold3, watermark, p);
    }
}

```

```

    p = EmbedInCoef(coef3Level, ref detailHor3, threshold3, watermark, p);
    p = EmbedInCoef(coef3Level, ref detailVert3, threshold3, watermark, p);
    p = EmbedInCoef(coef3Level, ref detailDiag3, threshold3, watermark, p);

    p = EmbedInCoef(coef2Level, ref detailHor2, threshold2, watermark, p);
    p = EmbedInCoef(coef2Level, ref detailVert2, threshold2, watermark, p);
    p = EmbedInCoef(coef2Level, ref detailDiag2, threshold2, watermark, p);

    p = EmbedInCoef(coef1Level, ref detailHor1, threshold1, watermark, p);
    p = EmbedInCoef(coef1Level, ref detailVert1, threshold1, watermark, p);
    p = EmbedInCoef(coef1Level, ref detailDiag1, threshold1, watermark, p);
    Console.WriteLine("p {0}", p);

    // Restore blue component from coefficients
    double[,] componentWithWatermark = new double[blueComponent.GetLength(0),
blueComponent.GetLength(1)];
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Approximation, 3, approx3);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Horizontal, 3, detailHor3);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Vertical, 3, detailVert3);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Diagonal, 3, detailDiag3);

    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Horizontal, 2, detailHor2);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Vertical, 2, detailVert2);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Diagonal, 2, detailDiag2);

    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Horizontal, 1, detailHor1);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Vertical, 1, detailVert1);
    componentWithWatermark = Wavelet.SetCoefficient(componentWithWatermark,
Wavelet.Coefficients.Diagonal, 1, detailDiag1);

    double[,] untrasformedComponent = Wavelet.Untransfrom(componentWithWatermark,
3);

    // Embed blue component into initial image
    image = image.UpdateColorComponent(mainColorComponent, untrasformedComponent);

    return image;
}

private int EmbedInCoef(double constCoefficient, ref double[,] coefficients,
double treshold, Watermark watermark, int p)
{
    Console.WriteLine("watermark length {0}", watermark.Length);
    for (int i = 0; i < coefficients.GetLength(0); i++)
    {
        if (p >= watermark.Length)
            break;
        for (int j = 0; j < coefficients.GetLength(1); j++)
        {
            if (p >= watermark.Length)
                break;
            if (coefficients[i, j] > treshold)

```

```

        {
            double coefficient = coefficients[i, j];
            coefficients[i, j] = coefficients[i, j] + constCoefficient * coefficients[i,
j] * watermark[p];
            p++;
        }
    }
}
return p;
}

```

```

public DoubleImage KIMextract(DoubleImage initialImage, DoubleImage
changedImage, int widthWatermark, int heightWatermark)
{
    // Get blue component from changed image
    double[,] blueComponentChanged =
changedImage.GetColorComponent(mainColorComponent);

    // Transfrom blue component of changed image with wavelet transformation
    double[,] waveletTransformedChangedComponent =
Wavelet.Transform(blueComponentChanged, 3);

    // Get coefficients of transformation
    double[,] approx3 = Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Approximation, 3);
    double[,] detailHor3 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Horizontal, 3);
    double[,] detailVert3 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Vertical, 3);
    double[,] detailDiag3 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Diagonal, 3);

    double[,] detailHor2 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Horizontal, 2);
    double[,] detailVert2 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Vertical, 2);
    double[,] detailDiag2 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Diagonal, 2);

    double[,] detailHor1 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Horizontal, 1);
    double[,] detailVert1 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Vertical, 1);
    double[,] detailDiag1 =
Wavelet.GetCoefficient(waveletTransformedChangedComponent,
Wavelet.Coefficients.Diagonal, 1);

    // Get blue component from initial image
    double[,] blueComponentInitial =
initialImage.GetColorComponent(mainColorComponent);

    // Transfrom blue component of initial image with wavelet transformation

```

```

    double[,] waveletTransformedComponent = Wavelet.Transform(blueComponentInitial,
3);

    // Get coefficients of transformation
    double[,] approx3Initial = Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Approximation, 3);
    double[,] detailHor3Initial =
Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 3);
    double[,] detailVert3Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Vertical,
3);
    double[,] detailDiag3Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Diagonal,
3);

    double[,] detailHor2Initial =
Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 2);
    double[,] detailVert2Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Vertical,
2);
    double[,] detailDiag2Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Diagonal,
2);

    double[,] detailHor1Initial =
Wavelet.GetCoefficient(waveletTransformedComponent,
Wavelet.Coefficients.Horizontal, 1);
    double[,] detailVert1Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Vertical,
1);
    double[,] detailDiag1Initial =
Wavelet.GetCoefficient(waveletTransformedComponent, Wavelet.Coefficients.Diagonal,
1);

    // Get thresholds from initial image
    double threshold1 = Threshold.GetThreshold(detailHor1Initial,
detailVert1Initial, detailDiag1Initial);
    double threshold2 = Threshold.GetThreshold(detailHor2Initial,
detailVert2Initial, detailDiag2Initial);
    double threshold3 = Threshold.GetThreshold(detailHor3Initial,
detailVert3Initial, detailDiag3Initial, approx3Initial);

    int watermarkSize = widthWatermark * heighthWatermark;
    Watermark watermark = new Watermark(watermarkSize);

    // Extract watermark from coefficients
    int p = 0;
    p = ExtractFromCoef(coefApprox, approx3Initial, approx3, threshold3, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef3Level, detailHor3Initial, detailHor3, threshold3, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef3Level, detailVert3Initial, detailVert3, threshold3, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef3Level, detailDiag3Initial, detailDiag3, threshold3, p,
watermarkSize, ref watermark);

    p = ExtractFromCoef(coef2Level, detailHor2Initial, detailHor2, threshold2, p,
watermarkSize, ref watermark);

```

```

    p = ExtractFromCoef(coef2Level, detailVert2Initial, detailVert2, threshold2, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef2Level, detailDiag2Initial, detailDiag2, threshold2, p,
watermarkSize, ref watermark);

    p = ExtractFromCoef(coef1Level, detailHor1Initial, detailHor1, threshold1, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef1Level, detailVert1Initial, detailVert1, threshold1, p,
watermarkSize, ref watermark);
    p = ExtractFromCoef(coef1Level, detailDiag1Initial, detailDiag1, threshold1, p,
watermarkSize, ref watermark);

    watermark = UnTransformWatermark(watermark);
    DoubleImage watermarkImage = watermark.ToDoubleImage(widthWatermark,
heightWatermark);
    return watermarkImage;
}

private int ExtractFromCoef(double constCoefficient, double[, ]
initialCoefficients, double[, ] changedCoefficients, double treshold, int p, int
watermarkSize, ref Watermark watermark)
{
    for (int i = 0; i < initialCoefficients.GetLength(0); i++)
    {
        if (p >= watermarkSize)
            break;
        for (int j = 0; j < initialCoefficients.GetLength(1); j++)
        {
            if (p >= watermarkSize)
                break;
            if (initialCoefficients[i, j] > treshold)
            {
                double initial = initialCoefficients[i, j];
                double changed = changedCoefficients[i, j];
                int watermarkBit = Convert.ToInt32(Math.Round((changed - initial) /
(constCoefficient * initial)));
                watermark[p] = watermarkBit;
                p++;
            }
        }
    }
    return p;
}

private Watermark TransformWatermark(Watermark watermark)
{
    Watermark newWatermark = new Watermark(watermark.Length);
    for (int i = 0; i < watermark.Length; i++)
    {
        if (watermark[i] == 0)
            newWatermark[i] = -1;
        else newWatermark[i] = 1;
    }
    return newWatermark;
}

private Watermark UnTransformWatermark(Watermark watermark)
{
    Watermark newWatermark = new Watermark(watermark.Length);
    for (int i = 0; i < watermark.Length; i++)

```

```
{
  if (watermark[i] == -1)
    newWatermark[i] = 0;
  else newWatermark[i] = 1;
}
return newWatermark;
}
}
```

Код класса Threshold

```

public static class Threshold
{
    public static double GetThreshold(double[,] horizontal, double[,] vertical,
double[,] diagonal)
    {
        double horMax = MaxMatrix(horizontal);
        double vertMax = MaxMatrix(vertical);
        double diagMax = MaxMatrix(diagonal);
        double max = MaxArray(new double[] { horMax, vertMax, diagMax });

        double treshold = Math.Pow(2, (Math.Log(max, 2) - 1));
        return treshold;
    }

    public static double GetThreshold(double[,] horizontal, double[,] vertical,
double[,] diagonal, double[,] approximation)
    {
        double horMax = MaxMatrix(horizontal);
        double vertMax = MaxMatrix(vertical);
        double diagMax = MaxMatrix(diagonal);
        double approxMax = MaxMatrix(approximation);
        double max = MaxArray(new double[] { horMax, vertMax, diagMax, approxMax });

        double treshold = Math.Pow(2, (Math.Log(max, 2) - 1));
        return treshold;
    }
    private static double MaxMatrix(double[,] matrix)
    {
        double max = matrix[0, 0];
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(1); j++)
            {
                var absNumber = Math.Abs(matrix[i, j]);
                if (absNumber > max)
                    max = absNumber;
            }
        }
        return max;
    }
    private static double MaxArray(double[] array)
    {
        double max = array[0];
        for (int i = 0; i < array.Length; i++)
            if (array[i] > max)
                max = array[i];
        return max;
    }
}

```

Код добавленных функций в класс JRKimAlgorithm

Встраивание в несколько цветовых составляющих:

```
public DoubleImage KIMembed(DoubleImage image, Watermark initialWatermark)
{
    DoubleImage copy = image.Copy();
    int maxWatermarkLength = GetOptimalWatermarkLength(copy);
    if (maxWatermarkLength < initialWatermark.Length)
        throw new Exception("Watermark length is too big to embed it in the image.");

    DoubleImage result = KIMembedComponent(image, initialWatermark,
DoubleImage.ColorComponent.Blue);
    if (p < initialWatermark.Length)
        result = KIMembedComponent(result, initialWatermark,
DoubleImage.ColorComponent.Red);
    if (p < initialWatermark.Length)
        result = KIMembedComponent(result, initialWatermark,
DoubleImage.ColorComponent.Green);
    p = 0;
    return result;
}
```

Извлечение ЦВЗ, которое было встроено в несколько цветовых составляющих:

```
public DoubleImage KIMextract(DoubleImage initialImage, DoubleImage changedImage,
int widthWatermark, int heightWatermark)
{
    Watermark watermark = KIMextractComponent(initialImage, changedImage,
widthWatermark, heightWatermark, DoubleImage.ColorComponent.Blue);
    if (p < widthWatermark * heightWatermark)
    {
        int startIndex = p;
        watermark = watermark.Append(KIMextractComponent(initialImage, changedImage,
widthWatermark, heightWatermark, DoubleImage.ColorComponent.Red), startIndex);
        //watermark = KIMextractComponent(initialImage, changedImage, widthWatermark,
heightWatermark, DoubleImage.ColorComponent.Red);
    }
    if (p < widthWatermark * heightWatermark)
    {
        int startIndex = p;
        watermark = watermark.Append(KIMextractComponent(initialImage, changedImage,
widthWatermark, heightWatermark, DoubleImage.ColorComponent.Green), startIndex);
    }
    p = 0;
    return watermark.ToDoubleImage(widthWatermark, heightWatermark);
}
```


Код добавленных функций в класс JRKimAlgorithm

Расчет максимально возможного размера ЦВЗ для данного изображения:

```
public int GetMaxWatermarkLength(DoubleImage image)
{
    Watermark watermark = new Watermark(image.Height * image.Width);
    KIMembedComponent(image, watermark, DoubleImage.ColorComponent.Blue);
    KIMembedComponent(image, watermark, DoubleImage.ColorComponent.Red);
    KIMembedComponent(image, watermark, DoubleImage.ColorComponent.Green);
    int maxWatermarkLength = p;
    p = 0;
    return maxWatermarkLength;
}

private static int MaxSquaredNumber(int number)
{
    double root = Math.Sqrt(number);
    return (int)root;
}
```

Расчет оптимального размера ЦВЗ для данного изображения:

```
public int GetOptimalWatermarkLength(DoubleImage image)
{
    Watermark watermark = new Watermark(image.Height * image.Width);
    KIMembedComponent(image, watermark, DoubleImage.ColorComponent.Blue);
    int result = p;
    p = 0;
    return result;
}
```

Код функций для расчета процента совпадения двух ЦВЗ

```

    public int CompareBitToBit(Watermark watermark)
    {
        int equalBits = 0;
        int comparisonLength = Math.Min(this.Length, watermark.Length);
        for (int i = 0; i < comparisonLength; i++)
        {
            if (watermark._bits[i] == this._bits[i])
                equalBits++;
        }
        return equalBits;
    }

    static string LongestCommonSubstring(string a, string b)
    {
        var n = a.Length;
        var m = b.Length;
        var array = new int[n, m];
        var maxValue = 0;
        var maxI = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (a[i] == b[j])
                {
                    array[i, j] = (i == 0 || j == 0)
                        ? 1
                        : array[i - 1, j - 1] + 1;
                    if (array[i, j] > maxValue)
                    {
                        maxValue = array[i, j];
                        maxI = i;
                    }
                }
            }
        }
        return a.Substring(maxI + 1 - maxValue, maxValue);
    }

```