Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №13

**ИССЛЕДОВАНИЕ МЕТОДОВ ТЕКСТОВОЙ СТЕГАНОГРАФИИ**

Студент: Сосновец М.И.

ФИТ 3 курс 4 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель:** изучение стеганографических методов встраивания/извлечения тайной информации с использованием электронного файла-контейнера текстового формата, приобретение практических навыков программной реализации методов (рассчитана на 4 часа аудиторных занятий: 2 часа – часть 1, 2 часа – часть 2)

**Задачи:**

1. Закрепить теоретические знания из области текстовой стеганографии, классификации, моделирования стеганосистем подобного вида и сущности основных методов.
2. Изучить основные алгоритмы встраивания/извлечения тайной информации на основе методов текстовой стеганографии, получить опыт практической реализации методов.
3. Разработать приложение для реализации алгоритмов встраивания/извлечения тайной информации на основе методов текстовой стеганографии.
4. Познакомиться с методиками оценки стеганографической стойкости методов.
5. Результаты выполнения лабораторной работы (отдельно по каждой из 2 частей) оформить в виде описания разработанного приложения (для части 2), методики выполнения экспериментов с использованием приложений и результатов экспериментов.

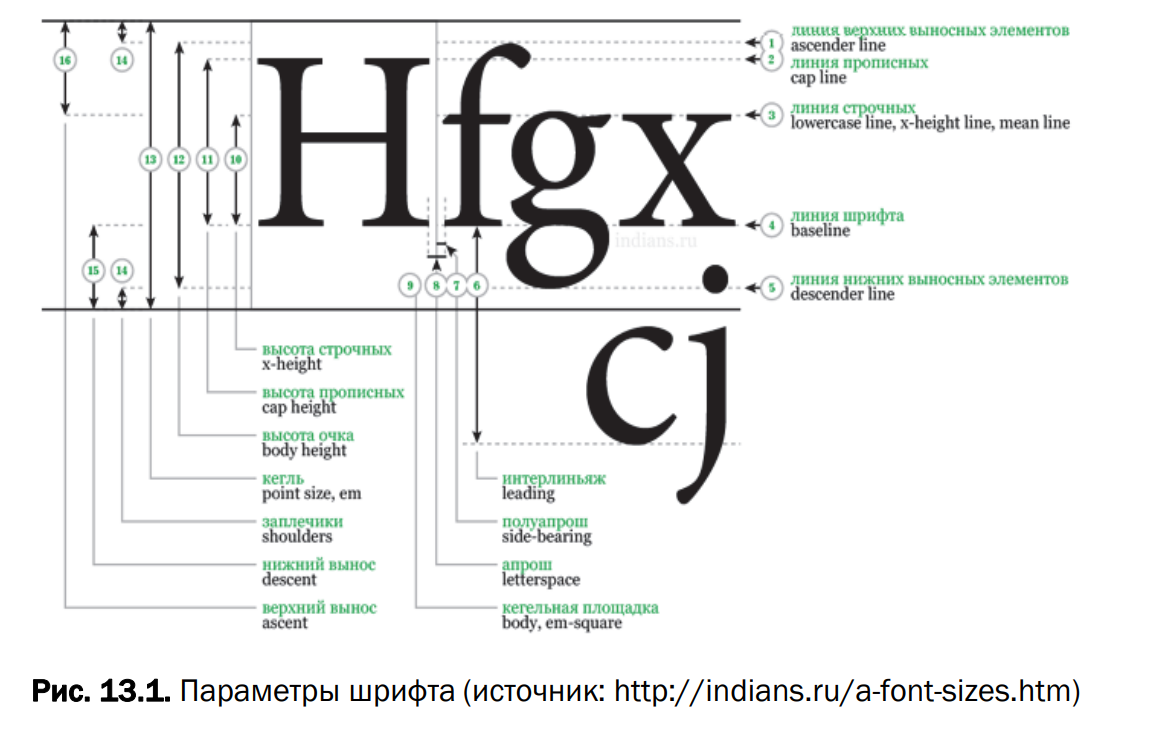
**Теоретические сведения**

Многообразие методов текстовой стеганографии подразделяется на синтаксические методы, которые не затрагивают семантику текстового сообщения, и лингвистические, которые основаны на эквивалентной трансформации текстовых файлов-контейнеров, сохраняющей смысловое содержание текста, его семантику (cм. [2, 52–56]).

Для понимания сущности некоторых из методов полезно познакомиться с важнейшими особенностями и параметрами использования стилей (в том числе пространственно-геометрическими параметрами шрифтов), на основе которых строится текстовый файлконтейнер. На рис. 13.1 показаны основные из параметров шрифта.

К синтаксическим методам компьютерной стеганографии, которые характеризуются сравнительно невысокой эффективностью (с точки зрения объема встраиваемой информации), относятся следующие (такие методы мы отнесем к числу базовых синтаксических методов):

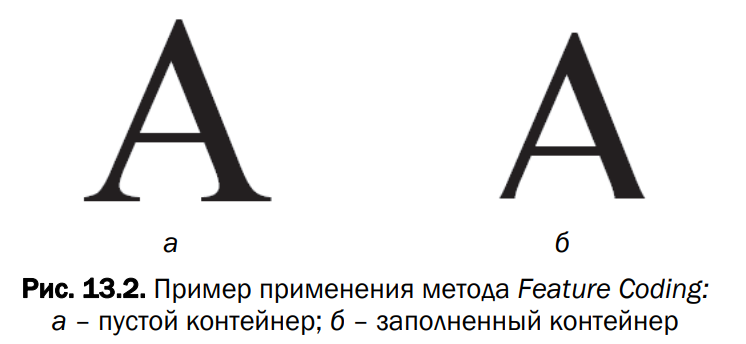
• изменение расстояния между строками электронного текста (Line-Shift Coding); называется методом изменения межстрочных интервалов; сущность заключается в том, что используется текст с различными межстрочными расстояниями: выделяется максимальное и минимальное расстояния между строками, позволяющее кодировать соответственно символы «1» и «0» осаждаемого сообщения;



• изменение расстояния между словами в одной строке электронного текста (Word-Shift Coding); суть метода состоит в том, что осаждение информации основано на модификации расстояния между словами текста-контейнера;

• изменение количества пробелов между словами (частный случай метода Word-Shift Coding); основан та том, что, например, чередование одинарного пробела и двойного (хх\_хх\_\_хх) кодирует «1», переход же с двойного пробела на одинарный кодирует «0» (хх\_\_хх\_хх);

• на основе внесения специфических изменений в шрифты, т. е.начертания отдельных букв (Feature Coding); заключается в изменении написания отдельных букв используемого стандартного шрифта: визуально заметны различные образы, соответствующие буквам с верхними (например, l, t, d) или нижними (например, a, g) выносными элементами (см. рис. 13.1); например, букву «А» можно модифицировать, незначительно укорачивая длинную нижнюю часть буквы (рис. 13.2);



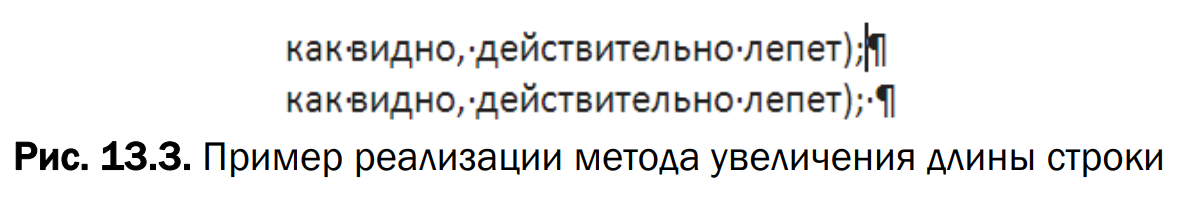
• изменение интервала табуляции; аналогичен вышеописанному методу изменения количества пробелов, только в этом случае меняется не количество пробелов, а соответственно расстояние между строками и интервал табуляции;

• Null Chipper (дословно – несуществующий, нулевой лепет); предполагает размещение тайной информации на установленных позициях слов или в определенных словах текста-контейнера, который, как правило, лишен логического смысла (как видно, действительно лепет);

• увеличение длины строки; предусматривает искусственное увеличение длины каждой строки за счет пробелов: например, нет пробела (определяется положением знака перехода на новую строку) – «0», один пробел – «1» (рис. 13.3);

• использование регистра букв; для обозначения бита секретного сообщения, представленного единицей, используется символ нижнего регистра, а нулем – верхнего (или наоборот);

• использование невидимых символов; знак «пробел» кодируется символом с кодом 32, но в тексте его можно заменить также символом, имеющим код 255 (или 0), который является «невидимым» и отображается как пробел.



Методы также легко применяются к любому тексту, независимо от его содержания, назначения и языка. Синтаксические системы стеганографии легко реализуются в программном коде, так как они полностью автоматические и не требуют вмешательства оператора. Однако синтаксические методы неустойчивы к форматированию текста (вспомним робастность систем на основе ЦВЗ), и поэтому информация может быть потеряна при простом применении иного стиля форматирования текста-контейнера, скрывающего в себе стегосообщение. К тому же с помощью синтаксических методов можно передать незначительное количество информации.

К числу основных лингвистических методов относятся [2, 52]:

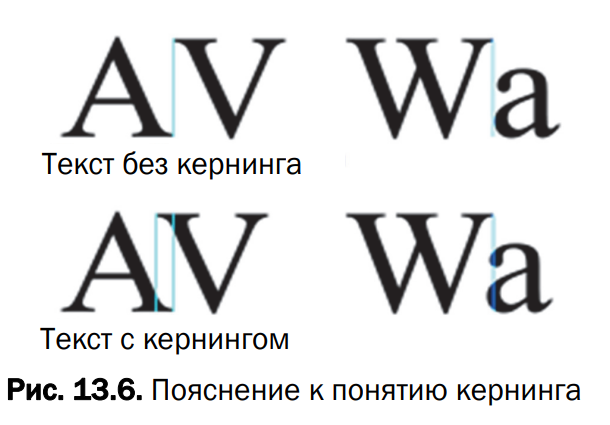
* метод синонимов; в качестве примера приведем подмножество синонимов: {«тайный», «секретный», «конфиденциальный», «доверительный»}. В приведенном подмножестве каждое слово имеет единственное одинаковое смысловое значение, что позволяет закодировать каждое слово своим уникальным кодом (т. е. выполнить операцию осаждения), например, «доверительный» – 00, «конфиденциальный» – 01, «секретный» – 10, «тайный» – 11. Подобное кодирование позволяет выбирать одно из четырех слов (как видим, они для удобства расположены по алфавиту) в зависимости от двух битов секретного сообщения. Отметим, что при этом, независимо какое из четырех слов будет выбрано, семантика сообщения не изменится. Очевидно, что при этом количество символов, соответствующих одному из синонимов используемого подмножества, зависит от общего числа элементов в подмножестве. Кроме того, обеим сторонам стеганосистемы должен быть известен общий алгоритм кодирования, т. е. один из ключей системы. Следует отметить, что в каждом подмножестве синонимов их упорядочивание должно выполняться по одному и тому же алгоритму и у отправителя сообщения, и у его получателя. В случае наличия слов с несколькими смысловыми значениями подобное кодирование оказывается невозможным. Также невозможно кодирование, если один из синонимов состоит из двух (или более) разделенных пробелом слов;
* метод переменной длины слова; основан на том, что длина слов в сообщении зависит от содержания секретного сообщения и способа кодирования слов: обычно одно слово текста-контейнера определенной длины кодирует два бита информации из стеганосообщения; например, слова текста длиной в 4 и 8 символов могут означать комбинацию битов «00», длиной в 5 и 9 – «01», 6 и 10 – «10», 7 и 11 букв – «11»; слова короче 4 и длиннее 11 букв можно вставлять где угодно для лексической и грамматической связки слов в предложении – программное приложение, которое декодирует принятое сообщение (извлекает сообщение из стеганоконтейнера), будет просто игнорировать их;
* метод первой буквы – программа-помощник в этом методе накладывает ограничение уже не на длину слова, а на первую (можно на вторую) букву; обычно одну и ту же комбинацию могут кодировать несколько букв, например, комбинацию «101» означают слова, начинающиеся с «А», «Г» или «Т;
* мимикрия; мимикрия генерирует осмысленный текст, используя синтаксис, описанный в Context Free Grammar (CFG), и встраивает информацию, выбирая из CFG определенные фразы и слова; грамматика CFG – это один из способов описания языка, который состоит из статических слов и фраз языка, а также узлов.

Метод на основе апроша. Апрош определяет расстояние между соседними символами текста. Фактически апрош состоит из двух таких расстояний – полуапрошей, являющихся как бы пространством, прилегающим к каждому из символов-соседей.

Идея метода [69, 70] заключается в следующем. Встраивание сообщения в контейнер может быть основано на модификации базового (устанавливаемого текстовым процессором по умолчанию) значения апроша ао, его изменением от базового до некоторого максимального аmax (или минимального аmin), которое зрительно не должно отличаться от стандартного. Такое изменение производится с определенным шагом (дискретно) Δаi, каждому значению которого присваивается определенный бит или определенная комбинация битов.

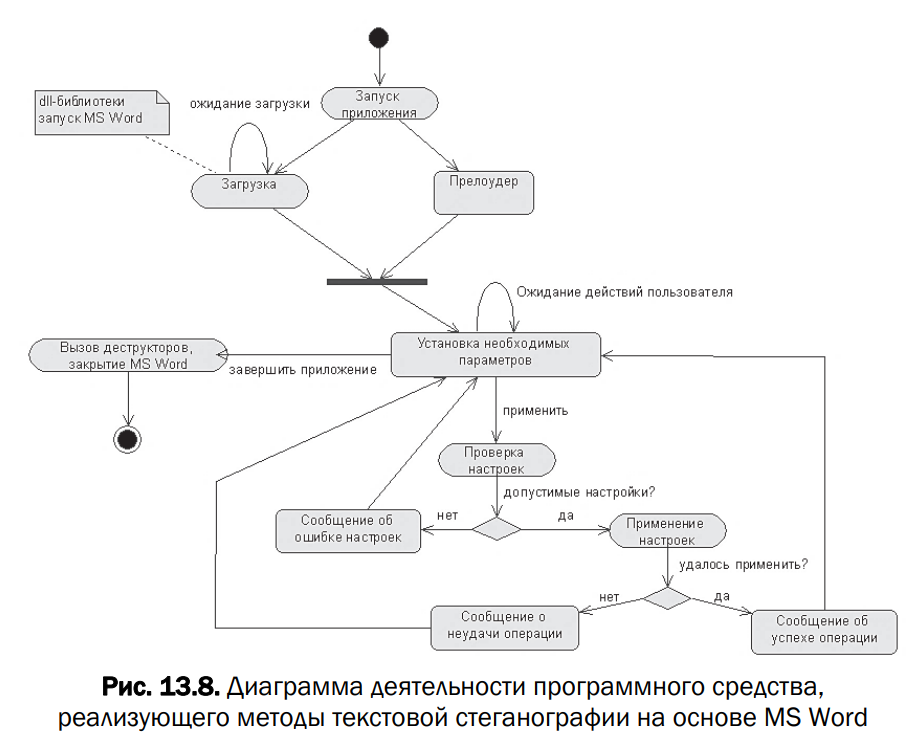
Метод на основе кернинга. В текстовых документах встречаются такие сочетания знаков, которые образовывают визуальные «дыры» либо «сгущения». Например, в текстах на основе кириллицы – это такие сочетания: «ГА», «TA», «ATA», «ЬТ» и т. п., на основе латиницы – «AY», «AV», «T;», «ff», а на основе греческого алфавита – «ΘΑ», «ΔΟ», «λκ» и др. Такие сочетания называются кернинговыми парами. Особенности «кернингования» приведены на рис. 13.6.

Под кернингом обычно понимается процесс изменения межсимвольного расстояние между отдельными парами символов или кернинговыми парами (именно фактор парности отличает кернинг от апроша).



Таким образом, технология кернинга, появившаяся в полиграфии после внедрения фотонабора (а затем и компьютерного набора), включает подбор межбуквенных интервалов для конкретных пар букв с целью улучшения внешнего вида и удобочитаемости текста. Такой избирательный подбор позволяет компенсировать неравномерности визуальной плотности текста, получаемой при использовании стандартных апрошей для каждой буквы.

Особенности программной реализации методов. Общий принцип работы некоторого условного приложения проиллюстрирован на диаграмме деятельности (рис. 13.8).



**Практическое задание**

Разработать авторское приложение, реализующее один из методов текстовой стеганографии на основе модификации пространственно-геометрических параметров текста-контейнера. Варианты заданий приведены в таблице. Дополнительные параметры согласуются с преподавателем.

|  |  |
| --- | --- |
| Вариант | Реализуемые методы |
| 12 | Изменение длины строки; модификация цвета |

В лабораторной работе было разработано приложение, которое реализует методы такие как изменение длины строки и модификация цвета.

Суть метода изменения длины строки заключается в том, что скрытие информации происходит за счет манипуляции с пробелами в конце строк текста. Каждая строка кодирует один бит информации:

* Пробел в конце строки = бит '1'
* Отсутствие пробела = бит '0'

Алгоритм работы:

1. Встраивание сообщения:

* Исходное сообщение преобразуется в бинарный вид
* Программа читает текстовый файл, разбивает его на строки
* Для каждого бита сообщения:
  + - 1. Берется соответствующая строка
      2. Если бит = '1', добавляется пробел в конец строки
      3. Если бит = '0', строка остается без изменений
* Измененные строки сохраняются в новый файл

1. Извлечение сообщения:

* Файл снова разбивается на строки
* Для каждой строки проверяется наличие пробела в конце
* Собирается бинарная последовательность
* Бинарный код преобразуется обратно в текст

Реализация метода изменения длины строки встраивания сообщения представлена в листинге 1.1.

|  |
| --- |
| async embed(*message*, *inputFile*, *outputFile*) {          try {              let text = await fs.readFile(*inputFile*, 'utf-8');              let lines = text.split('\n');              const nonEmptyLines = lines.map((*line*, *index*) => ({                  content: *line*.trim(),                  originalIndex: *index*              })).filter(*item* => *item*.content !== '');                       const binaryMessage = this.textToBinary(*message*);                console.log(`Бинарное сообщение: ${binaryMessage}`);              console.log(`Количество непустых строк: ${nonEmptyLines.length}`);                          if (binaryMessage.length > nonEmptyLines.length) {                  throw new Error(`Нужно ${binaryMessage.length} строк, но найдено только ${nonEmptyLines.length} непустых строк`);              }              for (let i = 0; i < binaryMessage.length; i++) {                const originalIndex = nonEmptyLines[i].originalIndex;                  const bit = binaryMessage[i];  *// Добавляем пробел в конец строки, если бит = 1*                  lines[originalIndex] = nonEmptyLines[i].content + (bit === '1' ? this.bitSeparator : '');              }              await fs.writeFile(*outputFile*, lines.join('\n'));              console.log(`Сообщение встроено в ${*outputFile*}`);              return true;          } catch (error) {              console.error(`Ошибка: ${error.message}`);              return false;          }} |

Листинг 1.1 - Реализация метода изменения длины строки встраивания сообщения

Результат работы метода изменения длины строки встраивания сообщения показан на рисунке 1.1.

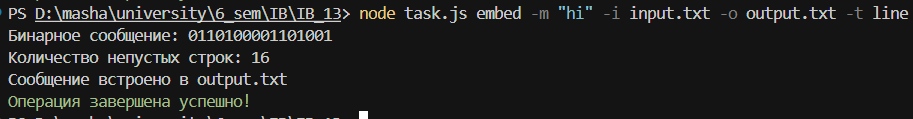


Рисунок 1.1 - Результат работы метода изменения длины строки встраивания сообщения

На рисунке 1.2 слева показан файл до работы приложения, а справа после работы приложения.

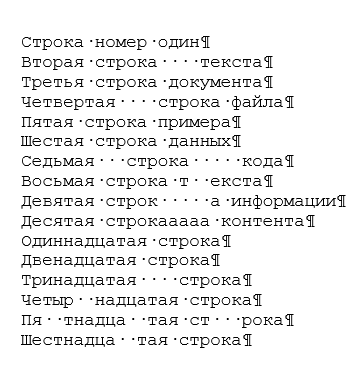
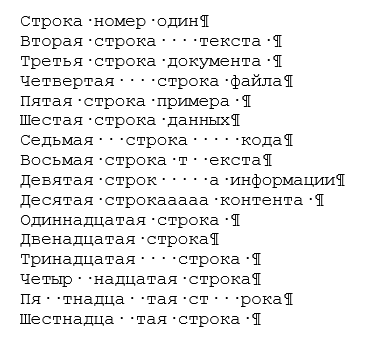
 

Рисунок 1.2 – Результат работы приложения

Реализация метода изменения длины строки извлечения сообщения представлена в листинге 1.2.

|  |
| --- |
| async extract(*inputFile*) {          try {              const text = await fs.readFile(*inputFile*, 'utf-8');              const lines = text.split('\n');              let binaryMessage = '';              for (const line of lines) {                  const trimmed = line.trim();                  if (trimmed !== '') {                      binaryMessage += line.endsWith(this.bitSeparator) ? '1' : '0';                  }              }              const message = this.binaryToText(binaryMessage);              console.log(`Извлеченное сообщение: ${message}`);              return message;          } catch (error) {              console.error(`Ошибка при извлечении: ${error.message}`);              return null; }} |

Листинг 1.2 - Реализация метода изменения длины строки извлечения сообщения

Результат работы метода изменения длины строки извлечения сообщения показан на рисунке 1.3.

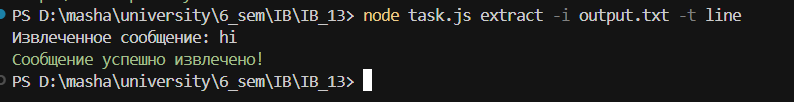


Рисунок 1.3 - Результат работы метода изменения длины строки извлечения сообщения

Особенности:

* Работает только с непустыми строками
* Требует, чтобы количество строк было не меньше длины бинарного сообщения
* Изменения почти незаметны визуально

Так же был реализован метод модификации цвета, суть которого заключается в том, что скрытие информации происходит за счет изменения цвета отдельных символов в тексте:

* Зеленый цвет символа = бит '0'
* Розовый цвет символа = бит '1'

Алгоритм работы:

1. Встраивание сообщения:

* Сообщение преобразуется в бинарный вид
* Текст разбивается на отдельные символы
* Бинарные биты равномерно распределяются по тексту
* Каждый выбранный символ оборачивается в HTML-тег с соответствующим цветом
* Весь текст сохраняется как HTML-документ

1. Извлечение сообщения:

* HTML-документ считывается как текст
* Программа ищет цветные теги в тексте
* По цвету тега определяет биты ('0' или '1')
* Собранная бинарная последовательность преобразуется в текст

Реализация метода модификации цвета встраивания сообщения представлена в листинге 1.3.

|  |
| --- |
| async embed(*message*, *inputFile*, *outputFile*) {          try {              let text = await fs.readFile(*inputFile*, 'utf-8');              const binaryMessage = this.textToBinary(*message*);              let textChars = text.split('');                if (binaryMessage.length > textChars.length) {                  throw new Error('Сообщение слишком длинное для данного файла');              }              const step = Math.max(1, Math.floor(textChars.length / binaryMessage.length));                for (let i = 0; i < binaryMessage.length; i++) {                  const pos = Math.min(i \* step, textChars.length - 1);                  const bit = binaryMessage[i];                  textChars[pos] = (bit === '1' ? this.color1 : this.color0) +                                  textChars[pos] +                                  this.resetColor;              }              const result = this.htmlHeader + textChars.join('') + this.htmlFooter;              await fs.writeFile(*outputFile*, result);              console.log(`Сообщение успешно встроено в ${*outputFile*}`);              console.log(`Откройте файл в браузере для просмотра цветов`);              return true;          } catch (error) {              console.error(`Ошибка при встраивании: ${error.message}`);              return false;}} |

Листинг 1.3 - Реализация метода модификации цвета встраивания сообщения

Результат работы метода модификации цвета встраивания сообщения показан на рисунке 1.4.

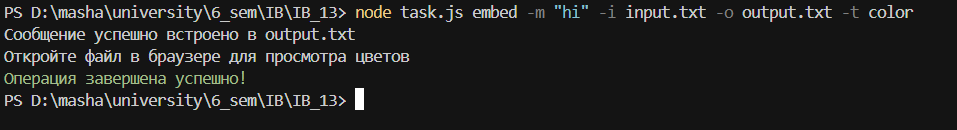


Рисунок 1.4 - Результат работы метода модификации цвета встраивания сообщения

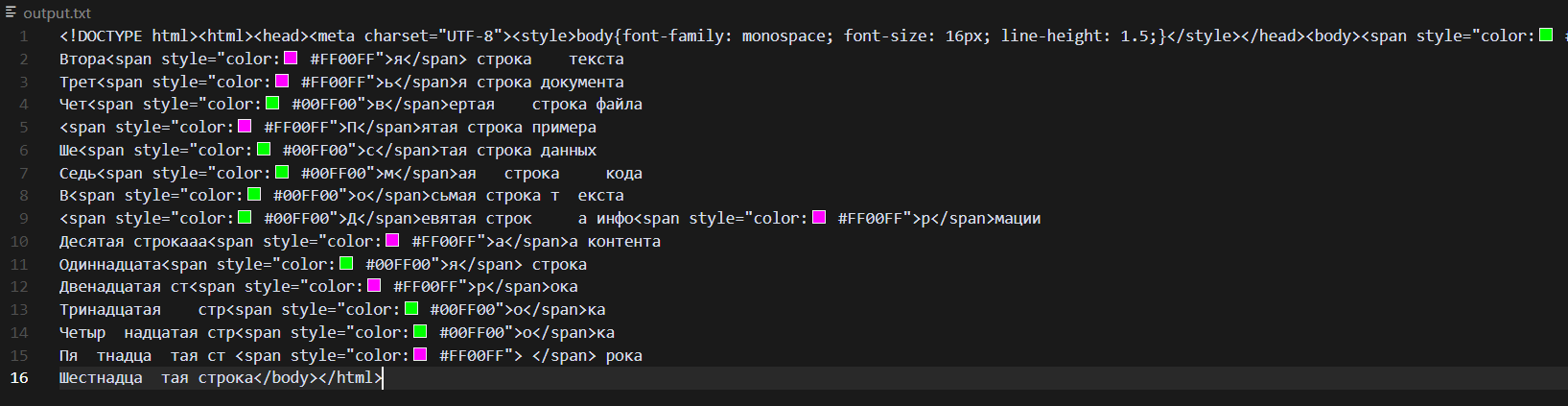


Рисунок 1.5 - Результат работы метода модификации цвета

Реализация метода модификации цвета извлечения сообщения представлена в листинге 1.4.

|  |
| --- |
| async extract(*inputFile*) {          try {              const text = await fs.readFile(*inputFile*, 'utf-8');              let binaryMessage = '';              let i = 0;              i = text.indexOf('<body>') + 6;              while (i < text.length) {                  if (text.startsWith(this.color0, i)) {                      binaryMessage += '0';                      i += this.color0.length;                  } else if (text.startsWith(this.color1, i)) {                      binaryMessage += '1';                      i += this.color1.length;                  } else if (text.startsWith(this.resetColor, i)) {                      i += this.resetColor.length;                  } else {                      i++;                  }              }              const message = this.binaryToText(binaryMessage);              console.log(`Извлеченное сообщение: ${message}`);              return message;          } catch (error) {              console.error(`Ошибка при извлечении: ${error.message}`);              return null;          }      } |

Листинг 1.4 - Реализация метода модификации цвета извлечения сообщения

Результат работы метода модификации цвета извлечения сообщения показан на рисунке 1.6.

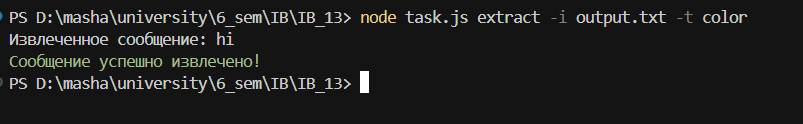


Рисунок 1.6 - Результат работы метода модификации цвета извлечения сообщения

Особенности:

* Позволяет скрыть больше информации, чем метод длины строки
* Изменения заметны только при просмотре исходного кода или в специальных редакторах

**Вывод**: в ходе работы были успешно изучены и реализованы два метода текстовой стеганографии: изменение длины строк и модификация цвета символов, позволяющие скрывать информацию в текстовых файлах. Разработанное консольное приложение продемонстрировало работоспособность обоих подходов, хотя метод изменения длины строк оказался более уязвим к форматированию. Полученные результаты подтвердили, что выбор оптимального метода зависит от требований к скрытности и устойчивости передаваемых данных.