

Цель работы:

Приобретение навыков встраивания и извлечения информации их видео-контейнера методом LSB с псевдослучайным изменением сообщения перед встраиванием и уменьшением допускаемых искажений.

Задачи работы:

1. Программно реализовать алгоритм встраивания и извлечения информации в видео-контейнер методом LSB с псевдослучайным изменением сообщения перед встраиванием и уменьшением допускаемых искажений.
2. Провести сравнение с использованием метода уменьшения допускаемых искажений и без использования данного метода.
3. Рассчитать параметр PSNR для исходного видео-контейнера и видео-контейнера со встроенной информацией.
4. Провести замеры времени с использованием алгоритма уменьшения допустимых искажений и без.
5. Провести атаки на стегоконтейнер.
6. Провести оценку целесообразности встраивания.

Теоретическая часть

Методы встраивания скрытой информации в цифровые видеопоследовательности и в особенности методы защиты видео от несанкционированного копирования при помощи встроенного ЦВЗ получили значительное распространение в последние два десятилетия. Большинство существующих алгоритмов встраивания ЦВЗ в видеопоследовательности основано на покадровом подходе к встраиванию ЦВЗ. Данный подход предполагает, что и при встраивании, и при извлечении ЦВЗ каждый кадр видеопоследовательности обрабатывается независимо от других кадров. Таким образом, данный подход рассматривает видеопоследовательность как упорядоченный набор изображений, в каждое из которых встраивается один и тот же ЦВЗ малого объёма (до 30–50 бит) [3].

Метод предварительного кодирования и встраивания информации в видео. Предлагаемый метод предназначен для встраивания ЦВЗ повышенной информационной ёмкости в цифровые видеопоследовательности. Метод встраивания скрытой информации является уязвимым к атаке с приближённым вычислением ЦВЗ в том случае, если встраиваемая информационная последовательность и ключ встраивания одинаковы для всех кадров видеопоследовательности [4].

Если один и тот же стеганографический ключ K использовался для встраивания ЦВЗ во все кадры видеопоследовательности, то становится возможной достаточно тривиальная атака, направленная на извлечение встроенного ЦВЗ. Атакующий, используя метод главных компонент (РСА) или схожий с ним метод независимых компонент (ICA) [2], может приближённо вычислить «шумоподобную» компоненту $D'(n,m)$, присутствующую в каждом кадре и кодирующую встроенный ЦВЗ. Далее атакующий может и обнаружить присутствие встроенной информации без знания ключа, и извлечь её полностью или частично, если известен способ кодирования. Данная атака становится

возможной именно в том случае, когда значение ЦВЗ (а значит, и соответствующее ему значение $D'(n,m)$) одинаково для всех кадров видеопоследовательности. Кроме того, с целью удаления или искажения встроенного ЦВЗ при минимальных искажениях видео атакующий может применить так называемую атаку с «приближённым вычислением ЦВЗ», используя приближённое значение $D'(n,m)$ [1].

Существующие алгоритмы встраивания ЦВЗ в видеопоследовательности для защиты от атак с приближённым вычислением ЦВЗ используют набор из различных ключей K_t . В то же время, как альтернатива схемам с динамически генерируемыми ключами для защиты от атаки с приближённым вычислением ЦВЗ, может быть использован подход, при котором ключ встраивания остаётся постоянным для всех кадров, но сам встраиваемый ЦВЗ изменяется некоторым псевдослучайным образом для каждого кадра видеопоследовательности. Такой подход обеспечивает отсутствие статичной шумоподобной составляющей во всех кадрах видеопоследовательности, тем самым обеспечивая стойкость ЦВЗ к упомянутой выше атаке [1].

Рассмотрим более подробно способ предварительного кодирования встраиваемой информации, основанный на указанном подходе.

Пусть H – последовательность бит встраиваемой информации, состоящая из L непересекающихся фрагментов длиной B_H бит каждый. j -ый бит i -го фрагмента H мы будем обозначать как $H_{i,j}$, $i \in [0, L - 1]$, $j \in [0, B_H - 1]$. В каждый кадр исходного видео-контейнера встраивается один из L независимых фрагментов S_0, S_1, \dots, S_{L-1} , каждый из которых состоит из $B = B_I + B_H$ бит, где $B_I = \text{round}[\log_2 L] + 1$. Фрагменты S_i формируются по следующему правилу: $S_i = i_0 i_1 \dots i_{B_I-1} H_{i,0} H_{i,1} \dots H_{i,B_H-1}$, где $i_0 i_1 \dots i_{B_I-1}$ – бинарное представление индекса i , а $H_{i,0} H_{i,1} \dots H_{i,B_H-1}$ – i -ый фрагмент H [1].

Например, если $L = 8$ и $B_H = 1$, то для $H_7 = 0$ S_7 будет равен 1110, где 111 – это двоичное представление числа 7. Далее в тексте $i_0 i_1 \dots i_{B_I-1}$ – будет именоваться *индексной частью* S , а $H_{i,0} H_{i,1} \dots H_{i,B_H-1}$ – *информационной частью*.

Далее при встраивании информации для каждого кадра псевдослучайным образом выбирается один фрагмент из набора S_0, S_1, \dots, S_{L-1} . После этого выбранный фрагмент встраивается в изображение-кадр любым алгоритмом встраивания информации в цифровые изображения. Единственным требованием к используемому алгоритму в данном случае является возможность встраивания и слепого (без знания исходного фрагмента ЦВЗ или его части) извлечения не менее чем B бит информации.

Таким образом, предлагаемый подход позволяет избежать статической или периодически повторяющейся структуры встраиваемой информации, не требуя динамически изменяемого ключа встраивания. Используя структуру фрагмента ЦВЗ, указанную в декодере ЦВЗ может извлечь исходную информационную последовательность H , не используя никакую дополнительную информацию об исходной нумерации кадров. Фактически предложенная избыточная структура фрагмента позволяет сделать ЦВЗ устойчивым как к преднамеренным, так и к случайным (например, связанным со сбоями передающего оборудования) ошибкам потери синхронизации [1].

Встречаются видео-контейнеры, для видеофрагментов которых способы маскирования шумоподобной компоненты не способны обеспечить низкую визуальную различимость встроенной информации. К таким видеопоследовательностям относятся фрагменты, в которых большая часть кадра имеет постоянную яркость. Это могут быть пустые фрагменты (затемнения или осветления экрана между сценами), экраны с титрами, сцены на постоянном фоне и пр. Для таких фрагментов встраивание информации не производится [1].

Практическая часть

Для программной реализации алгоритма встраивания и извлечения информации в видео-контейнер методом LSB с псевдослучайным изменением сообщения перед встраиванием и уменьшением допускаемых искажений был выбран язык программирования Python 3.8. Далее мы разберем алгоритм встраивания информации по этапам.

Этап 1. Кодирование исходного сообщения в двоичную строку.

Преобразование исходного сообщения в двоичную строку происходит следующим алгоритмом: каждый символ исходного сообщения при помощи таблицы символов Unicode преобразуется в десятичное число (номер символа), затем из данного числа вычитается число 1024 для уменьшения количества встраиваемых символов. В итоге, каждый символ исходного сообщения (состоящего из кириллических символов и пробелов) кодируется со стандартизированной размерностью в 7 бит. В конец двоичной строки добавляются 7 нулей как символ окончания строки. На выходе алгоритма получаем двоичную строку для дальнейшего преобразования.

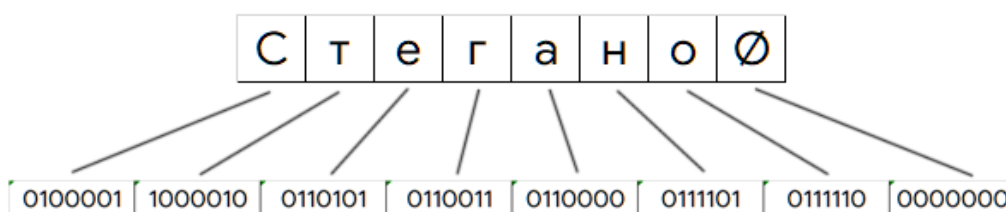


Рисунок 1. Демонстрация кодирования исходного сообщения.

Этап 2. Псевдослучайное изменение фрагментов сообщения.

На данном этапе происходит преобразование двоичной строки в список псевдослучайно измененных сообщений для встраивания, причем, количество сообщений будет зависеть от размера встраиваемой области. Псевдо-изменение сообщений происходит по следующему алгоритму (теоретическое описание

приведено в теоретической части): двоичная строка, полученная после кодирования, разбивается на некоторое количество фрагментов (количество зависит от размеров видео-контейнера, от указанных размеров области для встраивания, и от длины исходного сообщения), затем, к каждому фрагменту слева и справа добавляется номер данного фрагмента в двоичном виде (длина добавляемой части одинакова для всех фрагментов и зависит от количества фрагментов). Добавление номера фрагмента позволяет в дальнейшем проводить встраивание в любой кадр видео-контейнера, вне зависимости от изначального порядка расположения фрагментов. Добавление номера фрагмента и слева, и справа позволяет в дальнейшем исключить ошибки извлечения, связанные со случайным совпадением значений битов в заданных позициях для встраивания.



Рисунок 2. Демонстрация псевдослучайного изменения фрагментов.

Этап 3. Генерация списка позиций для встраивания фрагментов.

Для встраивания в кадры видео-контейнера была выбрана область по краям кадра, так называемая «рамка» для встраивания. Рамка состоит из 4 частей, верхней, нижней, левой и правой полос. Ширина верхней и нижней, а также левой и правой полос может быть настроена вручную в процентах от общего

размера кадра. После извлечения координат пикселей, содержащихся в «рамке», происходит случайная выборка координат пикселей. Псевдослучайность выборки задается «случайным зерном» или seed – число, используемое для инициализации генератора псевдослучайных чисел. Данное число может служить «закрытым ключом» для встраивания и извлечения сообщений.



Рисунок 3. Демонстрация сгенерированных позиций для встраивания.

Этап 4. Алгоритм уменьшения допускаемых искажений.

На данном этапе происходит процесс, позволяющий уменьшить допускаемые искажения при встраивании сообщения. Для его работы необходимы следующие данные: список фрагментов для встраивания, список позиций для встраивания и список кадров из видео-контейнера (извлечение кадров разберем далее). Алгоритм заключается в выборке таких кадров, встраивание в которые приведет к наименьшему количеству измененных бит в кадре. Алгоритм работает следующим образом: каждый фрагмент из списка фрагментов по очереди независимо встраивается в каждый кадр видео-контейнера, во время встраивания происходит замер количества измененных бит

для каждого фрагмента и каждого кадра. В итоге мы получаем таблицы для каждого кадра, содержащие номер фрагмента и количества измененных бит при встраивании данного фрагмента. Затем, происходит выборка лучших значений (наименьших изменений) для каждого кадра и определяется, какой конкретно фрагмент лучше встроить в данный кадр, так происходит со всеми кадрами. В итоге мы получаем список формата (номер кадра, номер фрагмента).

Давайте посмотрим, насколько эффективен данный способ и сколько бит из исходного видео-контейнера нам удастся «спасти». Встроим 2500 символов в видео-контейнер без применения и с применением алгоритма:

Без применения алгоритма		С применением алгоритма	
№ кадра	Кол-во изменений	№ кадра	Кол-во изменений
38	2132	93	1463
31	1853	199	1494
162	1729	119	1468
1	1839	49	1483
34	1825	181	1480
6	2687	167	1497
Всего	12065	Всего	8885
"Спасённых" бит:		26%	

Рисунок 4. Демонстрация результатов уменьшения допускаемых искажений.

Из результатов испытания видно, что 26% бит из общего числа замен не были изменены. То есть количество измененных бит в видео-контейнере будет на ~26% меньше.

Этап 5. Извлечение кадров из видео-контейнера.

Для работы с видео-контейнерами был выбран модуль для Python 3.8 под названием OpenCV. Это библиотека компьютерного зрения и машинного обучения с открытым исходным кодом. Однако, для данной работы нам понадобился лишь частичный функционал данной библиотеки.

Извлечение кадров происходит стандартными методами библиотеки OpenCV, и алгоритм действий таков: методом VideoCapture() происходит открытие видео-контейнера по указанному пути, методом read() происходит чтение первого кадра, затем циклически извлекаются все кадры видео-контейнера и объединяются в один список.

Этап 6. Встраивание фрагментов в кадры видео-контейнера.

Для встраивания методом LSB нам понадобятся следующие данные: список псевдослучайно измененных фрагментов, список позиций для встраивания, список кадров из видео-контейнера и список наилучших кадров для встраивания. Циклически перебирая фрагменты, биты во фрагментах, координаты позиций для встраивания и кадры для встраивания, происходит встраивание бит в наименее значимый 8-ой бит значения яркости синего цвета пикселя. Синий цвет был выбран из-за того, что человеческий глаз наименее чувствителен к синему цвету.

Этап 7. Формирование видео-контейнера и его сохранение.

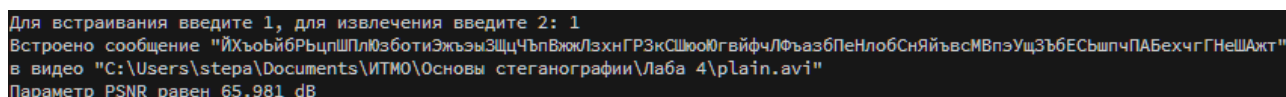
Для формирования видео-контейнера используется ранее упомянутая библиотека OpenCV. Методом VideoWriter() создается поток с указанным видео-кодеком (в нашем случае FFV1, об этом далее), количеством кадров в секунду и размером видео-контейнера. Затем, методом write() циклично записываются кадры в виде списка значений яркости пикселей в видео-поток. В конце, видео-контейнер сохраняется по указанному пути.

Теперь разберемся, с какими видеокодеками можно работать в случае использования метода LSB для встраивания. Кодек – программа, способная выполнять преобразование данных или сигнала для хранения, передачи или шифрования потока данных. В кодеках могут использоваться два вида сжатия данных: сжатие с потерями и сжатие без потерь. Сжатие с потерями существенно уменьшает объём данных для хранения или передачи, но приводит к ухудшению

качества звука или видео при воспроизведении, так как теряются наименее значимые значения пикселей или частот. В нашем случае данный тип сжатия не подходит, так как метод LSB подразумевает встраивание в наименее значимые биты. В нашей работе могут быть использованы только кодеки сжатия без потерь. Это методы сжатия, при использовании которых закодированные данные однозначно могут быть восстановлены с точностью до бита или пикселя. При этом оригинальные данные полностью восстанавливаются из сжатого состояния. В данной работе был использован видеокодек FFV1, относящийся к семейству FFmpeg, а в частности библиотеке libavcodec.

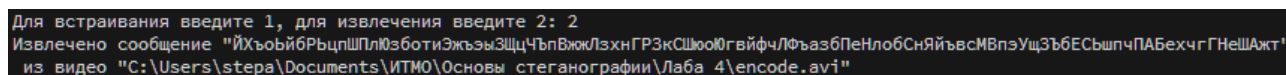
Демонстрация работы программы.

В качестве видео-контейнера был взят видеоролик формата avi со следующими характеристиками: ширина 160 пикселей, высота 120 пикселей, продолжительность 10 секунд, 25 кадров в секунду, формат RGB.



```
Для встраивания введите 1, для извлечения введите 2: 1
Встроено сообщение "ЙХъобйбРьцпшПлюзботиЭжъэЗщцЧъпВжжЛзхнГРЗкШююЮгвйфчЛФъазбПеНлобСняйъвсМВпэУщЗъбЕСьшпчПАБехчгГНеШАжт"
в видео "C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 4\plain.avi"
Параметр PSNR равен 65.981 dB
```

Рисунок 5. Демонстрация работы программы при встраивании сообщения.



```
Для встраивания введите 1, для извлечения введите 2: 2
Извлечено сообщение "ЙХъобйбРьцпшПлюзботиЭжъэЗщцЧъпВжжЛзхнГРЗкШююЮгвйфчЛФъазбПеНлобСняйъвсМВпэУщЗъбЕСьшпчПАБехчгГНеШАжт"
из видео "C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 4\encode.avi"
```

Рисунок 6. Демонстрация работы программы при извлечении сообщения.

Как мы видим, сообщение извлекается в 100% объеме, без ошибок.

Расчет параметра PSNR.

Так как в каждый кадр встраивается одинаковое количество информации, то разницы в значениях параметра PSNR для разных кадров не будет. PSNR будет оставаться на одном уровне при разном количестве встраиваемой информации. Однако, значение PSNR будет меняться при изменении области для встраивания.

Проведем исследование, изменяя область кадра для встраивания от 1% до 100% и посмотрим, как изменится параметр PSNR.

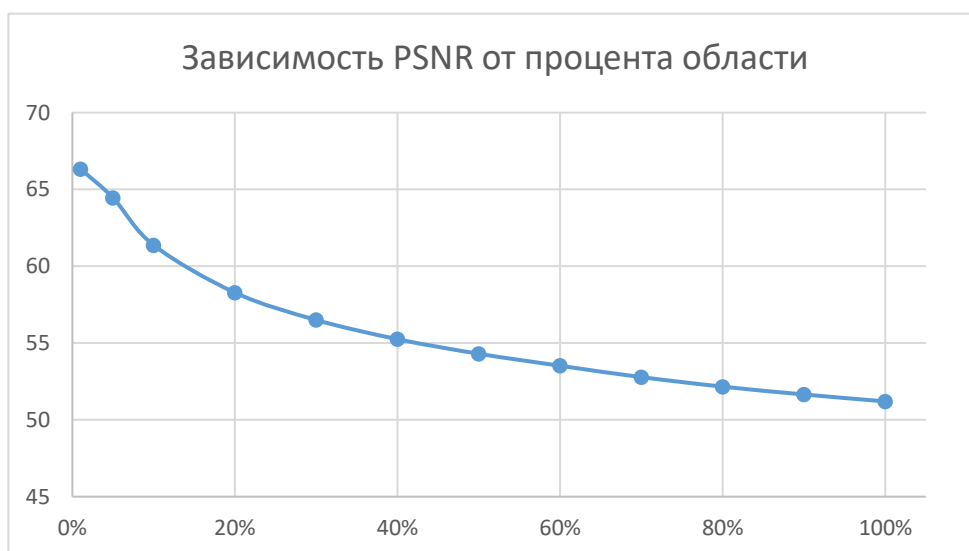


Рисунок 7. Зависимость PSNR от процента области для встраивания.

График зависимости параметра PSNR от процента области для встраивания получился постоянно убывающим, напоминающим убывающий экспоненциальный график. Это говорит о том, что чем больше процент области для встраивания, тем больше шума наблюдается на изображении, так как параметр PSNR есть отношение максимально возможного сигнала к уровню шума.

Проведение атаки методом сжатия видео-контейнера.

Для проведения этой атаки воспользуемся главным свойством видеокодеков сжатия с потерями, а именно, заменой/редактированием наименее значимых значений, в которых мы и скрываем информацию.

Для этого встроим в видео-контейнер сообщение «Атака методом сжатия видеоконтейнера с применением видеокодека сжатия с потерями», и на финальном шаге сборки видеоконтейнера применим к нему не кодек сжатия без потерь, а кодек сжатия с потерями. Будем использовать популярный кодек сжатия с потерями Motion JPEG. Посмотрим какое сообщение извлечется после проведения атаки:

```
Для встраивания введите 1, для извлечения введите 2: 1
Встроено сообщение "Атака методом сжатия видеоконтейнера с применением видеокодека сжатия с потерями" в видео
```

Рисунок 8. Атака методом сжатия, встраивание.

```
Для встраивания введите 1, для извлечения введите 2: 2
Извлечено сообщение "bхuжbKBrFf" из видео "C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 4\encode.avi"
```

Рисунок 9. Атака методом сжатия, извлечение.

Как мы видим, извлечь встроенное сообщение не удалось из-за особенности работы кодека сжатия с потерями, который изменил наименее значимые биты значений яркости пикселей.

Для проверки, что видеокодек сработал корректно, сравним размер исходного видео-контейнера и полученного видео-контейнера:

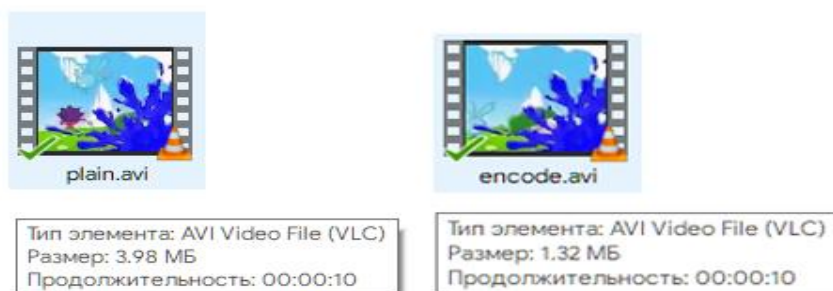


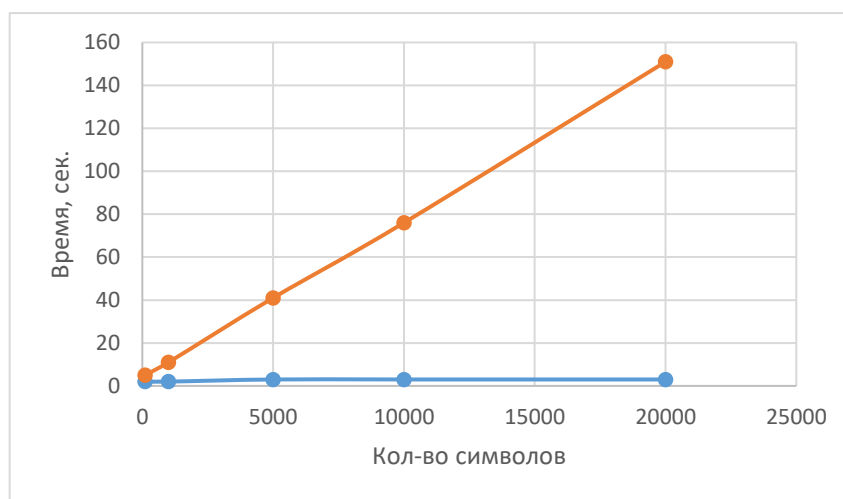
Рисунок 10. Атака методом сжатия, сравнение файлов.

Сравнение времени работы программы без применения алгоритма уменьшения допускаемых искажений и с применением.

Проведем сравнение времени работы программы без применения алгоритма уменьшения допускаемых искажений и с применением. Для этого, в случае без использования алгоритма, будем встраивать фрагменты сообщения в кадры по порядку, пропустив функцию выбора наилучших кадров для встраивания.

Встроим сообщение длиной 100 символов в видео-контейнер и сравним время выполнения программ: без использования алгоритма: 2 секунды, с использованием алгоритма: 5 секунд.

Как мы видим время выполнения программы значительно увеличивается. Теперь проведем замер времени при встраивании 100, 1000, 5000, 10000 и 20000 символов.



Видно, что при больших объемах встраивания время выполнения программы стремительно увеличивается. Это объясняется многочисленными операциями, выполняемыми при работе алгоритма. Алгоритм целесообразно использовать в случае необходимости встроить небольшой объем информации, либо необходимости встроить большой объем информации с наименьшими последствиями для качества видео-контейнера.

Оценка целесообразности встраивания.

Так как встраивание производилось только в последний бит значения интенсивности субпикселя, то визуально данные шумы незаметны. Уязвимостью данного метода являются алгоритмы сжатия с потерями. При таком алгоритме встраивания при 100% области встраивания в кадр в видеоролик размером 160x120 пикселей, длиной 10 секунд и 25 кадров в секунду можно встроить 4.800.000 бит информации, со значением PSNR в ~50 dB. Данный способ является целесообразным к использованию из-за: большого объема встраивания, 100% правильности извлечения, хорошего значения PSNR, незаметности визуальных искажений. Алгоритм уменьшения допускаемых искажений целесообразно использовать в случае необходимости встроить небольшой объем информации, либо необходимости встроить большой объем информации с наименьшими последствиями для качества видео-контейнера.

Вывод

В результате работы над программной реализацией метода встраивания и извлечения информации были приобретены навыки встраивания и извлечения информации в видео-контейнер методом LSB с псевдослучайным изменением сообщения перед встраиванием и уменьшением допускаемых искажений.

Данный метод является довольно целесообразным к использованию из-за большого объема встраивания, 100% правильности извлечения, хорошего значения PSNR, незаметности визуальных искажений. Алгоритм уменьшения допускаемых искажений позволяет уменьшить количество изменений значений наименее значимых битов на ~25% благодаря анализу и выборке подходящих кадров для встраивания. Псевдослучайное изменение сообщений перед встраиванием позволяет встраивать фрагменты в любые кадры, вне зависимости от их изначального порядка, а также позволяет избежать статической или периодически повторяющейся структуры встраиваемой информации, не требуя динамически изменяемого ключа встраивания.

Однако, метод обладает уязвимостью к алгоритмам сжатия с потерями. Что делает невозможным его использование для передачи сообщения в видео-контейнере через каналы связи, использующие алгоритмы сжатия с потерями, по типу социальных сетей и видео-хостингов.

Список литературы

1. «Метод встраивания информации в видео, стойкий к ошибкам потери синхронизации. Метод предварительного кодирования и встраивания информации в видео» // Митекин В.А., Федосеев В.А // Институт систем обработки изображений РАН, Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет) (СГАУ) // Журнал Компьютерная оптика 2014г. Том 38 №3 564с.-573с.
2. «Алгоритмы встраивания цифровых водяных знаков в растровые изображения» // Семёнов К.П., Зайцев П.В. // Научный журнал «Информационная безопасность регионов» 2012г. 46с.-50с.
3. «Temporal synchronization in video watermarking» // E.T. Lin, E.J. Delp // IEEE Transactions on Signal Processing 2004 г. 3007с.-3022с.
4. «Spatial synchronization using watermark key structure» // E.T. Lin, E.J. Delp // IEEE Transactions on Signal Processing 2004 г. 536с.-547с.
5. «OpenCV Python Documentation» // Электронный ресурс. Доступ открытый. URL: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html (дата обращения: 25.05.2020)
6. «Сравнение видеокодеков без потерь» // Ватолин Д., Гришин С. // Московский государственный университет имени М.В.Ломоносова, Лаборатория Графики и Медиа 2004г. 8с.-14с.

Приложение 1. Листинг программного кода.

```
import math
import random
import numpy as np
import cv2

plain_path = r'C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 4\plain.avi'
encode_path = r'C:\Users\stepa\Documents\ИТМО\Основы стеганографии\Лаба 4\encode.avi'
pos_percent = 0.1 # процент числа позиций (от 0.01 до 1)
h_percent = 0.1 # процент высоты рамки для встраивания (от 0.01 до 0.5)
w_percent = 0.1 # процент ширины рамки для встраивания (от 0.01 до 0.5)

# Кодирование двоичной строки
def strEncode(msg_plain):
    msg_cypher = ''
    for m in msg_plain:
        if m == ' ':
            msg_cypher += '1111111'
        else:
            msg_cypher += format(ord(m)-1024, '07b')
    msg_cypher += '0000000'
    return msg_cypher

# Декодирование двоичной строки
def strDecode(msg_cypher):
    temp = ''
    msg_plain = ''
    for b in range(len(msg_cypher)):
        temp += msg_cypher[b]
        if len(temp) == 7:
            if temp == '1111111':
                msg_plain += ' '
            elif temp == '0000000':
                break
            else:
                msg_plain += chr(int(temp, 2)+1024)
        temp = ''
    return msg_plain

# Преобразование видео-контейнера в список BGR кадров
def videoToList(path):
    frames_list = []
    # Открываем видео-контейнер на чтение
    vid_in = cv2.VideoCapture(path)
    # Считываем первый кадр
    is_frame, frame = vid_in.read()
    vid_height, vid_width = frame.shape[:2]
    vid_size = (vid_width, vid_height)
    # Считываем последующие кадры
    while is_frame:
        frames_list.append(frame)
        is_frame, frame = vid_in.read()
    vid_in.release()
    return tuple(frames_list), vid_size

# Преобразование списка BGR кадров в видео-контейнер
def listToVideo(frames_list, vid_size, path):
    # Создаем видео-контейнер на запись с кодеком и фпс
    video_out = cv2.VideoWriter(path, cv2.VideoWriter_fourcc(*'FFV1'), 25, vid_size)
    # Записываем все кадры
    for i in range(len(frames_list)):
        video_out.write(frames_list[i])
    video_out.release()
    return 0

def canEmbed(msg_cypher, vid_size, frames_count):
    pre_size = len(bin(frames_count)) - 2
    vid_width, vid_height = vid_size
    pos_width = math.ceil(vid_width*w_percent)
    pos_height = math.ceil(vid_height*h_percent)
    pos_count = math.ceil(((vid_width*pos_height)*2 + ((vid_height-
2*pos_height)*pos_width)*2)*pos_percent)
    bit_count = math.ceil(len(msg_cypher) / (pos_count - pre_size*2)) * pos_count
    pos_count *= frames_count
    if bit_count > pos_count:
        print('Недостаточно кадров для встраивания')
        exit(-1)
```

```

# Генерация списка позиций в кадре для встраивания
def getPositions(vid_size):
    pos_list = []
    pos_list_final = []
    vid_width, vid_height = vid_size
    pos_width = math.ceil(vid_width*w_percent)
    pos_height = math.ceil(vid_height*h_percent)
    # Генерируем список позиций для встраивания
    [ [ pos_list.append(tuple((a, b))) for b in range(vid_width) ] for a in range(vid_height-
pos_height, vid_height) ] #нижняя рамка
    [ [ pos_list.append(tuple((a, b))) for b in range(vid_width-pos_width, vid_width) ] for a
in range(pos_height, vid_height-pos_height) ] #правая рамка
    [ [ pos_list.append(tuple((a, b))) for b in range(pos_width) ] for a in range(pos_height,
vid_height-pos_height) ] #левая рамка
    [ [ pos_list.append(tuple((a, b))) for b in range(vid_width) ] for a in range(pos_height)
] #верхняя рамка
    random.seed(3) #Ключ для случайного перемешивания
    # Перемешивание списка позиций
    random.shuffle(pos_list)
    # Выборка позиций для встраивания в кадр
    for i in range(0, len(pos_list), round(1/pos_percent)):
        pos_list_final.append(pos_list[i])
    return tuple(pos_list_final)

# Разбиение двоичной строки на список строк формата (№ блока + содержимое + № блока)
def msgToList(msg_cypher, list_size, frames_count):
    msg_cypher_list = []
    pre_size = len(bin(frames_count)) - 2
    n = 0
    for i in range(0, len(msg_cypher), list_size-pre_size*2):
        # При несовпадении длин остатка сообщения и блока добавление '0'ей в конец сообщения
        if len(msg_cypher[i:i+list_size-pre_size*2]) < list_size-pre_size*2:
            msg_cypher_list.append(format(n, '0'+str(pre_size)+'b') +
msg_cypher[i:i+list_size-pre_size*2] + (list_size-pre_size*2 - len(msg_cypher[i:i+list_size-
pre_size*2]))*'0' + format(n, '0'+str(pre_size)+'b'))
            # Добавление номера блока в двоичном виде к началу и концу блока
        else:
            msg_cypher_list.append(format(n, '0'+str(pre_size)+'b') +
msg_cypher[i:i+list_size-pre_size*2] + format(n, '0'+str(pre_size)+'b'))
            n += 1
    return tuple(msg_cypher_list)

# Формирование двоичной строки из списка строк формата (№ блока + содержимое + № блока),
отсортированных по возрастанию № блока
def listToMsg(msg_cypher_list, list_size, frames_count):
    msg_cypher = ''
    temp_list = []
    pre_size = len(bin(frames_count)) - 2
    # Формирование списка с кортежами формата (№ блока, содержимое) из списка строк
    for s in msg_cypher_list:
        temp_list.append(tuple((int(s[:pre_size], 2), s[pre_size:-pre_size])))
    # Сортировка блоков по возрастанию номеров
    temp_list.sort(key=lambda num: num[0])
    # Склеивка двоичной строки
    for i in range(len(temp_list)):
        msg_cypher += temp_list[i][1]
    return msg_cypher

# Расчет кол-ва измененных бит после встраивания
def msgEmbedTest(msg, pos_list, frame):
    temp_frame = frame.copy()
    changes = 0
    # Перебор битов в строке
    for m in range(len(msg)):
        y, x = pos_list[m][0], pos_list[m][1]
        if msg[m] == '0':
            if temp_frame[y][x][0] % 2 == 1: # 0 == Blue
                changes += 1
        elif msg[m] == '1':
            if temp_frame[y][x][0] % 2 == 0:
                changes += 1
    return changes

# Поиск пар формата (№ кадра, № блока) с лучшим показателем для встраивания
def getBestFrames(msg_cypher_list, pos_list, frames_list):
    frames_best_list = [] # список лучших кадров для встраивания
    temp_list = []
    # Формирование списка словарей [№ кадра]{№ блока : кол-во изменений}
    for f in range(len(frames_list)):
        temp_dict = {}
        for m in range(len(msg_cypher_list)):
            temp_dict[m] = msgEmbedTest(msg_cypher_list[m], pos_list, frames_list[f])
        temp_list.append(temp_dict)

```

```

# Поиск лучших кадров
for m in range(len(msg_cypher_list)):
    best = [0, 999999] # [№ кадра, кол-во изменений]
    for f in range(len(frames_list)):
        if (temp_list[f][m] < best[1]) and (f not in frames_best_list):
            best[0], best[1] = f, temp_list[f][m]
    frames_best_list.append(best[0])
return tuple(frames_best_list)

# Встраивание сообщения в лучшие кадры
def msgEmbed(msg_cypher_list, pos_list, frames_list, f_b_l):
    for msg in range(len(msg_cypher_list)):
        for m in range(len(msg_cypher_list[msg])):
            y, x = pos_list[m][0], pos_list[m][1]
            if msg_cypher_list[msg][m] == '0':
                if frames_list[f_b_l[msg]][y][x][0] % 2 == 1:
                    frames_list[f_b_l[msg]][y][x][0] -= 1
            elif msg_cypher_list[msg][m] == '1':
                if frames_list[f_b_l[msg]][y][x][0] % 2 == 0:
                    frames_list[f_b_l[msg]][y][x][0] += 1
    return frames_list

# Извлечение списка строк со встроенным сообщением из кадров
def msgExtract(frames_list, pos_list):
    msg_cypher_list = []
    pre_size = len(bin(len(frames_list))) - 2
    for f in range(len(frames_list)):
        temp = ''
        for yx in range(len(pos_list)):
            y, x = pos_list[yx][0], pos_list[yx][1]
            if frames_list[f][y][x][0] % 2 == 1:
                temp += '1'
            elif frames_list[f][y][x][0] % 2 == 0:
                temp += '0'
        if temp[:pre_size] == temp[-pre_size:]:
            msg_cypher_list.append(temp)
    return msg_cypher_list

# Расчет параметра PSNR
def getPsnr(path1, path2):
    frames_list1, vid_size1 = videoToList(path1)
    frames_list2, vid_size2 = videoToList(path2)
    vid_width, vid_height = vid_size1
    frames_count = len(frames_list1)
    psnr = 0
    for f in range(len(frames_list1)):
        mse = 0
        for y in range(vid_height):
            for x in range(vid_width):
                mse += (int(frames_list2[f][y][x][0]) - int(frames_list1[f][y][x][0])) ** 2
        mse /= vid_height * vid_width
        if mse == 0:
            frames_count -= 1
        else:
            psnr += (10 * math.log10((255 ** 2) / mse))
    psnr /= frames_count
    return round(psnr, 3)

# Встраивание сообщения в LSB кадров видео
def embedding(msg_plain, plain_path, encode_path):
    # Кодирование двоичной строки
    msg_cypher = strEncode(msg_plain)
    # Получение списка BGR кадров и характеристик видео
    frames_list, vid_size = videoToList(plain_path)
    # Проверка на возможность встраивания сообщения
    canEmbed(msg_cypher, vid_size, len(frames_list))
    # Генерация списка позиций для встраивания в кадр
    pos_list = getPositions(vid_size) # (y, x)
    # Разбиение двоичной строки на список строк формата (№ блока + содержимое + № блока)
    msg_cypher_list = msgToList(msg_cypher, len(pos_list), len(frames_list))
    # Получение списка лучших кадров для встраивания
    frames_best_list = getBestFrames(msg_cypher_list, pos_list, frames_list)
    # Встраивание сообщения в лучшие кадры
    frames_list = msgEmbed(msg_cypher_list, pos_list, frames_list, frames_best_list)
    # Создание видео со встроенным сообщением
    listToVideo(frames_list, vid_size, encode_path)
    return 0

```

```

# Извлечение сообщения из LSB кадров видео
def extracting(encode_path):
    # Получение списка BGR кадров и характеристик видео
    frames_list, vid_size = videoToList(encode_path)
    # Генерация списка позиций для извлечения из кадра
    pos_list = getPositions(vid_size)
    # Извлечение списка строк со встроенным сообщением из кадров
    msg_cypher_list = msgExtract(frames_list, pos_list)
    # Формирование двоичной строки из списка строк формата (№ блока + содержимое + № блока)
    msg_cypher = listToMsg(msg_cypher_list, len(pos_list), len(frames_list))
    # Декодирование двоичной строки
    msg_plain = strDecode(msg_cypher)
    return msg_plain

# -----
def main():
    # Выбор встраивания/извлечения
    option = input('Для встраивания введите 1, для извлечения введите 2: ')
    while option not in '12':
        print('Неверный символ: ', option)
        option = input('Для встраивания введите 1, для извлечения введите 2: ')
    # Встраивание
    if option == '1':
        # Ввод сообщения
        msg_plain = input('Введите сообщение для встраивания в текст: ')
        # Встраивание сообщения в кадры видео
        embedding(msg_plain, plain_path, encode_path)
        # Расчет параметра PSNR
        psnr = getPsnr(plain_path, encode_path)
        print('Встроено сообщение \'', msg_plain, '\'' в видео \'', plain_path, '\'', sep='')
        # print('Параметр PSNR равен', psnr, 'dB')
    # Извлечение
    elif option == '2':
        # Извлечение сообщения из кадров видео
        msg_plain = extracting(encode_path)
        print('Извлечено сообщение \'', msg_plain, '\'' из видео \'', encode_path, '\'',
sep='')
    return 0
# -----

if __name__ == "__main__":
    main()

```