МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.03 –	Информационная безопасность автоматизированных систем
Специализация –	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ЗАЩИТЫ И СКРЫТИЯ МЕДИАДАННЫХ

Студент _____ Персиков Егор Андреевич

Руководитель инженер-программист 1-ой кат.,
филиал «Аргус» АО «ПНИЭИ» _____ М.В. Мишин

Нормоконтролёр _____ А.П. Иванов

Рецензент Нач. НИГ-01 филиал «Аргус»
АО «ПНИЭИ» _____ Р.Ю. Миняев

Работа допущена к защите (протокол заседания кафедры от 10.06.22 № 9)

Заведующий кафедрой _____ С.Л. Зефиров

Работа защищена с отметкой Отлично (протокол заседания ГЭК
от 23.06.22 г № 6)

Секретарь ГЭК _____ А.П. Иванов

Пенза 2022

Оглавление

Введение....4
1 Анализ методов сокрытия информации в изображениях....6
1.1 Понятие стеганографии и ее направления....6
1.2 Методы сокрытия информации в изображениях....7
2 Сокрытие информации в изображении формата JPEG....10
2.1 Сжатие методом JPEG....10
2.2 Сокрытие информации в изображении формата JPEG....14
3 Программная реализация алгоритма шифрования «Кузнечик»....16
3.1 Общие сведения....16
3.2 X преобразование....18
3.3 S преобразование....18
3.4 R и L преобразования....19
3.5 Развертывание ключей....21
3.6 Зашифрование....22
3.7 Расшифрование....23
4 Программная реализация считывания двумерного штрих-кода PDF417....24
4.1. Общие сведения о PDF417 и сравнение с аналогами....24
4.2. Описание формата PDF417....26
4.3 Реализация считывателя PDF417....27
5 Разработка алгоритмов работы программной части программно-аппаратного комплекса защиты и скрытия медиаданных....31
5.1 Разработка общей схемы функционирования программно-аппаратного комплекса....31
5.2 Разработка алгоритма МРП CHDK....32
5.3 Разработка алгоритма программы для извлечения и расшифрования информации....35
6 Программная реализация алгоритмов работы программной части программно-аппаратного комплекса защиты и скрытия медиаданных....39

6.1 Программная реализация алгоритма работы МРП CHDK.....39
6.2 Программная реализация алгоритма работы программы для извлечения и расшифрования информации.....44
6.3 Реализация функции гарантированного затирания файла.....45
6.4 Реализация функции удаления МРП CHDK.....47
6.5 Создание загрузочной SD-карты.....49
7 Тестирование разработанного программного обеспечения.....52
7.1 Тестирование считывателя PDF417.....52
7.2 Тестирование программного средства МРП CHDK.....53
7.3 Тестирование программного средства для получения стеганографических данных из изображений формата JPEG и их расшифровывания.....55
8 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....56
Заключение.....58
Список используемых источников.....60
Приложение А (обязательное) Листинг программы МРП CHDK.....62
Приложение Б (обязательное) Листинг программы для извлечения и расшифрования информации.....84

Введение

Цифровая фотоаппаратура стала неотъемлемым спутником современного человека. В связи с этим появилась новая возможность обеспечения конфиденциальности информации, находящейся по ту сторону объектива. Путем модификации программного обеспечения фотоаппаратуры существует возможность существенного расширения ее функционала. Таким образом, в рамках данной работы, будет реализована возможность получения, скрытия и защиты медиаданных в формате JPEG, что даст возможность не только защитить изображения с помощью криптографических алгоритмов, но и скрыть факт наличия такой информации.

В качестве темы выпускной квалификационной работы была выбрана тема «Программно-аппаратный комплекс защиты и скрытия медиаданных». Такой комплекс функционирует на ряде фотоаппаратов производителя Canon, полный список который представлен на официальном форуме резидентной программы CHDK (Canon Hacker Development Kit) [1].

В качестве алгоритма скрытия информации в изображениях формата JPEG используется алгоритм записи скрываемой информации в коэффициенты дискретного косинусного преобразования. Подробнее об этом написано в первом и втором разделах данного дипломного проекта.

В качестве алгоритма предварительного зашифрования скрываемой информации был выбран алгоритм «Кузнечик» [2]. Подробный разбор этого алгоритма описан в третьем разделе дипломного проекта.

Для более удобной возможности получения ключа для криптографического алгоритма «Кузнечик» было принято решение реализовать считыватель двумерного штрих-кода. Таким образом фотоаппаратура должна считывать штрих-код с бумажного носителя ключевой информации. Бумажный носитель имеет возможность быстрого и бесследного уничтожения ключевой информации. Подробнее о двумерных

штрих-кодах описано в четвертом разделе данной выпускной квалификационной работы.

В качестве программной части разрабатываемого программно-аппаратного комплекса выступают два программного средства:

— модифицированная резидентная программа (МРП CHDK);

— программное средству для получения стеганографических данных из изображений.

На пятом этапе дипломного проекта был составлен алгоритм основных функций для каждого программного средства, а также общая схема функционирования программно-аппаратного комплекса.

Шестой этап содержит программную реализацию алгоритмов, описанных в пятом этапе, а также следующие дополнительные функции: гарантированное затирание файлов, удаление МРП CHDK, создание загрузочной SD-карты.

В седьмом этапе было проведено тестирование разработанных программных средств.

На восьмом этапе были рассмотрены вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

1 Анализ методов сокрытия информации в изображениях

1.1 Понятие стеганографии и ее направления

Стеганография — способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи (хранения) [3].

Одним из методов обеспечения конфиденциальности хранимой или передаваемой информации является цифровая стеганография. Цифровая стеганография — направление классической стеганографии, основанное на сокрытии или внедрении дополнительной информации в цифровые объекты, вызывая при этом некоторые искажения этих объектов.

Цифровая стеганография родилась недавно и включает в себя следующие направления:

— встраивание информации с целью ее скрытной передачи или хранения;

— встраивание цифровых водяных знаков;

— встраивание идентификационных номеров;

— встраивание заголовков.

Встраивание информации с целью ее скрытной передачи или хранения используется с целью обеспечения не только конфиденциальности защищаемой информации, но и сокрытия факта наличия такого рода информации.

Цифровые водяные знаки (ЦВЗ), в основном, применяются для защиты от копирования и несанкционированного использования цифровой информации, такой как фотографии, аудио и видео. ЦВЗ могут быть невидимыми, они анализируются специальным декодером который определяет их корректность.

Встраивание идентификационных номеров похожа на технологию ЦВЗ, отличие заключается в том, что каждая копия информационного продукта

имеет свой уникальный идентификационный номер, который позволяет производителю отследить незаконное тиражирование.

Встраивание заголовков это своего рода добавление мета-информации в цифровой продукт.

Стеганографическая система состоит из двух элементов: вставка и извлечение. Стеганографическая система с применением криптографии представлена на рисунке 1.

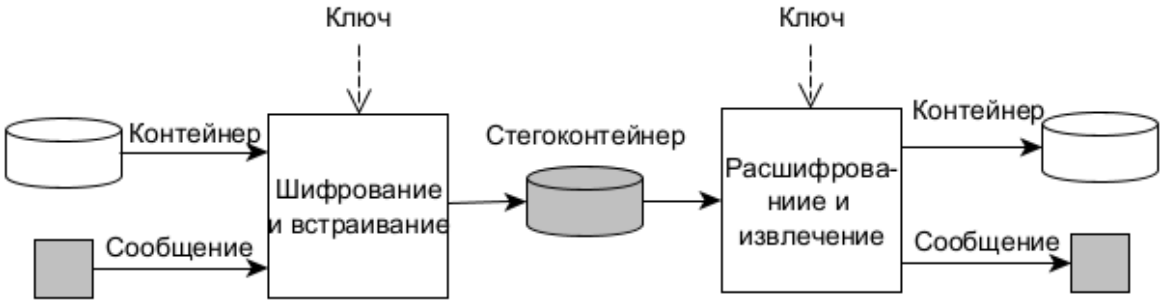

Рисунок 1 — Стеганографическая система с применением криптографии

В качестве контейнера в стеганографической системе выступает любая информация, которая является пригодной для сокрытия в ней скрываемой информации. Стегоконтейнер — контейнер, в котором находится скрытая информация. Ключ — секретный ключ, который используется криптографическими алгоритмами для обеспечения конфиденциальности скрываемого сообщения.

1.2 Методы сокрытия информации в изображениях

Существует множество методов сокрытия информации в цифровых изображениях. Один из самых популярных методов, который используется в несжимаемых изображениях или в изображениях, которые сжимаются без потерь, это метод LSB-стеганографии (Least Significant Bit).

Наиболее популярной цветовой моделью является RGB (red, green, blue), где каждый пиксель кодируется с помощью трех байт, каждый из

который отвечает за свой основной цвет. В результате смешения цветов из красного, зеленого и синего каналов пиксель получает нужный оттенок. Изменение младших бит этих байтов несет минимальное изменение изображения, которое человек заметить не способен. Размер скрываемого файла должен составлять до 1/8 размера контейнера, если изменяется только последний бит в каждом байте матрицы цветов пикселей, или до 1/4 контейнера, если изменяются последние два бита и тд. Более наглядно метод LSB представлен на рисунке 2.

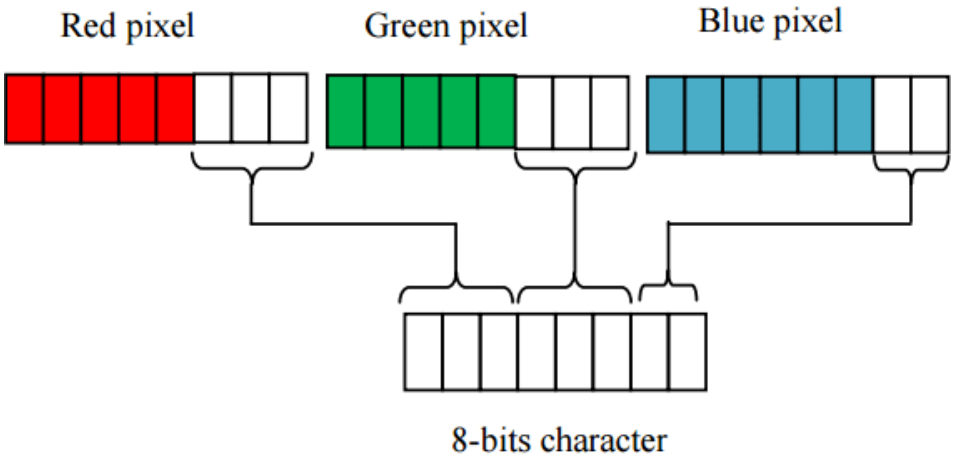

Рисунок 2 — Младшие биты в цветовой модели RGB

Метод LSB-стеганографии в изображениях формата JPEG реализуется немного по-другому. Скрываемая информация, в таком случае, встраивается в в наименее значащие биты квантованных коэффициентов дискретного косинусного преобразования (подробнее см. п. 2.2). Такой метод будет использоваться в данной работе, поскольку фотоаппаратура, для которой разрабатывается программное обеспечение, работает с изображениями формата JPEG.

Существуют и другие методы сокрытия информации, которые основаны на методе LSB, но используют в своей реализации алгоритмы, направленные на уменьшение визуального искажения изображения и увеличение вместимости изображений-контейнеров. Примером таких методов является PVD-стеганография (Pixel Value Difference — разность

значений пикселей), который заключается в том, что блоки для встраивания информации определяются по следующему принципу: на участках изображения с близкими по значениям яркости изменение будет более заметно, нежели на участках, где значения яркости разнятся сильнее. Таким образом алгоритм задействует больше младших бит в тех пикселях, у которых яркость сильнее отличается от соседних.

Для затруднения обнаружения скрытой информации в цифровых изображениях используются специальные алгоритмы внедрения скрываемой информации, например вставка только в байт определенного цвета или внедрение сообщения в байты, которые выбираются по псевдослучайной выборке. В случае использования генератора псевдослучайной последовательности для выбора адреса байта стегоконтейнера, необходимо хранить предыдущие значения выбранных адресов, так как может возникнуть коллизия во время генерации адресов и часть скрываемой информации может перетереться. Если случайный адрес был использован ранее, его необходимо отбросить и сгенерировать новый.

2 Сокрытие информации в изображении формата JPEG

2.1 Сжатие методом JPEG

Метод LSB-стеганографии в изображении формата JPEG реализуется посредством внедрения скрываемой информации в коэффициенты дискретного косинусного преобразования (ДКП). Для понимания такого метода необходимо знать как выполняется сжатие методом JPEG.

Метод сжатия JPEG – один из наиболее популярных методов сжатия с потерями, при котором уменьшение размера происходит за счет удаления избыточной информации из изображения и

Алгоритм сжатия JPEG состоит из следующих этапов:

– преобразование цветовой модели из RGB в YCbCr (где Y – яркость, Cb – синий компонент, Cr – красный компонент);

– сжатие цветовых компонент и разбиение изображения на блоки размером 8x8 пикселей;

– дискретное косинусное преобразование;

– квантование;

– преобразование кантованной матрицы в одномерный массив;

– групповое кодирование;

– кодирование Хаффмана [4].

На первом этапе происходит преобразование цветовой модели из RGB в YcbCr (Y - яркость, Cb – синий компонент, Cr – красный компонент). Для этого необходимо для каждого пикселя выполнить умножение матриц по формуле (1). Каждый элемент получившихся матриц находится в диапазоне значений от 0 до 255.

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}144 \\ -0{,}169 & -0{,}331 & 0{,}500 \\ 0{,}500 & -0{,}419 & -0{,}081 \end{vmatrix} \cdot \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \quad (1)$$

На втором этапе происходит сжатие цветовых компонент изображения разбиение изображения на блоки размером 8x8 пикселей. В матрицах, которые отвечают за красный и синий цвета, усредняются блоки по четыре пикселя, т. е. разрешение этих компонент в изображении сжимается в два раза как по координате X, так и по координате Y. Такое преобразование незначительно сказывается на визуальном восприятии изображения человеческим глазом, поскольку глаз больше воспринимает яркостную компоненту Y, чем цветовые Cb и Cr. Далее происходит разбиение изображения на блоки размером 8x8 пикселей по каждой компоненте. Поскольку каждый пискель состоит из трех компонент (Y, Cb и Cr), будет сформировано по блоку каждой составляющей размера 8x8.

На третьем этапе приминается дискретно-косинусное преобразование к каждой получившейся на предыдущем этапе матрице. Дискретное косинусное преобразование является разновидностью преобразования Фурье и, как и оно, имеет обратное преобразование. Это преобразование позволяет переключаться с пространственного отображения изображения на его спектральное отображение и наоборот. В результате выполнения данной функции из матрицы размерностью 8x8 получается блок из 64 коэффициентов, которые обозначают степень присутствия соответствующей каждому из них пространственной частоты, имеющейся в изображении. Теоретически ДКП является обратно преобразуемой функцией и не приводит к потере данных, однако на практике в ходе вычислений возникают ошибки в результате округления чисел с плавающей точкой и использования трансцендентных функций, которые приводят к небольшой потере информации. В получившейся матрице 8x8 коэффициенты, которые находятся в левом верхнем углу относятся к низкочастотной составляющей изображения, а в правом нижнем к высокочастотной. Элемент матрицы (0, 0) коэффициентов ДКП является средним значением для всего блока, значения по мере удаления от этой точки быстро убывают. Влияя на коэффициенты

ДКП, т.е. модифицируя наименее важные из них, можно найти компромисс между качеством воспроизведения и степенью сжатия.

Следующий этап в JPEG сжатии — квантование (округление коэффициентов ДКП). Квантование — это процесс уменьшения памяти, необходимой для хранения коэффициентов матрицы дискретного косинусного преобразования, в результате которого изображение теряет высокую детализацию. Процесс квантования — это поэлементное деление элементов матриц 8x8, полученных в результате ДКП, на элементы рекомендуемых стандартом JPEG, полученных опытным путем, матрицу квантования яркостной компоненты и матрицу квантования цветных компонент, пример которых приведен в ниже. Также существует возможность использования любых других матриц квантования, таким образом находя баланс между степенью сжатия и качеством результирующего изображения. В результате элементы матриц, полученные в результате деления, округляются. Формула квантования в общем виде представлена в формуле (2).

$$B_{i,j} = round\left(\frac{G_{i,j}}{Q_{i,j}}\right) \tag{2}$$

Пример матрицы квантования яркостной компоненты:

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

Пример матрицы квантования цветных компонент:

$$Q_C = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Следующим этапом сжатия является преобразование кантованной матрицы в одномерный массив для последующего сжатия Хаффмана. Преобразование матрицы 8x8 происходит по определенному принципу, показанному на рисунке 3.

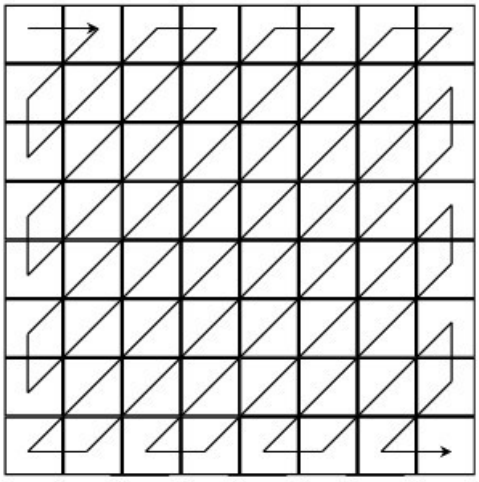

Рисунок 3 — Принцип преобразования кантованной матрицы в одномерный массив

На этапе группового кодирования полученный на предыдущем этапе вектор сворачивается, при этом получается другой вектор, состоящий из последовательности пар элементов (x, y), где x – количество элементов y. В конец массива ставится символ «EOB» (end of block), означающий конец блока.

Последним этапом сжатия JPEG является кодирование Хаффмана. В результате предыдущих шагов сформированы строки одномерных массивов с коэффициентами. Элементы массива, которые встречаются чаще всех, кодируются короткими кодовыми словами, чем реже встречаются элементы

— тем более длинными словами они кодируются.

В результате выполнения вышеописанных операций получается сжатое изображение. При сильном сжатии изображения блочная структура данных дает о себе знать, в результате чего на изображении становятся видны небольшие квадратики размером 8x8 пикселей. Это особенно заметно на однотонных участках изображения.

2.2 Сокрытие информации в изображении формата JPEG

В линейке фотоаппаратуры, поддерживаемой резидентной программой CHDK (набор разработчика для фотоаппаратуры Canon), возможна съемка в формате JPEG. Данный формат изображения интересен тем, что с ним удобно работать с помощью свободной библиотеки libjpeg, которая с легкостью позволяет извлекать коэффициенты дискретного косинусного преобразования из изображений.

При сокрытии информации в JPEG изображениях рекомендуется заполнять контейнеры (изображения, в которые будет встроена скрываемая информация) до 10% их собственного размера. Для оптимизации скорости сокрытия информации и используемой памяти в фотоаппаратуре выполнять сжатие скрываемого изображения. В рамках данной работы сжимание скрываемого изображения производиться не будет.

Внедрение скрываемой информации происходит после квантования коэффициентов дискретного косинусного преобразования, так как это этап, на котором происходит потеря информации при сжатии изображения формата JPEG. При квантовании используются матрицы квантования, значения всех элементов которых равны единице. В результате такого преобразования не происходит потерь информации, что позволяет сохранить исходный размер изображения-контейнера. После деления элементов матрицы коэффициентов дискретного косинусного преобразования на элементы матрицы квантования

яркостной компоненты и матрицы квантования цветных компонент будет производиться запись скрываемого сообщения, путем модифицирования квантованных коэффициентов ДКП. Место внедрения скрываемой информации в процессе ДКП изображения представлено на рисунке 4.

В процессе повторного сжатия изображения скрываемая информация будет потеряна.

Рисунок 4 — LSB-стеганография во время алгоритма сжатия JPEG

3 Программная реализация алгоритма шифрования «Кузнечик»

3.1 Общие сведения

Алгоритм шифрования «Кузнечик» является представителем симметричных алгоритмов блочного шифрования, который был разработан центром защиты информации и специальной связи ФСБ России с участием открытого акционерного общества "Информационные технологии и коммуникационные системы" (ОАО "ИнфоТеКС").

С начала 2016 года все средства шифрования в России должны функционировать с алгоритмами «Кузнечик» и «Магма». Рассматриваемые алгоритмы регулируются стандартом ГОСТ Р 34.12–2015 «Информационная технология. Криптографическая защита информации. Блочные шифры», который вступил в действие с 1 января 2016 года. Алгоритм шифрования «Магма» имеет длину шифруемого блока 64 бит, длину ключа 256 бит и использует сеть Фейстеля. Алгоритм «Кузнечик» теоретически является более стойким, по сравнению с «Магмой», имеет длину блока 128 бит, длину ключа 256 бит и использует подстановочно-перестановочную сеть (SP сеть).

SP сеть получает на вход ключ и блок данных определённой длины. В каждом раунде (итерации преобразования над блоком данных) выполняется стадия перестановки и подстановки. Для каждого раунда выполняется получение новой пары раундовых ключей, которые генерируются на основе главного ключа. В алгоритме шифрования «Кузнечик» выполняется 9 полных раундов и 1 неполный.

Алгоритм «Кузнечик» состоит из следующих функций (преобразований): развертывание ключа, зашифрование и расшифрование [5].

Функция шифрования и расшифрования одного блока длиной 128 бит представлена на рисунке 5.

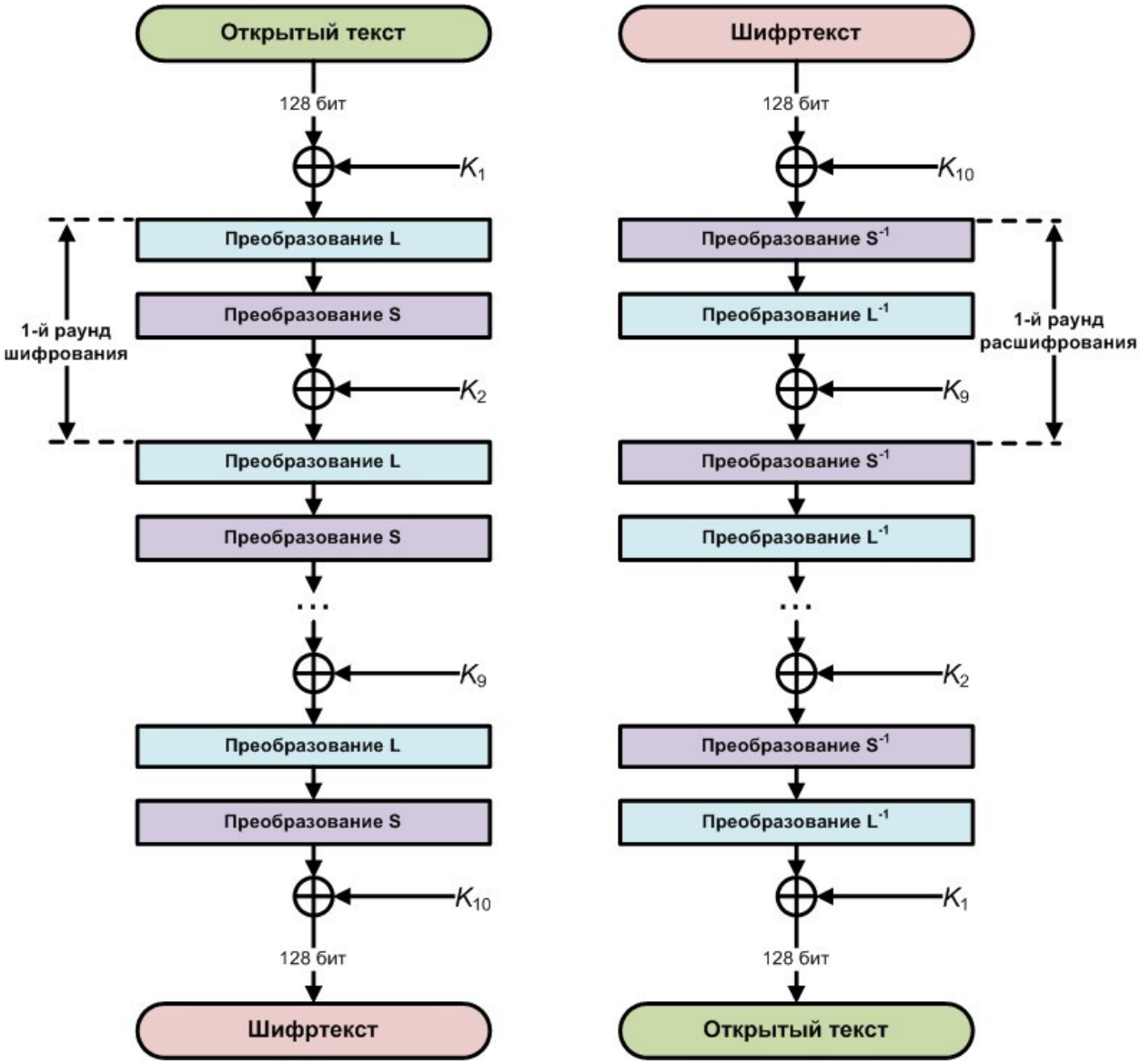

Рисунок 5 – Шифрование и расшифрование блока длиной 128 бит

ГОСТ 34.13 – 2015 предусматривает следующие режимы работы алгоритмов блочного шифрования:

— режим простой замены;

— режим гаммирования;

— режим гаммирования с обратной связью по выходу;

— режим простой замены с зацеплением;

— режим гаммирования с обратной связью по шифртексту;

— режим выработки имитовставки.

В рамках данной работы рассмотрен и выполняется только режим

простой замены.

3.2 X преобразование

X преобразование выполняется в начале и конце каждого раунда. В ходе выполнения X преобразования происходит сложение входного блока данных с раундовым ключом.

Листинг функции X преобразования:

```
//#define SIZE_BLOCK 128
void kuznechik_X(uint8_t *in_block, uint8_t *key, uint8_t *out_block)
{
 for (int i = 0; i < SIZE_BLOCK; i++)
    out_block[i] = in_block[i] ^ key[i];
}
```

3.3 S преобразование

S преобразование выполняет следующее действие: формируется выходной массив, каждый элемент которого является элементом последовательности π (массива данных размером 256 элементов) с индексом a_i, который является значением элемента входного массива. S преобразование показано на рисунке 6.

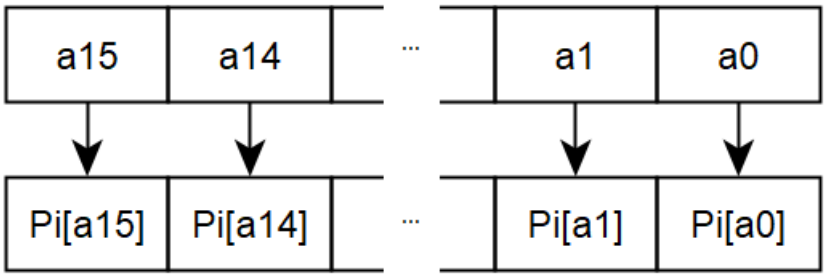

Рисунок 6 – S преобразование

Листинг функции S преобразования:

```
//#define SIZE_BLOCK 128
void kuznechik_S(uint8_t *in_block, uint8_t *out_block)
{
 for (int i = 0; i < SIZE_BLOCK; i++)
    out_block[i] = Pi[in_block[i]];
}
```

3.4 R и L преобразования

R и L преобразования используют функцию умножения чисел, выполняющую перемножение заранее известных коэффициентов на входную последовательность в конечном поле над неприводимым полиномом $x^8 + x^7 + x^6 + x + 1$. Реализация функции умножения в конечном поле:

```
uint8_t kuznechik_Mult_GF(uint8_t a, uint8_t b)
{
 uint8_t c = 0;
 uint8_t high_bit;
 for (int i = 0; i < 8; i++)
 {
    if (b & 1) c ^= a;
    high_bit = a & 0x80;
    a <<= 1;
    if (high_bit)
       a ^= 0xc3;
    b >>= 1;
 }
 return c;
}
```

R преобразование выполняется с использованием сдвигового регистра с обратной связью. Байты входного массива сдвигаются на 1 позицию в сторону младшего разряда. Вместо старшего байта результирующей

последовательности данных записывается сумма байт, сложенных по модулю 2, заранее умноженная на соответствующие коэффициенты в конечном поле. R преобразование показано на рисунке 7, где ℓ — это операция сложения и умножения входной последовательности на соответствующе коэффициенты.

L преобразование выполняет R преобразование 16 раз.

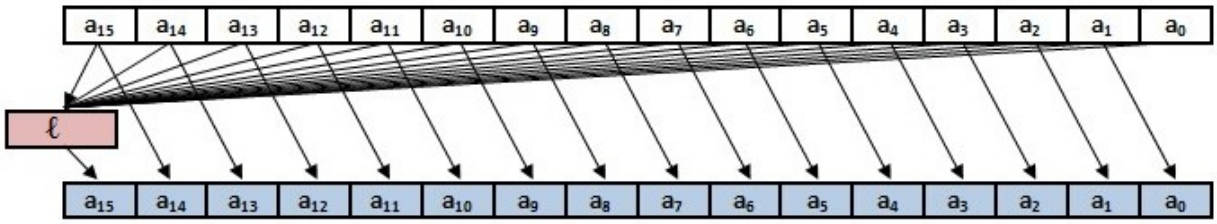

Рисунок 7 – R преобразование

Реализация функции R и L преобразований на языке Си:

```
static const unsigned char l_vec[16] = {
    1, 148, 32, 133, 16, 194, 192, 1,
    251, 1, 192, 194, 16, 133, 32, 148
};
//#define SIZE_BLOCK 128
void kuznechik_L(const uint8_t *in_block, uint8_t *out_block)
{
 vect vect_L, vect_R;
 uint8_t a15;
 memcpy(vect_L, in_block, SIZE_BLOCK);
 for (int j = 0; j < 16; j++)
 {
    a15 = 0;
    for (int i = 15; i >= 0; i--)
    {
       vect_R[i - 1] = vect_L[i];// Сдвиг к младшему байту
       a15 ^= kuznechik_Mult_GF(vect_L[i], l_vec[i]);
    }
    vect_R[15] = a15;  // Запись в освободившийся байт
    memcpy(vect_L, vect_R, SIZE_BLOCK);
```

```
 }
 memcpy(out_block, vect_L, SIZE_BLOCK);
}
```

3.5 Развертывание ключей

Для каждого раунда необходимы собственные раундовые ключи. Для вычисления раундовых ключей необходимы 32 итерационные константы, которые были вычислены заранее с помощью L преобразования. Одна итерация развертывания ключа представляет собой функцию F, которая из пары ключей и итерационной константы генерирует пару итерационных ключей. Листинг функции F:

```
//#define SIZE_BLOCK 128
void kuznechik_F(const uint8_t *in_Key1, const uint8_t *in_Key2, uint8_t *out_Key1, uint8_t *out_Key2, uint8_t *i_const)
{
 vect foo;
 memcpy(out_Key2, in_Key1, SIZE_BLOCK);
 kyznechik_X(in_Key1, i_const, foo);
 kyznechik_S(foo, foo);
 kyznechik_L(foo foo);
 kyznechik_X(foo, in_Key2, out_Key1);
}
```

Первые два итерационных ключа являются двумя половинами главного ключа. Каждая пара ключей, за исключением последней, является основой для вычисления следующей пары ключей после 8 итераций развертывания ключа.

Была проведена проверка работоспособности написанной программы на тестовом главном ключе, который был дан в ГОСТ Р 34.12-2015 в качестве примера. Итерационные ключи, использованные в ГОСТ представлены на рисунке 8.

Итерационные ключи K_i, $i = 1, 2, \ldots, 10$, принимают следующие значения:

K_1 = 8899aabbccddeeff0011223344556677,

K_2 = fedcba98765432100123456789abcdef,

K_3 = db31485315694343228d6aef8cc78c44,

K_4 = 3d4553d8e9cfec6815ebadc40a9ffd04,

K_5 = 57646468c44a5e28d3e59246f429f1ac,

K_6 = bd079435165c6432b532e82834da581b,

K_7 = 51e640757e8745de705727265a0098b1,

K_8 = 5a7925017b9fdd3ed72a91a22286f984,

K_9 = bb44e25378c73123a5f32f73cdb6e517,

K_{10} = 72e9dd7416bcf45b755dbaa88e4a4043.

Рисунок 8 – Итерационные ключи в ГОСТ Р 34.12-2015

Были сгенерированы итерационные ключи на таком же главном ключе, как и в ГОСТ. Результат представлен на рисунке 9.

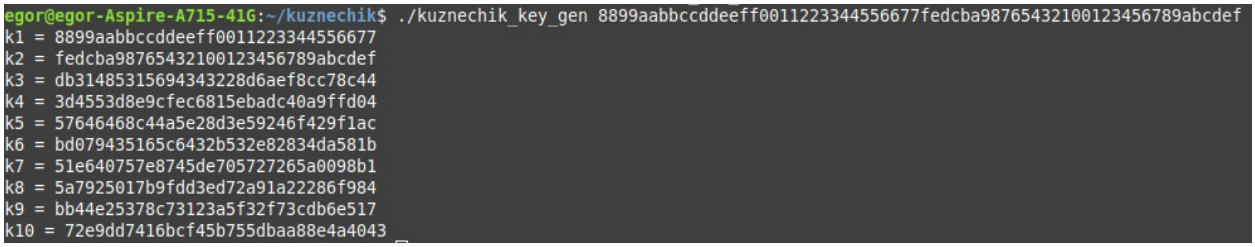

Рисунок 9 – Полученные итерационные в результате выполнения программы

3.6 Зашифрование

Зашифровывание открытого текста реализуется в результате выполнения девяти итераций, каждая из которых является последовательностью X, S, L преобразований, и неполной десятой итерации, в которой происходит только X преобразование.

Для проверки работоспособности реализованного алгоритма шифрования выполнено зашифование последовательности, представленной в ГОСТ. Результат зашифрования, представленный в ГОСТ, и результат выполнения программы представлены на рисунках 10 и 11 соответственно.

Пусть открытый текст, подлежащий зашифрованию, равен

a = 1122334455667700ffeeddccbbaa9988,

Результатом зашифрования является шифртекст

$b = X[K_{10}]LSX[K_9]\ldots LSX[K_1](a)$ = 7f679d90bebc24305a468d42b9d4edcd.

Рисунок 10 – Результат зашифрования из ГОСТ

```
Encrypted:
7f679d90bebc24305a468d42b9d4edcd
```

Рисунок 11 – Результат зашифрования разработанной программой

3.7 Расшифрование

При зашифровании и расшифровании используются одинаковые итерационные ключи. Расшифрование выполняется в обратном порядке и с обратными преобразованиями.

Отличием обратного S преобразования от прямого является использование обратной матрицы для подстановки π^{-1}.

В обратном L преобразовании используются 16 итераций обратного R преобразования. В обратном R преобразовании, в отличие от прямого, сдвиг происходит в сторону старшего разряда, а сумма, полученная в результате сложения по модулю 2, заносится в освободившийся младший байт.

Результат выполнения расшифрования разработанной программой представлен на рисунке 12.

```
egor@egor-Aspire-A715-41G:~/kuznechik$ ./kuznechik  open_text  8899aabbccddeeff001122334455667fedcba98765432100123456789abcdef
Key:
8899aabbccddeeff001122334455667fedcba98765432100123456789abcdef
Open text:
1122334455667700ffeeddccbbaa9988
Encrypted:
7f679d90bebc24305a468d42b9d4edcd
Decrypted:
1122334455667700ffeeddccbbaa9988
```

Рисунок 12 – Результат расшифрования разработанной программой

4 Программная реализация считывания двумерного штрих-кода PDF417

4.1. Общие сведения о PDF417 и сравнение с аналогами

Существует множество способов введения ключа в фотоаппаратуру для его дальнейшего использования криптографическими алгоритмами, например: заблаговременная запись ключа в файл с персонального компьютера или ввод ключа вручную с помощью текстового редактора, который предоставляет резидентная программа CHDK. Но таким способам ввода ключа присущи некоторые неудобства со стороны пользователя, поэтому было принято решение использования двумерного штрих-кода в качестве носителя ключевой информации. Одним из преимуществ такого способа получения ключа является возможность легкого уничтожения носителя.

Выбор двумерного кода состоял из наиболее популярных: QR-кода, DataMatrix, PDF417 [6]. Их внешний вид представлен на рисунке 13.

Рисунок 13 — Внешний вид рассматриваемых двумерных кодов

QR-код (Quick Response) — это двумерный штрих-код матричного типа, состоящий из черных квадратов, помещенных в один квадратный массив на белом фоне. Он может использоваться для шифрования любой информации, такой как номера телефонов, любые адреса (включая веб-адреса), а также заставить свой мобильный телефон выполнять различные команды.. Основное преимущество этого кода заключается в том, что сканеры могут

очень быстро распознавать информацию, содержащуюся в этом штрих-коде.

DataMatrix – двумерный матричный штрихкод, представляющий собой чёрно-белые элементы или элементы нескольких различных степеней яркости, обычно в форме квадрата, размещённые в прямоугольной или квадратной группе на белом фоне.

Штрих-коды PDF417 представляют собой тип сложенных двунаправленных двумерных штрих-кодов переменной длины. Это один из наиболее широко используемых двумерных штрих-кодов, который наиболее распространен в логистике, транспорте (посадочные талоны), правительственной идентификации (водительские права и удостоверения личности), инвентаризации и рабочем процессе (почтовые посылки).

В результате анализа особенностей вышеописанных двумерных штрих-кодов полученные сравнительные данные были сведены в таблицу 1.

Таблица 1 — Сравнительная таблица двумерных штрих-кодов

Название штрих-кода	QR-код	PDF417	Код DataMatrix
Плотность	Высокая	Высокая	Высокая
Набор символов	Полный ASCII	Полный ASCII	Полный ASCII
Максимальная ёмкость	2953 байт	1108 байт	1556 байт
Обнаружение и исправление ошибок	Алгоритм Рида-Соломона, позволяющий считывать код с 30% искажениями	Алгоритм Рида-Соломона, позволяющий считывать код с 50% искажениями	Алгоритм Рида-Соломона, позволяющий считывать код с 30% искажениями

В результате сравнения различных двумерных штрих-кодов было принято решение использовать PDF417 в качестве носителя ключевой информации, т. к. он имеет самый высокий уровень обнаружения и исправления ошибок по сравнению с аналогами, а максимальная ёмкость не имеет значения для реализации целей данной работы. Это преимущество PDF417 позволяет получать ключевую информацию с бумажного носителя при более слабом освещении или при большем искажении двумерного кода, а также использовать фотоаппаратуру с более низким качеством съемки.

4.2. Описание формата PDF417

Двумерный штрих-код PDF417 является свободно распространенным, его можно использовать без лицензирования или каких-либо правовых ограничений. Данная технология штрихового кодирования регулируется стандартом ГОСТ ISO/IEC 15438- 2018 [7]. PDF417 способен кодировать более 1100 байт, 1800 текстовых символов или 2710 цифр.

В ходе выполнения процедуры декодирования существует вероятность появления ошибок, они обусловлены неправильным считыванием кодового слова или же физическим дефектом носителя двумерного штрих-кода PDF417. Согласно государственному стандарту, упомянутому выше, PDF417 имеет алгоритм кодовых слов коррекции ошибок, который позволяет устранить следующие типы ошибок:

— стирание (когда позиция кодового слова известна, но она не декодируется);

— ошибку подстановки (когда позиция и значение кодового слова неизвестны).

Для устранения ошибки подстановки требуется два кодовых слова коррекции ошибки, для восстановления ошибки стирания требуется одно кодовое слово. Таким образом, чем больше кодовых слов коррекции ошибок имеет штрих-код, тем больше стираний и ошибок подстановки может быть исправлено. Соответствие уровней исправления ошибок количеству кодовых слов коррекции ошибок представлено в таблице 2.

Таблица 2 — Уровни исправления ошибок PDF417

Уровень исправления ошибок	Количество кодовых слов коррекции ошибок
0	2
1	4
2	8
3	16
4	32
5	64

Продолжение таблицы 2

Уровень исправления ошибок	Количество кодовых слов коррекции ошибок
6	128
7	256
8	512

Штрих-код PDF417 состоит из строк, количество которых варьируется от 3 до 90 штук, каждая из которых выглядит как небольшой линейный штрих-код. Структура символа PDF417 состоит из следующих элементов:

— свободная зона;

— знак «СТАРТ»;

— кодовые слова левого индикатора;

— от 1 до 30 кодовых слов данных;

— кодовые слова правого индикатора;

— знак «СТОП»;

— свободная зона.

Структура символа PDF417 представлена на рисунке 14.

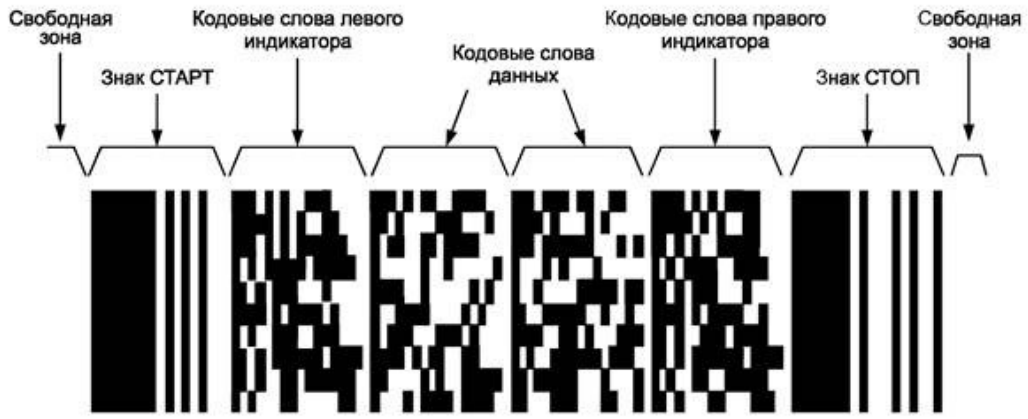

Рисунок 14 — Структура символа PDF417

4.3 Реализация считывателя PDF417

В разрабатываемом программно-аппаратном комплексе носителем секретного ключа для криптографических алгоритмов выступает двумерный штрих-код PDF417. Для преобразования закодированной информации в текст

используется многоформатная библиотека обработки изображений штрих-кодов с открытым исходным кодом 1D / 2D, которая реализованная на C++, под названием Zxing-C++ [8]. Первоначально данная библиотека была портирована из библиотеки Java ZXing, но получила дальнейшее независимое развитие и приобрела множество улучшений с точки зрения качества и производительности. Она имеет возможность считывать и создавать штрих-коды в форматах, таких как: QR-код, DataMatrix, PDF417 и других.

В рамках данной работы не будет рассматриваться процесс печати двумерного штрих-кода PDF417, так как существует множество реализованных как портативных, так и онлайн инструментов для его создания.

Для использования рассматриваемой библиотеки в качестве считывателя PDF417 необходимо выполнить загрузку изображения в память устройства. Такую возможность Zxing-C++ не предоставляет, данное действие выполняется сторонними средствами.

Чтение файла в буфер выполняется с помощью библиотеки stb_image из коллекции stb. Выполнение загрузки изображение представлено ниже:

```
stbi_uc* buffer = stbi_load(path.c_str(), &width, &height, &channels, 3);
```

Функция stbi_load принимает пять параметров: путь к изображению, ширина, высота, цветовые каналы, последний параметр — режим загрузки изображения в цветовой модели rgb. Параметры ширина, высота и цветовые каналы являются выходными и будут заполнены в результате выполнения функции.

Далее необходимо использовать функцию ReadBarcode, которая принадлежит библиотеке ZXing-C++. Прототип данной функции представлен ниже:

```
Result ReadBarcode(const ImageView& buffer, const DecodeHints&
```

```
hints = {});
```

Поскольку первым входным параметром рассматриваемой функции является объект класса ImageView, то необходимо его создать. Создание объекта класса ImageView и вызов функции ReadBarcode представлены ниже:

```
ImageView image{buffer, width, height, ImageFormat::RGB};
auto result = ReadBarcode(image, hints);
```

Полный листинг программы:

```
#include "ReadBarcode.h"
#include "TextUtfEncoding.h"
#include <cstring>
#include <iostream>
#include <string>
#define STB_IMAGE_IMPLEMENTATION
#include <stb_image.h>
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include <stb_image_write.h>
int main(int argc, char* argv[])
{
    DecodeHints hints;
    if(argc != 2){
       std::cout << "Insert one arg (path to image)" << std::endl;
       return 1;
    }
    std::string path = argv[1];
    bool angleEscape = false;
    hints.setEanAddOnSymbol(EanAddOnSymbol::Read);
    int width, height, channels;
    stbi_uc* buffer = stbi_load(path.c_str(), &width, &height, &channels, 3);
    if (!buffer) {
```

```
            std::cerr << "Failed to read image: " << path << "\n";
            return -1;
        }
        ImageView image{buffer, width, height, ImageFormat::RGB};
        auto result = ReadBarcode(image, hints);
        std::cout << path << " " << ToString(result.format());
        if (result.isValid())
            std::cout << " \"" << ToUtf8(result.text(), angleEscape)
<< "\"";
        else if (result.format() != BarcodeFormat::None)
            std::cout << " " << ToString(result.status());
        std::cout << "\n";
        return 0;
    }
```

Для проверки работоспособности написанной программы был сгенерирован двумерный штрих-код PDF417, который содержит строку «This is a PDF417 for MRP CHDK». Получившийся штрих-код представлен на рисунке 15.

Рисунок 15 — PDF417 для проверки работоспособности программы

В результате запуска написанной программы и проверки ее работоспособности двумерный штрих-код был декодирован и сообщение было выведено на экран. Результат представлен на рисунке 16.

```
egor@egor-Aspire-A715-41G:~/pdf417/zxing-cpp/build/example$ ./PDF417Reader barcode.jpg
barcode.jpg PDF417 "This is a PDF417 for  MRP CHDK"
```

Рисунок 16 — Результат проверки работоспособности программы

5 Разработка алгоритмов работы программной части программно-аппаратного комплекса защиты и скрытия медиаданных

5.1 Разработка общей схемы функционирования программно-аппаратного комплекса

Разрабатываемый программно-аппаратный комплекс, обеспечивающий скрытное получение, хранение и защиту медиаданных, состоит из следующих элементов:

— фотоаппаратура Canon, находящаяся в перечне поддерживаемых резидентной программой CHDK;

— персональных компьютер с установленной операционной системой;

— SD-карта 10 класса;

— бумажный носитель ключа в формате PDF417;

— модифицированная резидентная программа CHDK;

— программное средство для получения стеганографических данных из изображений формата JPEG и их расшифровывания.

Была разработана последовательность действий, которые выполняются в ходе работы с программно-аппаратным комплексом:

— создание загрузочной SD;

— включение защиты от записи;

— запуск МРП CHDK;

— выполнение действий в МРП CHDK:

  — фотографирование ключевой информации;

  — фотографирование скрываемой информации;

  — зашифрование и сокрытие полученного изображения.

— фотографирование ключевой информации;

— фотографирование скрываемой информации;

— зашифрование и сокрытие скрываемой информации;

— выключение защиты от записи;

— извлечение и расшифрование скрываемых изображений.

Была составлена общая схема работы программно-аппаратного комплекса. Результат представлен на рисунке 17.

Рисунок 17 — Общая схема работы программно-аппаратного комплекса

5.2 Разработка алгоритма МРП CHDK

В программе CHDK есть основная функция — core_spytask, которая работает в фоне, выполняет инициализацию интерфейса, консоли и других элементов, а также выполняет непрерывный цикл, в котором проверяется доступная необработанная информация формата RAW для ее последующей обработки, и перерисовывается графический интерфейс. Был составлен модифицированный алгоритм данной функции, в котором в цикл программы были добавлены условия, проверяющие флаги, обозначающие доступность информации (фотографий) для дальнейшей обработки (распаковка ключа или скрытие изображения). Результат представлен на рисунке 18.

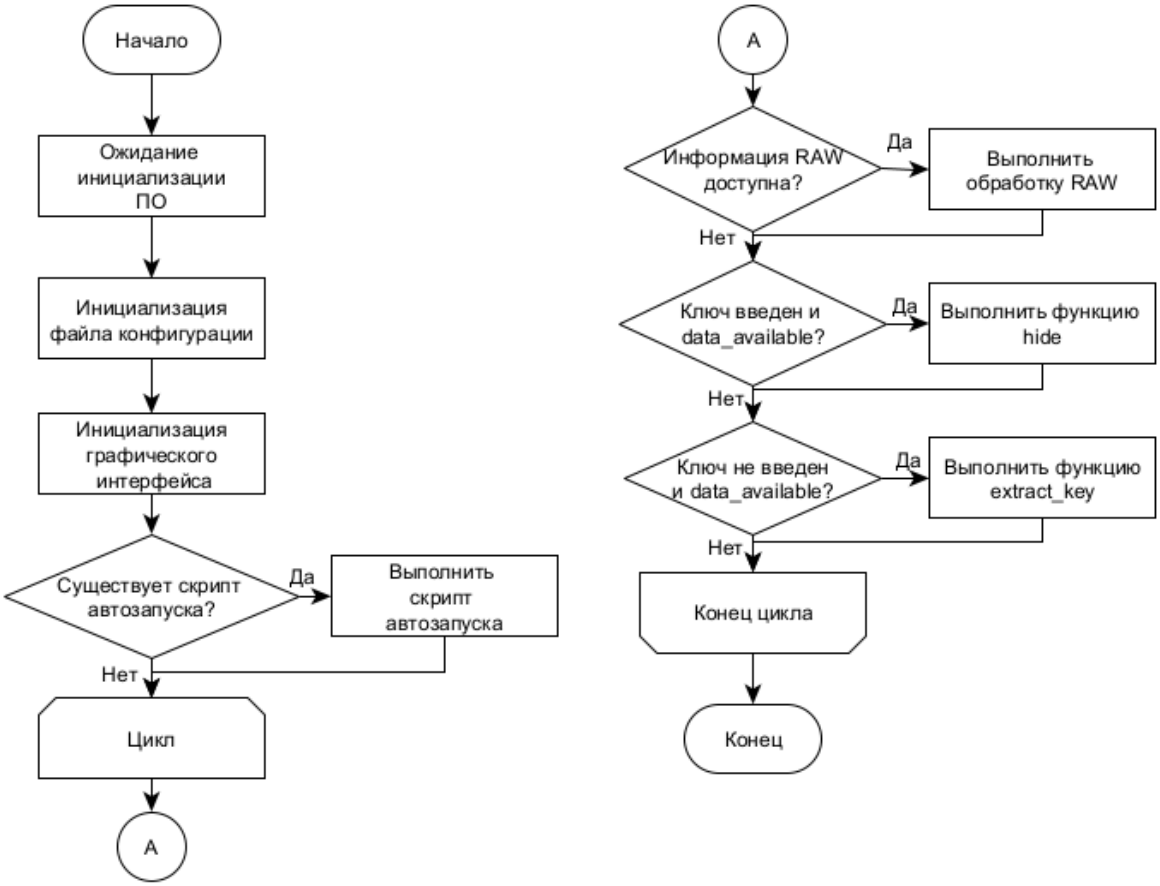

Рисунок 18 — Алгоритм функции core_spytask

Из фунции core_spytask вызываются функции hide и extract_key. Функция extract_key вызывается, если стоит флаг key_data_available. Данный флаг ставится в состояние true в случае, если было сохранено изображение формата JPG в результате фотографирования, и ключ не был введен. Функция hide вызывается, если стоит флаг hide_data_available. Этот флаг ставится в состояние true в случае, если было сохранено изображение формата JPG в результате фотографирования, и ключ был введен.

Функция extract_key выполняет распаковку ключа с фотографии носителя PDF417. Информация о файле, который является фотографией носителя ключевой информации, может быть передана в качестве аргументов или может быть найдена самостоятельно из конфигурации. В случае, если ключ не был распакован выводится сообщение об ошибке, и флаг key_entered, обозначающий, что ключ был введен, не изменится. Был

составлен алгоритм функции extract_key. Результат представлен на рисунке 19.

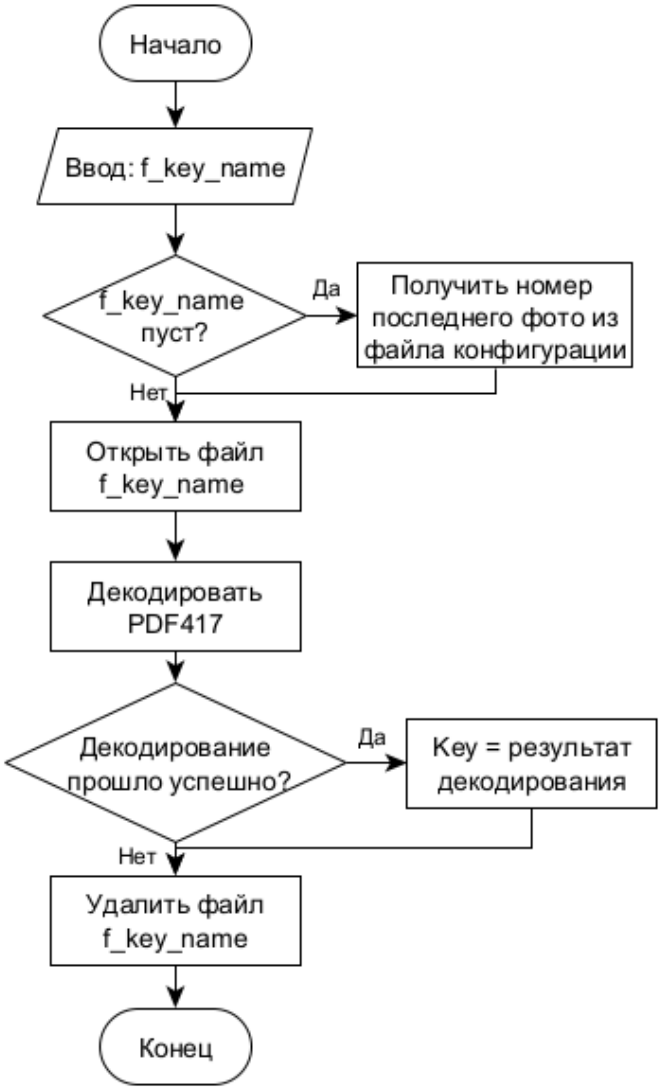

Рисунок 19 — Алгоритм функции extract_key

Функция hide выполняет шифрование и скрытие JPG изображения, полученного в результате фотографирования. Информация о файле (фотографии, которую необходимо скрыть), может быть передана в качестве аргументов или может быть найдена самостоятельно из конфигурации. Был составлен алгоритм функции hide. Результат представлен на рисунке 20.

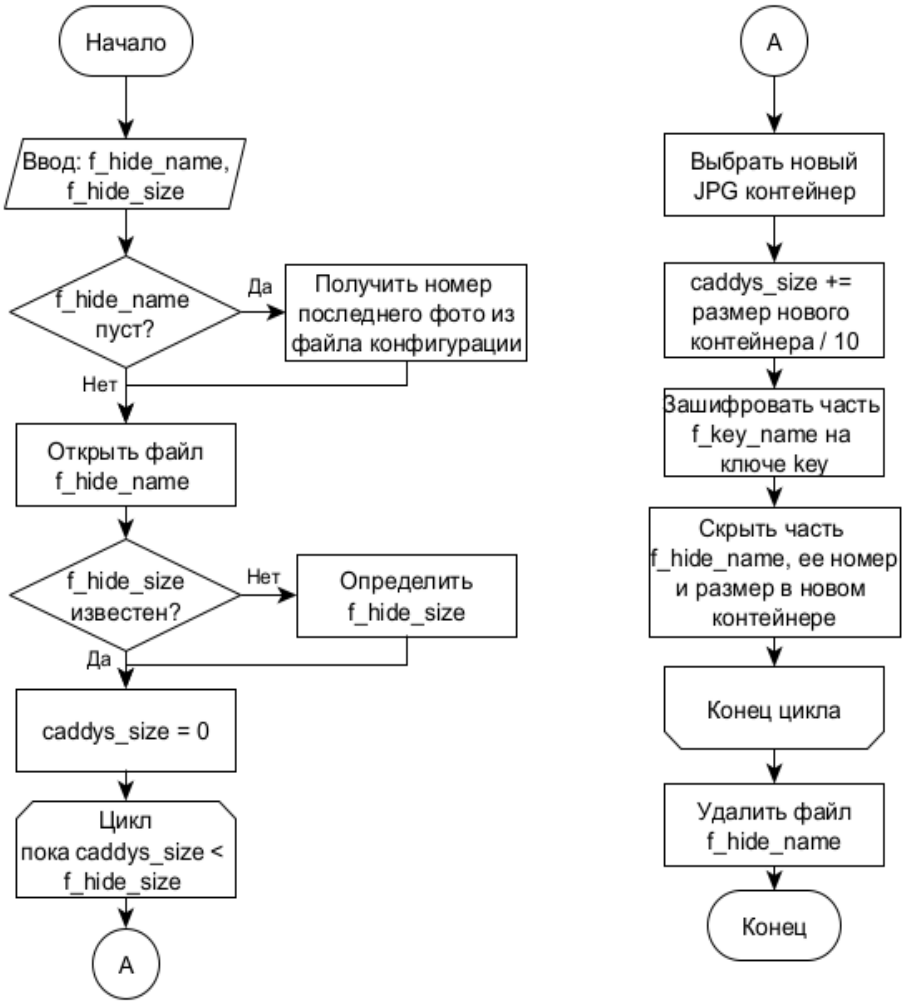

Рисунок 20 — Алгоритм функции hide

5.3 Разработка алгоритма программы для извлечения и расшифрования информации

Был разработан алгоритм программы для извлечения и расшифрования информации. Он состоит из двух функций: основная функция выполнения программы main и функция изъятия и расшифрования информации из изображения формата JPEG и записи полученной информации в выходной файл формата JPEG.

Входной информацией для функции main является путь к директории со стегоконтейнерами и ключ, на котором была зашифрована информация,

находящаяся в стегоконтейнерах. Данная функция выполняет просмотр введенной директории и формирует отсортированный контейнер с изображениями формата JPEG для последующей обработки. Далее функция main вызывает функцию jpseek, которая выполняет распаковку информации из стегоконтейнера и ее расшифрование. В зависимости от вернувшегося значения от функции jpseek, функция main выводит сообщение пользователю и формирует путь к новому выходному файлу для функции jpseek. Алгоритм функции main представлен на рисунке 21.

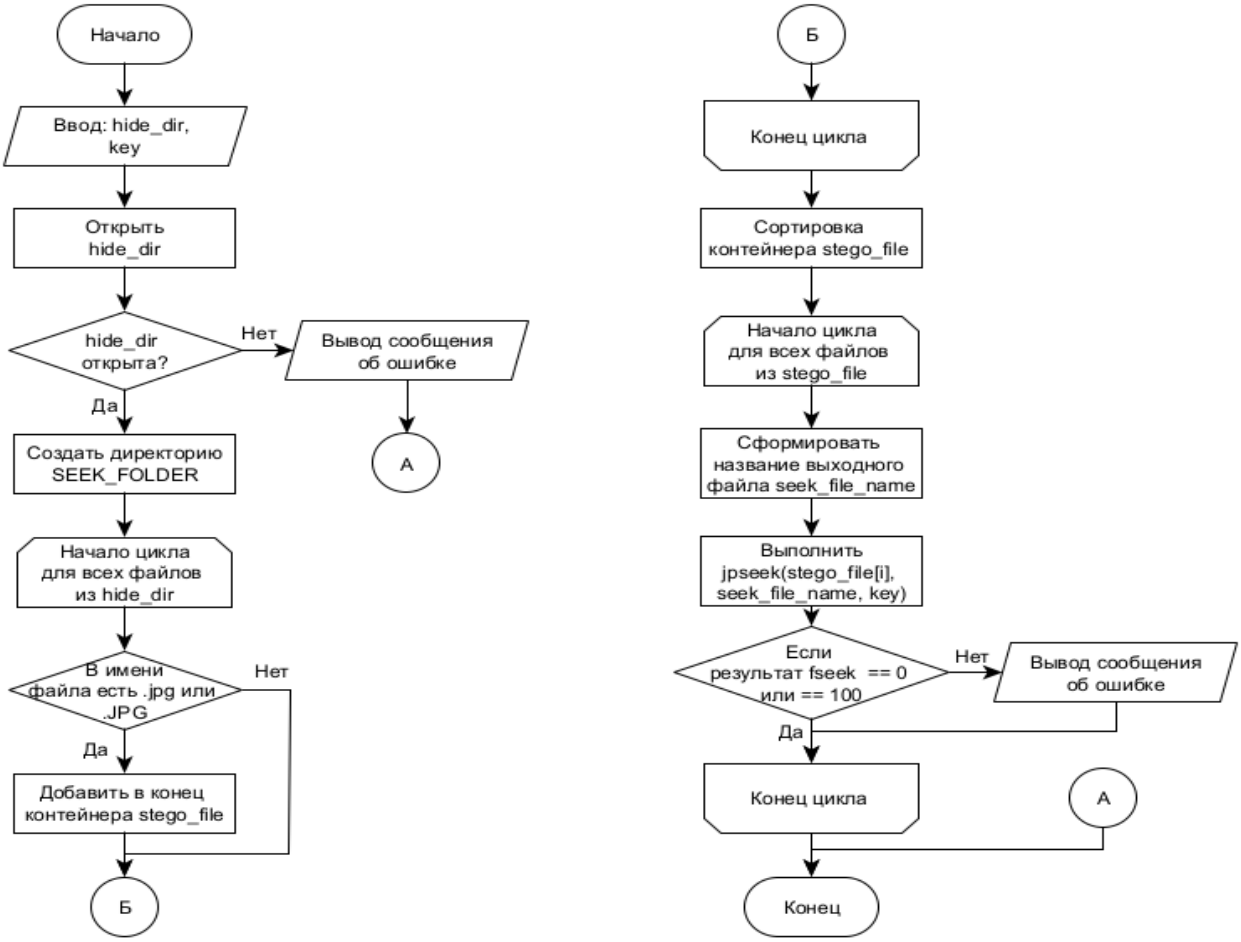

Рисунок 21 — Функция main программы программы для извлечения и расшифровывания информации

Входной информацией для функции jpseek являются: файл стегоконтейнера, файл для записи распакованной информации и ключ для расшифрования полученной информации. При выполнении рассматриваемой

функции могут возникнуть ошибки, при возникновении которых функция завершает работу и выводит код ошибки. Она (функция jpseek) выполняет распаковку скрытой информации в изображении формата JPEG и ее расшифрование. Так как одно изображение может быть скрыто в нескольких изображениях-контейнерах, а в каждом контейнере находится числа, обозначающие номер части фотографии и ее размер, функция следит за правильной последовательностью их распаковки.

Функция jpseek взаимодействует с библиотекой libjpeg для удобной работы с изображениями формата JPEG. Библиотека libjpeg предоставляет структуры jpeg_decompress_struct [9] (объект распаковки JPEG) и jpeg_error_mgr (объект обработчика ошибок JPEG) которые были использованы во время реализации алгоритма. Алгоритм функции jpseek представлен на рисунке 22.

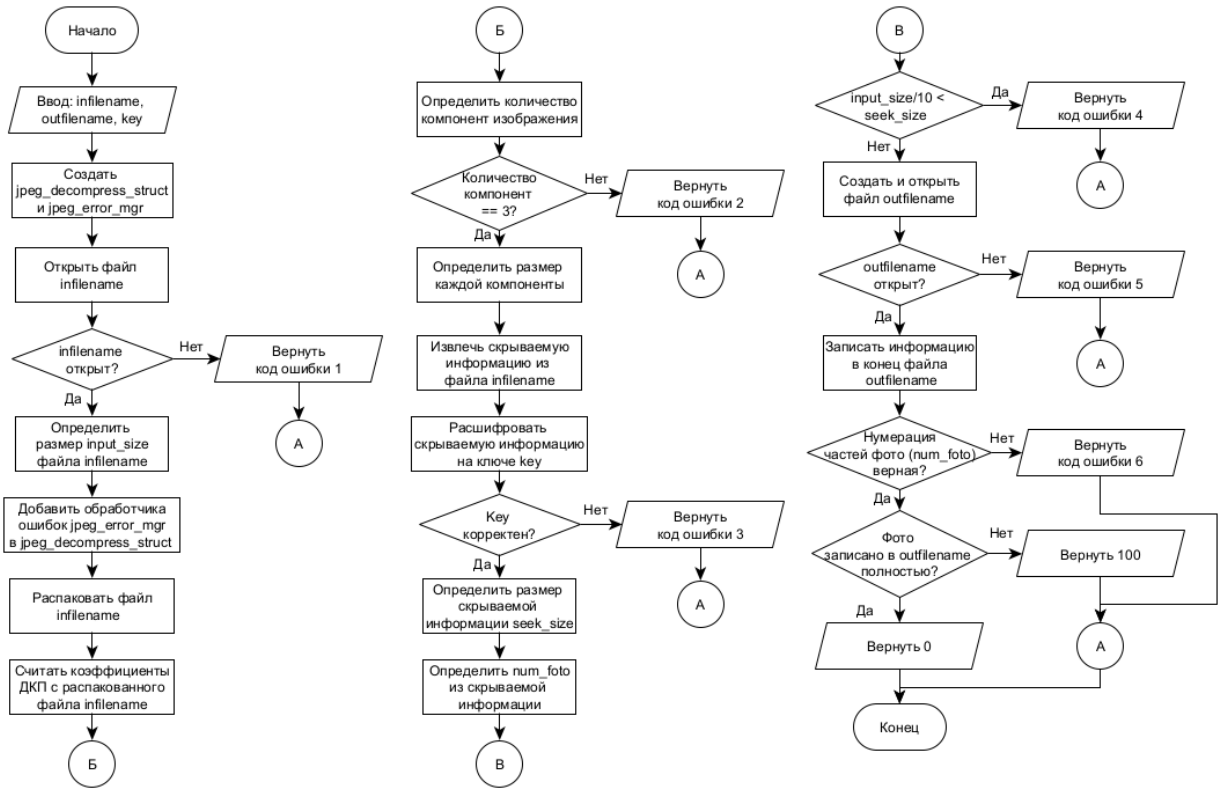

Рисунок 22 — Функция jpseek программы программы для извлечения и расшифровывания информации

Была составлена таблица соответствия кодов ошибок, возвращаемых

функцией jpseek, и информацией о них, выводимой функцией main. Результат представлен в таблице 3.

Таблица 3 — Соответствие кодов ошибок с выводимой о них информацией

Код ошибки	Информация об ошибке
1	Не возможно открыть входной файл Убедитесь, что файл существует
2	Не подходящий входной файл
3	Проверьте правильность введенного ключа
4	Входной файл не содержит информации Проверьте корректность введенного ключа
5	Не возможно создать файл Проверьте свободное место на диске
6	Не корректная нумерация в файле Добавьте недостающие файлы в список

6 Программная реализация алгоритмов работы программной части программно-аппаратного комплекса защиты и скрытия медиаданных

6.1 Программная реализация алгоритма работы МРП CHDK

В основном цикле выполнения резидентной программы CHDK происходит проверка доступности изображения формата RAW, которое получается в результате фотографирования, для дальнейшей его обработки. Реагирование программного обеспечения на нажатие функциональных кнопок фотоаппаратуры происходит по прерываниям, остальное время ПО пребывает в состоянии выполнения основного цикла. Текст основного цикла программы:

```
while (1)
{
   if (raw_data_available)
   {
      raw_need_postprocess = raw_savefile();
      hook_raw_save_complete();
      raw_data_available = 0;
      continue;
   }
   if (state_shooting_progress != SHOOTING_PROGRESS_PROCESSING)
   {
      if (((cnt++) & 3) == 0)
         gui_redraw();
      histogram_process();
      #ifdef OPT_EDGEOVERLAY
      if(conf.edge_overlay_thresh &&
      conf.edge_overlay_enable)
         edge_overlay();
      #endif
      }
      if ((state_shooting_progress ==
SHOOTING_PROGRESS_PROCESSING) && (!shooting_in_progress()))
      {
         state_shooting_progress = SHOOTING_PROGRESS_DONE;
         if (raw_need_postprocess)
            raw_postprocess();
      }
      msleep(20);
```

```
 }// while
```

В рассматриваемом цикле происходит ожидание доступного RAW файла для дальнейшей его обработки и сохранения и перерисовка графического интерфейса. Данный цикл был модифицирован. В основной цикл был добавлен следующий участок текста программы:

```
if (hide_data_available && key_entered)
{
    hide(NULL, 0); //hide(char* hide_name, int hide_size)
    hide_data_available = 0;
    continue;
}
else if (key_data_available)
{
    extract_key(NULL);//extract_key(char* fkey_name)
    key_data_available = 0;
    key_entered = true;
    continue;
}
```

В участке текста программы, который расположен выше, происходит проверка на доступность ключевой информации и скрываемой информации. В случае, если скрываемая информация доступна (в файловой системе находится изображение формата JPEG, которое необходимо скрыть) и ключ введен, происходит вызов функции hide, которая скрывает изображение и удаляет его из файловой системы. В случае если ключевая информация доступна (в файловой системе находится изображение формата JPEG носителя ключевой информации PDF417), происходит вызов функции extract_key, которая считывает ключевую информацию с изображения и удаляет его. Флаг hide_data_available получает значение true, когда ключ введен и изображение было сохранено в файловую систему в результате фотографирования. Флаг key_data_available получает значение true, когда ключ не был введен и изображение было сохранено в файловую систему в результате фотографирования.

Функция hide принимает на вход следующие параметры: имя файла для сокрытия и размер этого файла. Информация о файле передается в функцию

hide в случае, если она была вызвана из контекстного меню файлового менеджера. Если функция hide вызывается после фотографирования, то эти параметры не передаются. В таком случае функция берет из файла конфигурации о новом файле и вычисляет ее размер.

```
      if(hide_name == NULL)
      {
          // получение номера папки и файла
          // получение полного пути к файлу
          sprintf(name_hide_file,"A/DCIM/%ldCANON/IMG_%ld.JPG",
get_target_dir_num(), get_target_file_num());
      }
      else
      {   // запуск из менеджера файлов
          strcpy(name_hide_file, hn);
          size_hide_file = hs;
      }
      if(!size_hide_file)
      {
          fseek(f_hide, 0, SEEK_END);
          size_hide_file = ftell(f_hide);
          fseek(f_hide, 0, SEEK_SET);
      }
```

После определения размера файла функция ищет доступное изображение-контейнер. Далее выполняется сокрытие блоков информации скрываемого изображения, а также его размера и номера части изображения. В случае, если одного контейнера не достаточно, то выполняется поиск следующего контейнера. Информация об используемых изображениях-контейнерах находится в файле конфигурации, что не дает использовать его еще раз. Листинг функции поиска контейнера представлен ниже:

```
       int next_caddy_file(char *name_file, unsigned long *size_file,
time_t *time_file)
       {
          struct stat st;
          int  col = 0;
          while(col != 100) // ограничение на количество итераций на
случай, когда файлы отсутствуют
          {
             if (conf.num_caddy > 2472) // если файлы кончились
             {return 2;}
             sprintf(name_file,       "%s/IMG_%d.JPG",       DIRCAPTION,
conf.num_caddy);
```

```
            conf.num_caddy++;
            if (stat(name_file, &st) == 0) //если файл существует
            {
               *size_file = st.st_size; //информация о размере
контейнера
               *time_file = st.st_mtime; //информация о времени
контейнера
               return 0;
            }
            col++;
         }
      }
```

Помимо блоков информации скрываемого изображения в изображение-контейнер записывается информация о размере записанных блоков, а также порядковый номер, необходимый для правильной последовательности распаковки изображения. В качестве порядкового номера записывается 0, если это последний стегоконтейнер, 255, если вся информация умещается в один файл.

В конце выполнения функции hide выполняется запись модифицированных контейнеров, а также затирание изображений-контейнеров и скрытого изображения.

Работа с изображением формата JPEG выполняется с помощью библиотеки libjpeg. При выполнении функции hide создаются следующие структуры:

```
struct jpeg_decompress_struct srcinfo; // структура распаковки
struct jpeg_compress_struct dstinfo; // структура сжатия
struct jpeg_error_mgr jsrcerr, jdsterr; // структура обработки
ошибок
```

Структуры jpeg_decompress_struct и jpeg_compress_struct являются структурами сжатия и распаковки изображения JPEG соответственно, они содержат множества вспомогательных структур и переменных, в которых находится информация об изображении, над которым выполняются какие-либо манипуляции. Структура jpeg_error_mgr является обработчиком ошибок, в ней находится таблица с кодами ошибок и, соответствующими им,

сообщениями об ошибках. Для инициализации обработчика ошибок в структурах сжатия и распаковки выполняются следующие инструкции:

```
srcinfo.err = jpeg_std_error(&jsrcerr);
dstinfo.err = jpeg_std_error(&jdsterr);
```

Далее необходимо выполнить инициализацию структур распаковки и сжатия с помощью следующих инструкций:

```
jpeg_CreateDecompress(&srcinfo);
jpeg_CreateCompress(&dstinfo);
```

Для того, чтобы передать путь к файлу изображения в соответствующую структуру, используется инструкция, в которую передается структура распаковки и путь к изображению. Пример использования вышеописанной функции:

```
jpeg_stdio_src(&srcinfo, input_file);
jpeg_stdio_dest(&dstinfo, output_file);
```

Для выполнения сокрытия информации необходимо убедиться, что изображение имеет все три RGB компоненты, для этого выполняется чтение заголовков изображения и проверка количества компонент. Листинг данного участка программы:

```
jpeg_read_header(&srcinfo, TRUE);
i = srcinfo.num_components;
if (i != 3)
{// присутствуют не все RGB компоненты
    fclose(input_file);
       jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);
    return 2;
}
```

При завершении сжатия и распаковки необходимо освободить память, используемую структурами. Выполнение освобождения выполняется с помощью следующих инструкций:

```
jpeg_finish_decompress(&srcinfo);
```

```
jpeg_destroy_decompress(&srcinfo);
jpeg_finish_compress(&dstinfo);
jpeg_destroy_compress(&dstinfo);
```

Для считывания коэффициентов ДКП, которые далее будут модифицированы, выполняется следующая инструкция:

```
coef_arrays = jpeg_read_coefficients(&srcinfo);
```

Полный листинг программы представлен в приложении А.

6.2 Программная реализация алгоритма работы программы для извлечения и расшифрования информации

Программы для извлечения и расшифрования информации принимает на вход два аргумента: директория, содержащая стегоконтейнеры, и ключ, используемый криптографическим алгоритмом «Кузнечик». Поэтому в основной функции выполняется проверка на количество аргументов: в случае, если количество аргументов не равно требуемому, то программа аварийно завершается.

Программа выполняет сканирование переданного каталога на наличие файлов с расширением «.JPG» или «.jpg», после чего пути к файлам, которые удовлетворяют этому условию, помещаются в строковый контейнер и сортируются. Работа с директориями выполняется с помощью библиотеки «dirent.h». Рассматриваемый фрагмент программы представлен ниже:

```
while((tmp=readdir(hided_dir))){
   name=string(tmp->d_name);
   if(!(name.find(".JPG")!=string::npos ||
      name.find(".jpg")!=string::npos))
      continue;
   stego_file.push_back(string(argv[1]) + name);
}
sort(stego_file.begin(), stego_file.end());
```

Далее происходит выполнение цикла для каждого элемента вектора с путями к изображениям. Сначала формируется новый путь к

результирующему изображению:

```
name_file = string("SEEK_FOLDER/SEEK_") + to_string(seek_counter)
+ string(".JPG");
```

Переменная seek_counter увеличивается на единицу каждый раз, когда выполняется полное извлечение изображение из стегоконтейнеров, поэтому одно и то же значение переменной name_file может не меняться несколько итераций, пока скрываемое изображение не будет распаковано полностью. Листинг рассматриваемого цикла представлен ниже:

```
for(int i = 0; i < stego_file.size(); i++){
   name_file = string("SEEK_FOLDER/SEEK_") +
               to_string(seek_counter) +
               string(".JPG");
   res = jpseek(strdup(stego_file[i].c_str()),
               strdup(name_file.c_str()),
               strdup(key.c_str()));
   if (res == 0){
      seek_counter++;
   }
   else if(res != 100)
   cout << error_jpseek(res, strdup(stego_file[i].c_str()),
                           strdup(name_file.c_str())) << endl;
}
```

Функция «error_jpseek» — обработчик ошибок, она возвращает строковый результат в соответствии со значением ошибки, которую возвращает функция «jpseek». Полный список значений ошибок представлен в пункте 5.3. Значение «0», возвращаемое функцией «jpseek» означает, что изображение было распаковано полностью, значение «100» означает, что изображение распаковано неполностью.

Работа с изображениями формата JPEG происходит таким же образом, как и в МРП CHDK.

Полный листинг программы представлен в приложении Б.

6.3 Реализация функции гарантированного затирания файла

Функция remove из библиотеки stdio.h выполняет удаление

информации о файле из файловой таблицы и освобождение памяти для новой записи. Файл может быть частично или полностью перетёрт в случае, если на его месте в памяти будет записан новый файл. Если файл таким образом не был изменен, то его можно будет восстановить различными утилитами. Для исключения возможности восстановления файлов была написана функция гарантированного затирания содержимого файла. Данная функция перезаписывает файл блоками случайных данных размером 1024 байт. Входными параметрами функции являются файл, имя файла и размер файла. Размер файла передавать необязательно, т. к. функция может установить его самостоятельно. Обязательно нужно передать либо открытый файл типа FILE*, либо его путь в файловой системе. Листинг функции представлен ниже:

```
     int del_file_sure_all(FILE *fd, char* name_file, unsigned int
size_file)
      {
         int i, part_wipe;
         //set init null
         srand(time (0));
         for (i = 0; i < N; i++) //Заполнение буфера случайными
числами
         read_data[i] = rand(); //запись в блок размером 1024 байт
случайных значений
         if (fd == NULL) // если файл не был передан
         {
            fd = fopen (name_file, "r+");
            if (fd == NULL) //если файл не открывается
               return 1;
         }
         if (!size_file) //если размер файла неизвестен
         {
            fseek(fd, 0, SEEK_END); // определение
            size_file = ftell (fd); // размера
            fseek(fd, 0, SEEK_SET); // файла
         }
         fseek(fd, 0, SEEK_SET);
         part_wipe = (size_file >> 9) << 9;
         if (part_wipe) //если размер файла больше или равен 1024 байт
         {
            while (i <  part_wipe)
            {
               fwrite(read_data, sizeof(char), N, fd); // запись
```

```
блоками по 1024 байт
               i = i + N;
            }
            fwrite(read_data, sizeof(char), size_file -  part_wipe,
fd);
         }
         else
         {
            fwrite(read_data, sizeof(char), size_file, fd);
         }
         fclose (fd);
         return 0;
      }
```

6.4 Реализация функции удаления МРП CHDK

В случае необходимости удаления программного обеспечения МРП CHDK с фотоаппаратуры были написаны функции, позволяющие это сделать.

Была написана функция, выполняющая рекурсивное удаление директории и ее содержимого. Данная функция на вход принимает путь к удаляемой директории. Листинг функции:

```
      int delete_dir(char *path)
      {
         int i;
         DIR *dir;
         struct dirent *entry;
         dir = opendir(path);
         if(dir) //если директория открыта
         {
            while((entry = readdir(dir))) //чтение файлов в директории
            {
               strcat(path, "/");
               strcat(path, entry->d_name);
               //если директория, то рекурсивно
               //директория: entry->attrib == 16, файл: entry->attrib
== 32
               if (entry->attrib == 16)
               {
                  if((strcmp(".", entry->d_name)) &&
                     (strcmp("..", entry->d_name)))
                  {
                     delete_dir(path);
                  }
```

```
            }
            else
            {
               del_file_sure_all(NULL, path, 0);
               remove(path);//this file - remove
            }
            i = strlen(path);
            while(path[i] != '/')
               path[i--] = 0;
            path[i] = 0;
         }
         closedir(dir);
         remove(path);// del folder
      }
      return 0;
   }
```

Была написана функция, выполняющая гарантированное затирание и удаление бинарного файла, файла конфигурации, а также файлов с русской и английской локализациями, поскольку в них находятся пункты меню с возможностью скрыть файл. Листинг функции:

```
void del_bin()
 {
    if (!flag_del)
    {
       DIR *d;
       struct dirent *de;
       char str[SFN];

       started(); //включение светодиода
       // удаление бинарного файла
       read_data = (char *)malloc(N); // read file block
       del_file_sure_all(NULL, NAMESOFT, 0);
       remove(NAMESOFT);
       if (!del_file_sure_all(NULL,
          "A/CHDK/LANG/russian.lng", 0))
          remove("A/CHDK/LANG/russian.lng");
       if (!del_file_sure_all(NULL,
          "A/CHDK/LANG/english.lng", 0))
          remove("A/CHDK/LANG/russian.lng");
       //# define CONF_FILE    "A/CONFFILE.CFG" see conf.c
       if (!del_file_sure_all(NULL, "A/CONFFILE.CFG", 0))
          remove("A/CONFFILE.CFG");

       delete_dir("A/CHDK");
       free(read_data);
       finished(); //выключение светодиода
```

```
            flag_del = 1; // флаг не позволит зайти еще раз
        }
    }
```

Также был добавлен пункт «Удалить МРП CHDK» в меню, нажатие которого выполняет вызов функции del_bin, а также пункт «Скрыть файл», который выполняет скрытие выделенных файлов. На время выполнения функций загорается синий светодиод. Пункты меню представлены на рисунке 23.

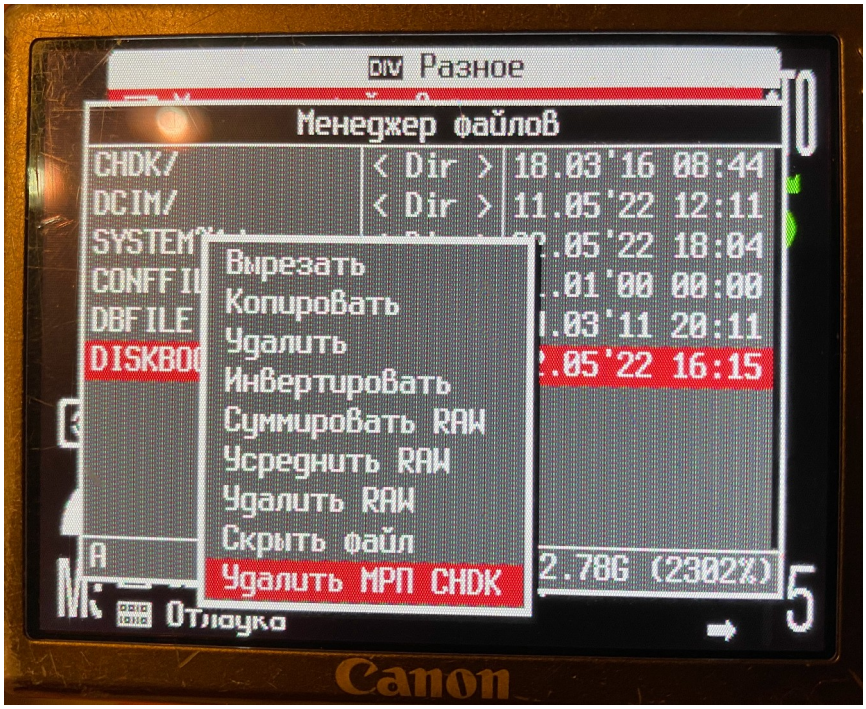

Рисунок 23 — Пункты меню «Скрыть файл» и «Удалить МРП CHDK»

6.5 Создание загрузочной SD-карты

При включении камеры Canon происходит проверка блокировки SD-карты (состояние lock), если карта заблокирована, то камера проверяет загрузочный сектор тома первого раздела на SD-карте на наличие строки «BOOTDISK», иначе камера загружается в обычном режиме. Строка «BOOTDISK» хранится с отступом 0x40 от начала загрузочного раздела файловой системы FAT16. Если надпись найдена, то камера проверяет корневой каталог SD-карты на наличие файла с именем «DISKBOOT.BIN». Если файл найден, то он загружается в память и получает дальнейшее управление. В результате загрузки программного средства фотоаппаратура

начинает игнорировать блокировку SD-карты и может производить манипуляции с файловой системой.

Для того, чтобы резидентная программа запускалась при каждом включении фотоаппаратуры необходимо сделать SD-карту загрузочной. Для этого необходимо создать раздел размером менее четырех гигабайт, поскольку файловая система FAT16 имеет некоторые ограничения. Создание раздела было выполнено с помощью утилиты fdisk. Создание файловой системы было выполнено с помощью следующей команды:

```
sudo mkfs.fat -F 16 /dev/sdb1
```

Далее была выполнена запись строки «BOOTDISK», которая обозначает, что раздел является загрузочным, в загрузочный сектор тома первого раздела. Запись была выполнена с помощью следующей инструкции:

```
sudo echo -n BOOTDISK | sudo dd bs=1 count=8 seek=64 of=/dev/sdb1
```

На созданный загрузочный раздел был скопирован бинарный файл «DISKBOOT.BIN», который является исполнительным файлом МРП CHDK, а также директория с настройками «CHDK». После этого была совершена попытка запуска фотоаппаратуры. В результате был выведен логотип CHDK, представленный на рисунке 24.

Рисунок 24 — Логотип CHDK

7 Тестирование разработанного программного обеспечения

7.1 Тестирование считывателя PDF417

Был протестирован модуль считывателя PDF417. В штрих-коде PDF417 было закодировано сообщение «This is a PDF417 for MRP CHDK» с использованием максимального уровня коррекции ошибок (512 кодовых слов коррекции ошибок). Результат кодирования представлен на рисунке 25. Результаты тестирования были сведены в таблицу 4.

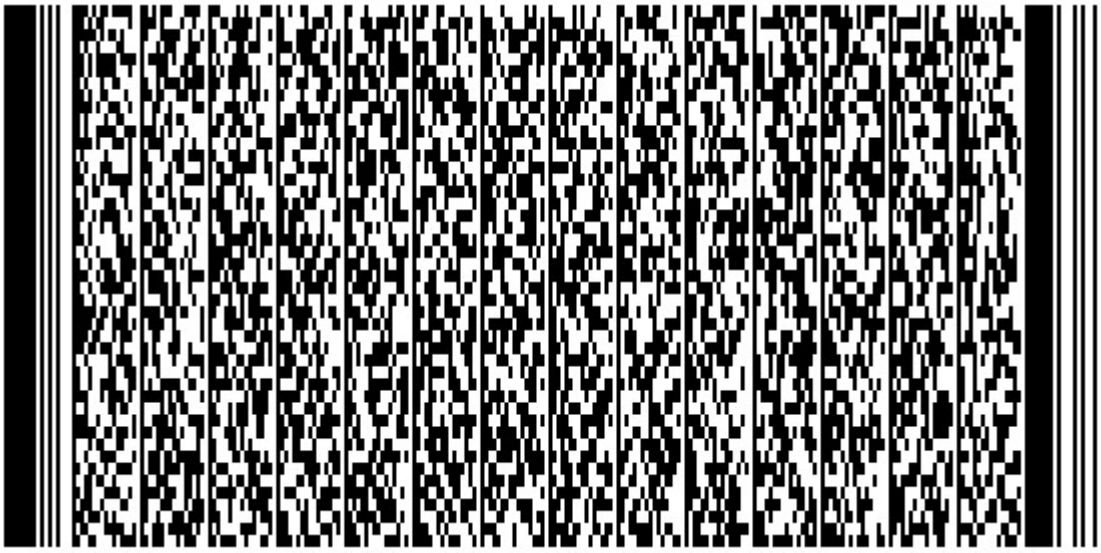

Рисунок 25 — Штрих-код PDF417 с максимальным уровнем коррекции ошибок

Таблица 4 — Результаты тестирования считывателя PDF417

Тест	Результат	Тест пройден
Считывание штрих-кода	"This is a PDF417 for MRP CHDK"	Да
Считывание повернутого на 180 градусов штрих-кода	"This is a PDF417 for MRP CHDK"	Да
Считывание отзеркаленного	"This is a PDF417 for MRP CHDK"	Да
Считывание поврежденного на 15% штрих-кода	"This is a PDF417 for MRP CHDK"	Да
Считывание поврежденного на 25% штрих-кода	"This is a PDF417 for MRP CHDK"	Да
Считывание поврежденного на 45% штрих-кода	"This is a PDF417 for MRP CHDK"	Да

Продолжение таблицы 4

Тест	Результат	Тест пройден
Считывание поврежденного более чем на 50% штрих-кода	None	Да

Максимально поврежденный штрих-код, который удалось прочитать, представлен на рисунке 26.

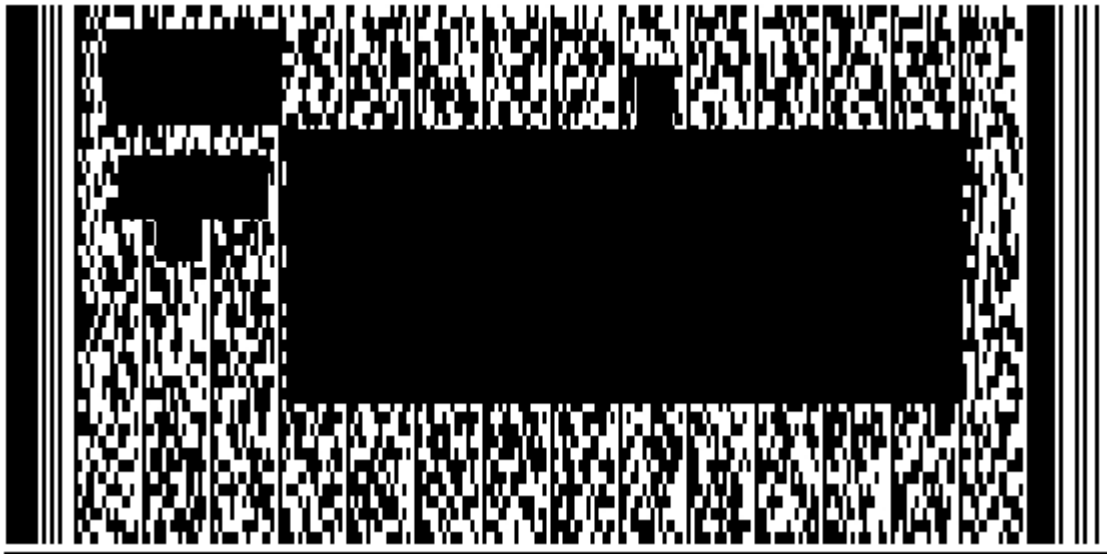

Рисунок 26 — Поврежденный штрих-код

7.2 Тестирование программного средства МРП CHDK

При запуске фотоаппаратуры с модифицированной резидентной программы высвечивается логотип CHDK с датой компиляции и версией устройства.

При первом фотографировании происходит чтение ключа с носителя ключевой информации PDF417. В случае неудачного считывания ключа необходимо повторить съемку носителя ключевой информации.

При успешном считывании ключа выполняется фото-съемка информации, которую необходимо скрыть. Во время выполнения зашифрования и скрытия полученного изображения светится синий светодиод, а также на экране выводятся изображения-контейнеры, которые были задействованы. Примерное время выполнения функции зашифрования

и скрытия информации 53 секунды. Процесс выполнения функции представлен на рисунке 27.

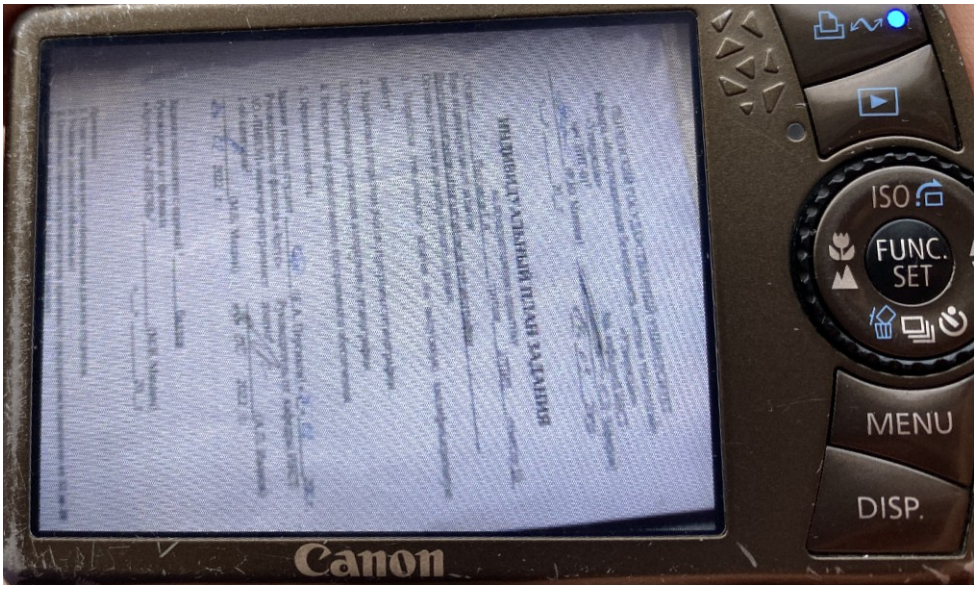

Рисунок 27 — Выполнение функции зашифрования и скрытия информации

Также существует возможность вызова рассматриваемой функции из файлового менеджера для одного или нескольких изображений. Такая возможность представлена на рисунке 28.

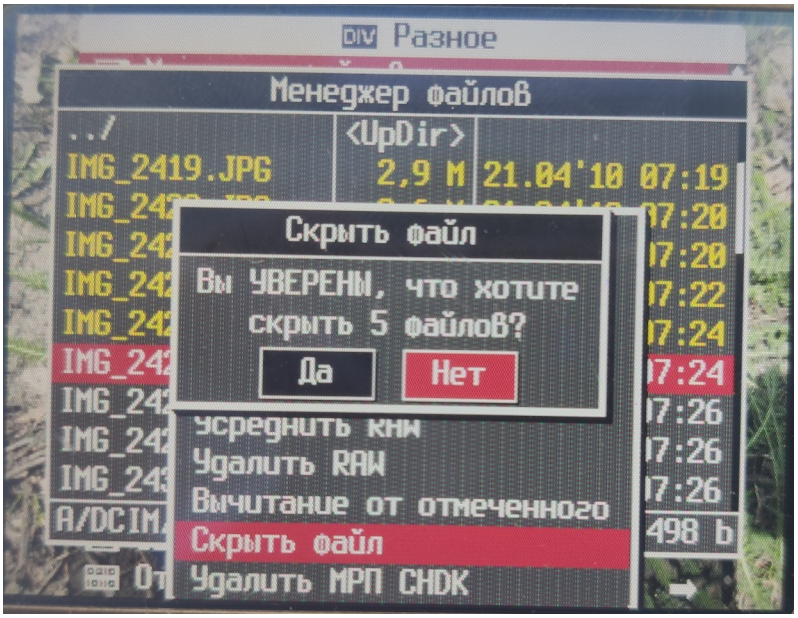

Рисунок 28 — Меню «Скрыть файл» в файловом менеджере

7.3 Тестирование программного средства для получения стеганографических данных из изображений формата JPEG и их расшифровывания

Директория со стегоконтейнерами была перемещена в память персонального компьютера. Было запущено разработанное программное обеспечение для получения стеганографических данных из изображений формата JPEG и их расшифровывания, в качестве параметров которому были переданы путь к директории со стегоконтейнерами и ключ к криптографическому алгоритму кузнечик. В ходе выполнения программы были выведены ошибки к тем файлам, которые не были затронуты во время работы МРП CHDK с данным ключом. В результате выполнения программы была создана директория «SEEK_FOLDER», в которой был создан файл «SEEK_0.JPG». Ход выполнения программы представлен на рисунке 29, результат работы представлен на рисунке 30.

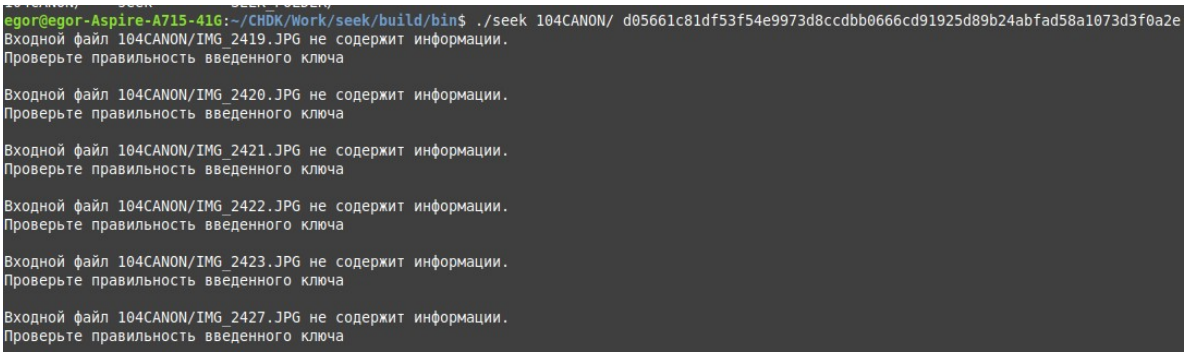

Рисунок 29 — Ход выполнения программы для получения стеганографических данных и их расшифровывания

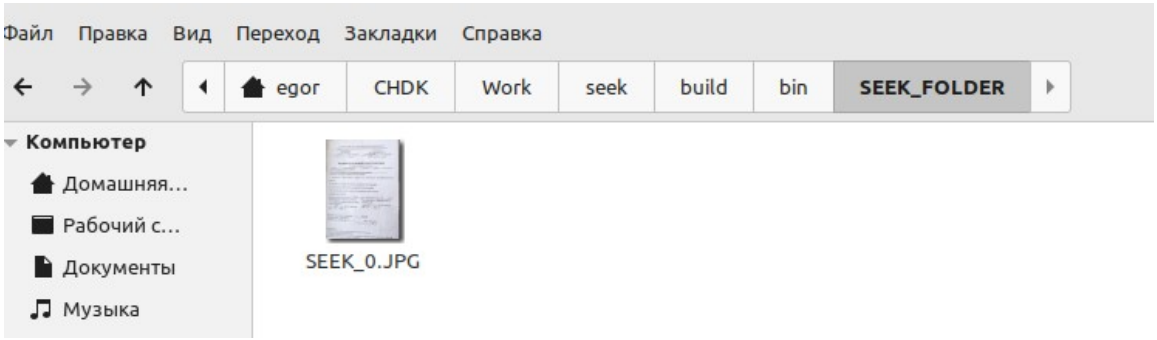

Рисунок 30 — Результат выполнения программы для получения стеганографических данных и их расшифровывания

8 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Согласно ГОСТ 12.0.003-2015 «Система стандартов безопасности труда. Опасные и вредные производственные факторы» [10] опасные и вредные психофизиологические факторы можно поделить на два типа:

— физические перегрузки, связанные с тяжестью трудового процесса;

— нервно-психические перегрузки, связанные с напряженностью трудового процесса.

В связи с тем, что работа специалиста по защите информации непосредственно связана с использованием персонального компьютера, из физических перегрузок можно выделить только статические, связанные с рабочей позой. В отличие от физических перегрузок, нервно-психических перегрузок, которые испытывает специалист, больше:

— умственное перенапряжение, в том числе вызванное информационной нагрузкой;

— перенапряжение анализаторов, в том числе вызванное информационной нагрузкой;

— эмоциональные перегрузки;

— длительность сосредоточенного наблюдения.

Для снижения воздействия статических физических перегрузок, связанных с рабочей позой, необходимо делать перерывы, на которых должны выполняться динамические упражнения, или хотя бы раз в час работник должен отходить с рабочего места и немного подвигаться.

Для психоэмоциональной разрядки советуют делать кратковременные перерывы в работе, на которых человек должен отвлечься от рабочего процесса и восстановить эмоциональное состояние. В крупных международных компаниях существуют комнаты для релаксации, в которых

созданы условия для психологической разгрузки. Например в США, в таких комнатах стоят манекены, на которых можно выплеснуть весь накопившийся гнев.

Зачастую работа специалиста по защите информации связана с государственной тайной, что, согласно статьи 24 ФЗ «О государственной тайне» [11], ведет к некоторому ограничению конституционных прав и свобод работника, а именно ограничению в: праве выезда за границу, распространение сведений, составляющих государственную тайну и т.д. Ограничение на распространение информации может негативно сказываться на психоэмоциональном состоянии работника, поскольку работник не может поделиться своими мыслями с семьей и друзьями. Как следствие работник вынужден накапливать в себе все негативные эмоции, связанные с работой, что отрицательно сказывается на психоэмоциональном состоянии специалиста по защите информации. Помимо этого таким работникам запрещены устройства связи, например сотовые телефоны. В связи с этим работник становится закрыт от внешнего мира на время рабочей смены, что мешает расслабиться и разгрузить психоэмоциональное состояние.

Заключение

В рамках данного дипломного проекта был выполнен обзор алгоритмов цифровой стеганографии, а также было принято решение использования стеганографии на основе внедрения скрываемой информации в коэффициентов дискретного-косинусного преобразования.

Был подробно рассмотрен алгоритм сжатия изображения JPEG, а также описан принцип внедрения скрываемой информации во время выполнения этого алгоритма.

Был разработан программно криптографический алгоритм «Кузнечик» на языке программирования Си, а также была проверена его работоспособность.

Далее были рассмотрены двумерные штрих-коды, а именно PDF417, QR-код и DataMatrix. В результате обзора данных штрих-кодов было выявлено, что все их заявленные характеристики подходили для задач данного программно-аппаратного комплекса, но было принято решение использовать PDF417, поскольку он имеет большую исправляющую способность, по сравнению с остальными. Также был реализован считыватль PDF417 с помощью библиотеки свободно распространяемой библиотеки Zxing.

После этого была разработана общая схема функционирования программно-аппаратного комплекса, а также была разработана последовательность действий, которые выполняются в ходе работы с данным комплексом.

Были программно реализованы разработанные алгоритмы для следующих программных средств:

— модифицированная резидентная программа;

— программное средству для получения стеганографических данных из изображений.

Также были реализованы дополнительные функции, такие как: гарантированное затирание файлов, удаление МРП CHDK, создание загрузочной SD-карты. Вместе с этим было добавлено меню в файловый менеджер фотоаппаратуры, который позволяет удалять разработанные программные средства из памяти SD-карты, а также возможность выбора нескольких изображений для скрытия.

Было произведено тестирование разработанных программных средств, а также считывателя PDF417.

Рассмотрены вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

Таким образом, задание выпускной квалификационной работы выполнено в полном объеме.

Список используемых источников

1 CHDK Wiki URL: https://chdk.fandom.com/wiki/CHDK, (дата обращения: 15.03.2022).

2 ГОСТ Р 34.12–2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» URL: https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf, (дата обращения: 20.03.2022).

3 Стеганография в XXI веке. Цели. Практическое применение. Актуальность. URL: https://habr.com/ru/post/253045/, (дата обращения: 05.04.2022).

4 Код Хаффмана. URL: https://ru.wikipedia.org/wiki/Код_Хаффмана, (дата обращения: 08.04.2022).

5 Извращения с импортзамещением. Работаем с алгоритмом блочного шифрования «Кузнечик» из ГОСТ 34.12-2015 URL: https://xakep.ru/2017/02/02/working-with-grasshopper/, (дата обращения: 13.04.2022)

6 What is the Difference Between QR Code, PDF417 and DataMatrix? URL: https://www.dynamsoft.com/blog/insights/qr-vs-pdf-417-vs-datamatrix/, (дата обращения: 20.04.2022).

7 Zxing-C++ URL: https://github.com/nu-book/zxing-cpp (дата обращения: 20.04.2022).

8 ГОСТ ISO/IEC 15438-2018 Информационные технологии (ИТ). Технологии автоматической идентификации и сбора данных. Спецификация символики штрихового кода PDF417, URL: https://docs.cntd.ru/document/1200159142 (дата обращения: 20.04.2022).

9 USING THE IJG JPEG LIBRARY URL: https://github.com/winlibs/libjpeg/blob/master/libjpeg.txt (дата обращения: 25.04.2022).

10 ГОСТ 12.0.003-2015 «Система стандартов безопасности труда.

Опасные и вредные производственные факторы» URL: https://docs.cntd.ru/document/1200136071/titles (дата обращения: 8.05.2022).

11 Закон РФ «О государственной тайне» URL: https://docs.cntd.ru/ document/9004687 (дата обращения: 08.05.2022).

Приложение А

(обязательное)

Листинг программы МРП CHDK

Файл main.c:

```
#include "platform.h"
#include "core.h"
#include "conf.h"
#include "keyboard.h"
#include "stdlib.h"
#include "gui.h"
#include "histogram.h"
#include "raw.h"
#include "motion_detector.h"
#ifdef OPT_EDGEOVERLAY
#include "edgeoverlay.h"
#endif

//remake
#include "jphide.h"

static int raw_need_postprocess;
static volatile int spytask_can_start;

void core_hook_task_create(void *tcb){}

void core_hook_task_delete(void *tcb)
{
    char *name = (char*)(*(long*)((char*)tcb+0x34));
    if (strcmp(name,"tInitFileM")==0) core_spytask_can_start();
}

long core_get_noise_reduction_value()
{
    return conf.raw_nr;
}

void dump_memory()
{
    int fd;
    static int cnt=1;
    static char fn[32];
    started();
    mkdir("A/DCIM");
    mkdir("A/DCIM/100CANON");
    sprintf(fn, "A/DCIM/100CANON/CRW_%04d.JPG", cnt++);
    fd = open(fn, O_WRONLY|O_CREAT, 0777);
    if (fd) {
#ifdef CAMERA_ixus65_sd630 // Zero is not readable on ixus65!
            write(fd, (int*)0xFFFF0000, 4);
            write(fd, (int*)4, 0x1900-4);
#else
            write(fd, (void*)0, 0x1900);
```

```
#endif
            write(fd, (void*)0x1900, 32*1024*1024-0x1900);
            close(fd);
        }
    vid_bitmap_refresh();
    finished();
}

static volatile long raw_data_available;

/* called from another process */
void core_rawdata_available()
{
    raw_data_available = 1;
}

static volatile long hide_data_available; // remake

/* called from another process */
void core_hide_available()// remake
{
    hide_data_available = 1;
}

void core_spytask_can_start() {
    spytask_can_start = 1;
}

void core_spytask()
{
    int fd = 0;
    int cnt = 1;
    int i=0;

    raw_need_postprocess = 0;

    spytask_can_start=0;

    while((i++<400) && !spytask_can_start) msleep(10);

    started();
    msleep(50);
    finished();
    drv_self_unhide();

    conf_restore();
    gui_init();

#if CAM_CONSOLE_LOG_ENABLED
```

```
    console_init();
#endif

    auto_started = 0;

    if (conf.script_startup==1)
        script_autostart(); // remote autostart
        if (conf.script_startup==2)
        {
            conf.script_startup=0;
            conf_save();
            script_autostart();
        }

    while (1)
    {
        if (raw_data_available)
        {
            raw_need_postprocess = raw_savefile();
            hook_raw_save_complete();
            raw_data_available = 0;
            continue;
        }
        // remake
        if (hide_data_available)
        {
            hide(NULL, 0);
            hide_data_available = 0;
            continue;
        }
        if (data_available && key_entered)
        {
            hide(NULL, 0); //hide(char* hide_name, int hide_size)
            data_available = 0;
            continue;
        }
        else if (data_available)
        {
            extract_key(NULL);//extract_key(char* fkey_name)
            data_available = 0;
            if(key != NULL)
                key_entered = true;
            continue;
        }

        if (state_shooting_progress != SHOOTING_PROGRESS_PROCESSING)
        {
            if (((cnt++) & 3) == 0)
                gui_redraw();
```

```
            histogram_process();
#ifdef OPT_EDGEOVERLAY
        if(conf.edge_overlay_thresh && conf.edge_overlay_enable)
                edge_overlay();
#endif
        }

        if ((state_shooting_progress == SHOOTING_PROGRESS_PROCESSING)
&& (!shooting_in_progress()))
        {
           state_shooting_progress = SHOOTING_PROGRESS_DONE;
            if (raw_need_postprocess)
                raw_postprocess();
        }
        msleep(20);
    }// while
}
```

Файл jphide.c:

```
#include "stdlib.h"
#include "jphide.h"
#include "stdarg.h"
//#include "lolevel.h"

// libjpeg
#include "jinclude.h"
#include "jpeglib.h"
#include "jerror.h" /* get library error codes too */

//kuznechik
#include "ltable.h"
#include "kuznechik.h"

#define DEBUG_MY_CHDK 1
#undef DEBUG_MY_CHDK

//debug
#define PRINT_SCREEN 1
#undef PRINT_SCREEN

#include "platform.h"

//conf.h - настройки CHDK restore num_caddy
#include "conf.h"

// interface  // show box
#include "gui_lang.h"
#include "gui_mbox.h"
#include "script.h"
```

```
// gui.h - gui_set_mode(GUI_MODE_HIDE);
//#include "gui.h"

#define NKSTREAMS 4
#define SFN 64

#define DIRCAPTION "A/DCIM/104CANON"
#define NAMESOFT "A/DISKBOOT.BIN"

//char flag_hf = 0; // срабатывает при запуске
char flag_del = 0; // модуль kbd.c

typedef unsigned long kuznechik_Data[2];

static KUZNECHIK_CTX ctx;

static jvirt_barray_ptr * coef_arrays;
static unsigned int  cpos[NKSTREAMS];

static kuznechik_Data cdata[NKSTREAMS];

static int tail;
static int tail_on;
static int mode;

static int hib[3];
static int wib[3];

static FILE *f_hide = NULL;

static int okfile;

//for function get_element & set_element
JBLOCKROW my_jblock_row;

// for read file blocks
#define N 1024
char *read_data;
int read_k = 0;
int read_l = 0;

// debug

#ifdef DEBUG_MY_CHDK
char debug_str[SFN];
//debug
FILE *debug_fd;
#endif
```

```
void jcopy_markers_setup (j_decompress_ptr srcinfo)
{
    int m;
    for (m = 0; m < 16; m++) // 5 = loop // TODO 16 !!!
    {
        jpeg_save_markers(srcinfo, JPEG_APP0 + m, 0xFFFF);
    }

}

void jcopy_markers_execute
(
        j_decompress_ptr srcinfo,
        j_compress_ptr dstinfo
)
{
    jpeg_saved_marker_ptr marker;

    for (marker = srcinfo->marker_list; marker != NULL; marker =
marker->next)
        jpeg_write_marker(dstinfo, marker->marker, marker->data,
marker->data_length);
}

static short get_element(struct jpeg_decompress_struct *jsrc, int
coef, int row, int col)
{
    return (((short**)(((void**)(((void**)coef_arrays)[coef]))[0]))
[row])[col];
}

static void set_element(struct jpeg_decompress_struct *jsrc, int coef,
int row, int col, short value)
{
    (((short**)(((void**)(((void**)coef_arrays)[coef]))[0]))[row])
[col] = value;
}

static int get_code_bit(int k)
{
    unsigned int a,b,c;
    unsigned char x,z;

    b = cpos[k] & 63;
    if (b == 0)
    {
          Kuznec_Encrypt(&ctx, &cdata[k][0], &cdata[k][1]);
    }
    a = b >> 3;
```

```
    x = ((unsigned char*)(cdata[k]))[a];
    c = b & 7;
    z = x << c;
    z = z >> 7;
    cpos[k]++;
    return(z);
}

static int merge_word(int word, int bit, int *d)
{
    int y,z;
    y = word;
    if (mode < 0)
    {
        z = bit << 1;
        if (y > 0)
        {
            y &= ~2;
            y |= z;
        }
        else
        {
            y = 0 - y;
            y &= ~2;
            y |= z;
            y = 0 - y;
        }
    }
    else
    {
        if (y == 0)
        {
            *d = 0 - *d;
            y = *d * bit;
        }
        else
        {
            if ((y == -1) || (y == 1))
            {
                *d = y;
                y = bit * y;
            }
            else
            {
                if (y > 0)
                {
                    *d = 1;
                    y &= ~1;
                    y |= bit;
```

```
                }
                else
                {
                    *d = -1;
                    y = 0 - y;
                    y &= ~1;
                    y |= bit;
                    y = 0 - y;
                }
            }
        }
    }
    return(y);
}

static int get_word
(
        struct jpeg_decompress_struct *jsrc,
        int *value,
        int *lh,
        int *lw,
        int *lt,
        int *spos,
        int *coef
)
{
    int y, ok;

    for(;;)
    {
        *lw = *lw + 64;
        if (*lw > wib[*coef])
        {
            *lw = *spos;
            *lh = *lh + 1;
            if (! (*lh < hib[*coef]))
            {
                if (okfile)
                {
                    *lt = *lt + 3;
                    if (ltab[*lt] < 0)
                    {
                        return(-1);
                    }
                    if (tail < 0)
                    {
                        if (tail_on == 2)
                        {
                            tail_on = 3;
```

```
                    tail = 999999;
                }
                if (tail_on == 1)
                {
                    tail_on = 2;
                    tail = TAIL3;
                }
                if (tail_on == 0)
                {
                    tail_on = 1;
                    tail = TAIL2;
                }
            }
        }
        else
        {
            return(1);
        }
        *coef = ltab [*lt];
        *lw = *spos = ltab [*lt+1];
        *lh = 0;
        mode = ltab [*lt+2];
    }
}
y = (int)get_element(jsrc, *coef, *lh, *lw) ;
if ((*coef) || (*lh) || (*lw > 7))
{
    ok = 1;
}
else
{
    ok = 0;
}
if (ok)
{
    if (mode < 0)
    {
        if ((y > -mode) || (y < mode))
        {
            ok = get_code_bit(0);
            ok = ok << 1;
            ok |= get_code_bit(0);
        }
        else
        {
            ok = 0;
        }
    }
    else
```

```
    {
        if (mode == 3)
        {
            ok = get_code_bit(0);
        }
        else
        {
            ok = 1;
        }
        if (ok)
        {
            if ((y > 1) || (y < -1))
            {
                ok = 0;
            }
            else
            {
                ok = get_code_bit(0);
                if (mode)
                {
                    ok = ok << 1;
                    ok |= get_code_bit(0);
                }
            }
            if (ok)
            {
                ok = 0;
            }
            else
            {
                if (mode > 1)
                {
                    ok = get_code_bit(0);
                }
                else
                {
                    ok = 1;
                }
            }
        }
    }
}
if ((ok) && (tail_on > 0))
{
    ok = get_code_bit(2);
}
if ((ok) && (tail_on > 1))
{
    ok = get_code_bit(2);
```

```
        }

        if ((ok) && (tail_on > 2))
        {
            ok = get_code_bit(2);
        }

        if (ok)
        {
            *value = y;
            return(0);
        }
        set_element(jsrc, *coef, *lh, *lw, (short)y);
    }
}

static int get_bit(unsigned long *size_f, int *data_pos, unsigned char
*data_char)
{
    unsigned int b,z;

    if (okfile)
    {
        b = (*data_pos) & 7;
        //b = data_pos & 7;
        if (b == 0)
        {
            if(!read_k)
            {// позицию в файле сравниваем с size_in_file

                if (ftell(f_hide) >= *size_f)
                          read_k = 0;
                else
                {
                    read_k = fread(read_data, sizeof(char), N,
f_hide);
                    read_l = 0;
                }

            }
            if(read_k)
            {
                *data_char = read_data[read_l];
                read_k--;
                read_l++;
            }
            else
                okfile = 0;
        }
```

```
    }

    if (okfile)
    {
        z =  (*data_char) & 128;
        z = z >> 7;
        z = z ^ get_code_bit(1);
        *data_char = (*data_char) << 1;
        *data_pos = *data_pos + 1;
//       data_pos++;
        tail--;
    }
    else
    {
        z = get_code_bit(2);
    }
    return (z);
}

int del_file_sure_all(FILE *fd, char* name_file, unsigned int
size_file)
{
    int i, part_wipe;
    srand(time (0));
    for (i = 0; i < N; i++)     //Заполнение буфера случайными числами
        //read_data[i] = 0; //int rand() ranges from 0 to RAND_MAX.
void srand (seed)
        read_data[i] = rand(); //запись в блок размером 1024 байт
случайных значений
    if (fd == NULL) // если файл не был передан
    {
        fd = fopen (name_file, "r+");
        if (fd == NULL) //если файл не открывается
            return 1;
    }
    if (!size_file) //если размер файла неизвестен
    {
        fseek(fd, 0, SEEK_END); // определение
        size_file = ftell (fd); // размера
        fseek(fd, 0, SEEK_SET); // файла
    }

    fseek(fd, 0, SEEK_SET);
    part_wipe = (size_file >> 9) << 9;

    if (part_wipe) //если размер файла больше или равен 1024 байт
    {
        while (i <  part_wipe)
        {
```

```
            fwrite(read_data, sizeof(char), N, fd); // запись блоками
по 1024 байт
            i = i + N;
        }
        fwrite(read_data, sizeof(char), size_file -  part_wipe, fd);
    }
    else
    {
        fwrite(read_data, sizeof(char), size_file, fd);
    }

    fclose (fd);
    return 0;
}

/// limited mem 198000-199000 byte
// no forget about time creat file
int jphide
(
    char* infilename,
    char* outfilename,
    char* hidefilename,
    unsigned long *length,
    unsigned long *len,
    unsigned char *num_foto
)
{
    int b,i,j,y;
    kuznechik_Data lendata;
    int d = 0;
    int spos;
    int lh = 0;
    int lt = 0;
    int lw;
    int coef = 0;
    int data_pos = 0;
    unsigned char data_char = 0;
    unsigned char iv[9];
    unsigned char len0;
    unsigned char len1;
    unsigned char len2;

    struct jpeg_decompress_struct srcinfo;
    struct jpeg_compress_struct dstinfo;
    struct jpeg_error_mgr jsrcerr, jdsterr;

    FILE * input_file;
    FILE * output_file;
```

```
    okfile = 1;

    input_file = fopen(infilename, "rb");
    if (input_file == NULL)
    {
        return 1;
    }

    srcinfo.err = jpeg_std_error(&jsrcerr);
    dstinfo.err = jpeg_std_error(&jdsterr);

    jpeg_CreateDecompress(&srcinfo);
    jpeg_stdio_src(&srcinfo, input_file);
    // Enable saving of extra markers that we want to copy
    jcopy_markers_setup(&srcinfo); //exif data
    jpeg_read_header(&srcinfo, TRUE);


// !!!!!!!!!!!!! NO FRGET A/ fopen = (A/file)

    i = srcinfo.num_components;
    if (i != 3)
    {// присутствуют не все RGB компоненты
        fclose(input_file);
        jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);

        return 2;
    }

    coef_arrays = jpeg_read_coefficients(&srcinfo);
    /*
#ifdef PRINT_SCREEN
    script_console_add_line("jpeg_read_coefficients");
#endif
 */
    for(i = 0; i < 3; i++)
    {
        hib[i] = srcinfo.comp_info[i].height_in_blocks;
        wib[i] = 64 * srcinfo.comp_info[i].width_in_blocks - 1;
    }

    for(i = 0; i < 8; i++)
    {
        iv[i] = (unsigned char)get_element(&srcinfo, 0, 0, i);
    }

    kuznechik_Init (&ctx, key, key_len);
```

```
    for(i = 0; i < NKSTREAMS; i++)
    {
        cpos[i] = 0;
        memcpy(cdata + i, iv, 8);

        kuznechik_Encrypt(&ctx, &cdata[i][0], &cdata[i][1]);

        iv[8] = iv[0];
        for(j = 0; j < 8; j++)
        {
            iv[j] = iv[j+1];
        }
    }
    memcpy(lendata, iv, 8);

    kuznechik_Encrypt(&ctx, &lendata[0], &lendata[1]);

    coef = ltab [0];
    spos = ltab [1];
    mode = ltab [2];

    lw = spos - 64;


    len0 = *length >> 16;
    len1 = (*length & 0xff00) >> 8;
    len2 = *length & 255;


    tail = (*length) * 8 - TAIL1;
    tail_on = 0;
    ((unsigned char*)lendata)[0] = len0;
    ((unsigned char*)lendata)[1] = len1;
    ((unsigned char*)lendata)[2] = len2;
    // встраиваем еще один байт нужной информации
    ((unsigned char*)lendata)[3] = *num_foto;
    kuznechik_Encrypt(&ctx, &lendata[0], &lendata[1]);

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            if (get_word(&srcinfo, &y, &lh,     &lw, &lt, &spos,
&coef))

            {
                jpeg_finish_decompress(&srcinfo);
                jpeg_destroy_decompress(&srcinfo);
                fclose(input_file);
```

```
                free_all_large();
                return 3;
            }
            b = ((unsigned char*)lendata)[i];
            b = b >> j;
            b &= 1;

            set_element(&srcinfo, coef, lh, lw, (short)merge_word(y,
b, &d));
        }
    }

    while ((b = get_word(&srcinfo, &y, &lh, &lw, &lt, &spos, &coef))
== 0)
    {
        set_element(&srcinfo, coef, lh, lw, (short)merge_word(y,
get_bit(len, &data_pos, &data_char), &d));
    }

    if (b == -1)
    {
        jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);
        fclose(input_file);
        free_all_large();
        return 4;
    }

    output_file = fopen(outfilename, "wb");
    if (output_file == NULL)
    {
        jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);
        fclose(input_file);
        free_all_large();
        return 5;
    }

    jpeg_CreateCompress(&dstinfo);

    jpeg_copy_critical_parameters(&srcinfo, &dstinfo);

   // dstinfo.optimize_coding = TRUE; //оптимизация отключена

    jpeg_stdio_dest(&dstinfo, output_file);

    jpeg_write_coefficients(&dstinfo, coef_arrays);

#ifdef PRINT_SCREEN
```

```
    script_console_add_line("jpeg_write_coefficients");
#endif
    jcopy_markers_execute(&srcinfo, &dstinfo);//exif data // TODO

    jpeg_finish_compress(&dstinfo);
    jpeg_destroy_compress(&dstinfo);

    jpeg_finish_decompress(&srcinfo);
    jpeg_destroy_decompress(&srcinfo);

    free_all_large();

    // del caddy
    del_file_sure_all(input_file, infilename, 0);
    // fclose(input_file);
    fclose(output_file);

    return 0;
}

// include conf.h
int next_caddy_file(char *name_file, unsigned long *size_file, time_t
*time_file)
{
    struct stat st;
    int  col = 0;
    while(col != 100) // ограничение на количество итераций случай
когда файлы отсутствуют
    {
        if (conf.num_caddy > 2472)
        {
            return 2;
        }
        sprintf(name_file, "%s/IMG_%d.JPG", DIRCAPTION,
conf.num_caddy);
        conf.num_caddy++;
        if (stat(name_file, &st) == 0) //если файл существует
        {
            *size_file = st.st_size;
            *time_file = st.st_mtime;
            return 0;
        }
        col++;
    }
    return 1;
}

int hide(char *hn, unsigned long hs)
{
```

```
    char infilename[SFN];
    char outfilename[SFN];
    char name_hide_file[SFN];

    unsigned long size_in_file, size_hide_file = 0, length, len = 0;

    int cont_file = 0; // number new file TODO переменную убрать
    unsigned char num_foto = 0;
    int res = 0;


#ifdef DEBUG_MY_CHDK
    debug_fd = fopen("A/dbfile", "w");
#endif
    struct utimbuf t_jpg; //change time creat file

    if(hn == NULL)
    {
        // получаем номера папки и файла
        // получаем полный путь к файлу
        sprintf(name_hide_file,"A/DCIM/%ldCANON/IMG_%ld.JPG",
get_target_dir_num(), get_target_file_num());
    }
    else
    {   // запуск из менеджера файлов

        strcpy(name_hide_file, hn);
        size_hide_file = hs;
    }

    started(); //on led

    while (f_hide == NULL)
    {
        msleep(50);
        f_hide = fopen(name_hide_file, "rb+");
    }

    if(!size_hide_file)
    {
        fseek(f_hide, 0, SEEK_END);
        size_hide_file = ftell(f_hide);
        fseek(f_hide, 0, SEEK_SET);
    }

    read_data = (char *)malloc(N); // read file block
    kuznechik_Creat(&ctx);

    while(len != size_hide_file)
```

```
    {
        read_k = 0;
        read_l = 0;

        if (next_caddy_file(infilename, &size_in_file,
&t_jpg.modtime))
        {
            //fwrite(infilename, sizeof(char), strlen(infilename),
debug_fd);
            break;
        }

        if(!strcmp(infilename, name_hide_file)) // чтоб не совпали
                continue;
        length = ((size_in_file/10)>>10) << 10;
        len += length;

        if (size_hide_file > len)
        {//т.е. потребуются еще контейнеры
                num_foto++;
        }
        else
        {
            if (num_foto != 0) //т.е. была нумерация и сейчас будет
последний файл
                num_foto = 0;
            else // т.е. все умещается в один файл num = 255
                num_foto = 255;
        }

        if(len > size_hide_file)
        {
                length = size_hide_file - (len - length) ;
                len = size_hide_file;
        }
        sprintf(outfilename, "%s/out%d.jpg", DIRCAPTION, cont_file+
+ );
#ifdef DEBUG_MY_CHDK
        fwrite(outfilename, sizeof(char), strlen(outfilename),
debug_fd);
#endif
        res = jphide(infilename, outfilename, name_hide_file, &length,
&len, &num_foto) ;
        if (res != 0)
        {
                len -= length; // откат длины
                cont_file--;
                continue;
        }
```

```
        // del caddy
        remove(infilename);

        // rename new jpg file
        RenameFile_Fut(outfilename, infilename);

        t_jpg.actime = t_jpg.modtime;
        utime(infilename, &t_jpg);      // меняется время outfilename
    }// while
#ifdef DEBUG_MY_CHDK
    fclose(debug_fd);
#endif
    kuznechik_Destroy(&ctx);
    //del_file_sure(f_hide, name_hide_file, 15, size_hide_file);//
clean data file fclose file
    del_file_sure_all(f_hide, name_hide_file, size_hide_file);
    free(read_data);
    //fclose(f_hide);
    f_hide = NULL; // обнуляем так как она глобальная
    remove(name_hide_file);// на выходе из hide должен удалять
созданный файл

    conf_save(); // сохраняем измененный num_foto
    finished();  // off led
    return 0;
}

// delete directory
int delete_dir(char *path)
{
    int i;
    DIR *dir;
    struct dirent *entry;
    dir = opendir(path);
    if(dir) //если директория открыта
    {
        while((entry = readdir(dir))) //чтение файлов в директории
        {
            strcat(path, "/");
            strcat(path, entry->d_name);
            //если директория, то рекурсивно
            //директория: entry->attrib == 16, файл: entry->attrib ==
32
            if (entry->attrib == 16)
            {
                if((strcmp(".", entry->d_name)) && (strcmp("..",
entry->d_name)))
                {
```

```
                    delete_dir(path);
                }
            }
            else
            {
                del_file_sure_all(NULL, path, 0);
                remove(path);//this file - remove
            }
            i = strlen(path);
            while(path[i] != '/')
                path[i--] = 0;
            path[i] = 0;
        }
        closedir(dir);
        remove(path);// del folder
     }
     return 0;
}

void del_bin()
{
    if (!flag_del)
    {
        DIR *d;
        struct dirent *de;
        char str[SFN];

        started(); //включение светодиода
        // удаление бинарного файла
        read_data = (char *)malloc(N); // read file block
        del_file_sure_all(NULL, NAMESOFT, 0);
        remove(NAMESOFT);

        if (!del_file_sure_all(NULL, "A/CHDK/LANG/russian.lng", 0))
            remove("A/CHDK/LANG/russian.lng");
        if (!del_file_sure_all(NULL, "A/CHDK/LANG/english.lng", 0))
            remove("A/CHDK/LANG/russian.lng");
        //# define CONF_FILE    "A/CONFFILE.CFG" see conf.c
        if (!del_file_sure_all(NULL, "A/CONFFILE.CFG", 0))
            remove("A/CONFFILE.CFG");

        delete_dir("A/CHDK");
        free(read_data);
        finished(); //выключение светодиода

        flag_del = 1; // флаг не позволит зайти еще раз
    }
}
```

Приложение Б

(обязательное)

Листинг программы для извлечения и расшифрования информации

Файл main.c:

```
#include <string>
#include <sys/stat.h>
#include "dirent.h"

#define DEBUG 1
//#undef DEBUG 1
extern "C"{
#include "kuznechik.h"
#include "ltable.h"

#include "jinclude.h"
#include "jpeglib.h"
#include "jerror.h"             /* get library error codes too */
}
#include <stdio.h>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

string error_jpseek(int er, string input_file, string output_file);

#define NKSTREAMS 4

KUZNECHIK_CTX ctx;

typedef unsigned long kuznechik_Data[2];

static jvirt_barray_ptr * coef_arrays;
static unsigned int  cpos[NKSTREAMS];
static kuznechik_Data cdata[NKSTREAMS];
static kuznechik_Data lendata;

static int length;
static int tail;
static int tail_on;
static int coef;
static int mode;
static int spos;
static int lh;
static int lt;
static int lw;
static int hib[3];
static int wib[3];
static int d;
```

```
unsigned char number_foto = 0;

static int get_code_bit(int k)
{
    unsigned int a,b,c;
    unsigned char x,z;

    b = cpos[k] & 63;
    if (b == 0)
    {
        kuznechik_Encrypt(&ctx, &cdata[k][0], &cdata[k][1]);
    }
    a = b >> 3;
    x = ((unsigned char*)(cdata[k]))[a];
    c = b & 7;
    z = x << c;
    z = z >> 7;
    cpos[k]++;
    return(z);
}

static int demerge_word(int word)
{

    int z;

    if (word < 0)
    {
        z = 0 - word;
    }
    else
    {
        z = word;
    }
    if (mode < 0)
    {
        z &= 2;
        z = z >> 1;
    }
    else
    {
        z &= 1;
    }
    return(z);
}

static int get_word(int *value)
{
 int y, ok;
```

```
 for(;;) {
  lw += 64;
  if (lw > wib[coef]) {
   lw = spos;
   lh++;
   if (! (lh < hib[coef])) {
    lt += 3;
    if (ltab[lt] < 0) {
     return(1);
    }
    if (tail < 0) {
     if (tail_on == 2) {
      tail_on = 3;
      tail = 999999;
     }
     if (tail_on == 1) {
      tail_on = 2;
      tail = TAIL3;
     }
     if (tail_on == 0) {
      tail_on = 1;
      tail = TAIL2;
     }
    }
    coef = ltab [lt];
    lw = spos = ltab [lt+1];
    lh = 0;
    mode = ltab [lt+2];
   }
  }
  y = (((short**)(((void**)(((void**)coef_arrays)[coef]))[0]))[lh])
[lw];
  if ((coef) || (lh) || (lw > 7)) {
   ok = 1;
  } else {
   ok = 0;
  }
  if (ok) {
   if (mode < 0) {
    if ((y > -mode) || (y < mode)) {
     ok = get_code_bit(0);
     ok = ok << 1;
     ok |= get_code_bit(0);
    } else {
     ok = 0;
    }
   } else {
    if (mode == 3) {
     ok = get_code_bit(0);
```

```
    } else {
     ok = 1;
    }
    if (ok) {
     if ((y > 1) || (y < -1)) {
      ok = 0;
     } else {
      ok = get_code_bit(0);
      if (mode) {
       ok = ok << 1;
       ok |= get_code_bit(0);
      }
     }
     if (ok) {
      ok = 0;
     } else {
      if (mode > 1) {
       ok = get_code_bit(0);
      } else {
       ok = 1;
      }
     }
    }
   }
  }
  if ((ok) && (tail_on > 0)) {
   ok = get_code_bit(2);
  }
  if ((ok) && (tail_on > 1)) {
   ok = get_code_bit(2);
  }
  if ((ok) && (tail_on > 2)) {
   ok = get_code_bit(2);
  }
  if (ok) {
   *value = y;
   return(0);
  }
 }
}

static int get_bit()
{
     int y;

     if (get_word(&y))
     {
        return(-1);
     }
```

```
     return (demerge_word(y));
}

#define LEN_BUF 1024

int jpseek
(
        char *infilename,
        char *outfilename,
        char *key
)
{
    FILE *my_f;
    unsigned char num_foto = 255;
    int b = 0, count = 0, i = 0, j = 0, len0 = 0, len1 = 0, len2 = 0;
    unsigned char iv[9];
    unsigned char v = 0;
    struct jpeg_decompress_struct srcinfo;
    struct jpeg_error_mgr jsrcerr;
    FILE * input_file;
    unsigned long length_input = 0;
        int index_buf = 0;
        int write_len = 0;
        char my_buffer[LEN_BUF];

///////DEBUG/////////
#ifdef DEBUG
        char debug_str[100];
        FILE *debug_fd = fopen("debug_file.txt", "a");
#endif
        length = 0;
        tail = 0;
        tail_on = 0;
        coef = 0;
        mode = 0;
        spos = 0;
        lh = 0;
        lt = 0;
        lw = 0;
        d = 0;

    if ((input_file = fopen(infilename,"rb")) == NULL)
    {
        //perror("Can't open input file");
        return (1);
    }

    fseek(input_file, 0, SEEK_END);
    length_input = ftell(input_file);
```

```
    fseek(input_file, 0, SEEK_SET);

    srcinfo.err = jpeg_std_error(&jsrcerr);
    jpeg_create_decompress(&srcinfo);
    //jpeg_CreateDecompress(&srcinfo);

    jpeg_stdio_src(&srcinfo, input_file);
    jpeg_read_header(&srcinfo, TRUE);

    coef_arrays = jpeg_read_coefficients(&srcinfo);

    i = srcinfo.num_components;
    if (i != 3)
    {
        // fprintf(stderr,"Unsuitable input jpeg file: No of
components = %d\n", i);
        jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);

        fclose(input_file);
        return 2;
    }

    for(i = 0; i < 3; i++)
    {
        hib[i] = srcinfo.comp_info[i].height_in_blocks;
        wib[i] = 64 * srcinfo.comp_info[i].width_in_blocks - 1;
    }

    for(i = 0; i < 8; i++)
    {
        iv[i] = (unsigned char)(((short**)(((void**)
(((void**)coef_arrays)[0]))[0]))[0])[i];
    }

    kuznechik_Init (&ctx, (unsigned char *)key, (int)strlen(key));
    cout << "(int)strlen(key): " << (int)strlen(key) << endl;
    for(i = 0; i < NKSTREAMS; i++)
    {
        cpos[i] = 0;
        memcpy(cdata + i, iv, 8);
        kuznechik_Encrypt(&ctx, &cdata[i][0], &cdata[i][1]);

        iv[8] = iv[0];
        for(j = 0; j < 8; j++)
        {
            iv[j] = iv[j + 1];
        }
    }
```

```
    coef = ltab [0];
    spos = ltab [1];
    mode = ltab [2];
    d = 0;
    lh = 0;
    lw = spos -64;
    lt = 0;

    for(i = 0; i < 8; i++)
    {
        v = 0;
        for(j = 0; j < 8; j++)
        {
            if ((b = get_bit()) < 0)
            {
                jpeg_finish_decompress(&srcinfo);
                jpeg_destroy_decompress(&srcinfo);
                fclose(input_file);
                return 3;
            }
            b = b << j;
            v |= b;
        }
        ((unsigned char*)lendata)[i] = v;
    }

    kuznechik_Decrypt(&ctx, &lendata[0], &lendata[1]);

    len0 = ((unsigned char*)lendata)[0];
    len1 = ((unsigned char*)lendata)[1];
    len2 = ((unsigned char*)lendata)[2];
    num_foto = ((unsigned char*)lendata)[3];
    length = (len0 << 16) | (len1 << 8) | len2;

#ifdef DEBUG
    fprintf(debug_fd, "infilename = %s\n", infilename);
    fprintf(debug_fd, "length = %d\nlength_input = %d\n", length,
length_input);
    fclose(debug_fd);
#endif

    if (length > (((length_input/10)>> 10) << 10))
    {
        jpeg_finish_decompress(&srcinfo);
        jpeg_destroy_decompress(&srcinfo);
        fclose(input_file);
        return 4;
    }
```

```
    tail = length * 8 - TAIL1;
    tail_on = 0;

    // fprintf(stderr,"length %d\n",length);
    if(number_foto)
    {
        if ((my_f = fopen(outfilename, "ab+")) == NULL)
        {
            jpeg_finish_decompress(&srcinfo);
            jpeg_destroy_decompress(&srcinfo);

            fclose(input_file);
            return 5;
        }
    }
    else
        if ((my_f = fopen(outfilename, "wb")) == NULL)
        {
            jpeg_finish_decompress(&srcinfo);
            jpeg_destroy_decompress(&srcinfo);

            fclose(input_file);
            return 5;
        }

    count = 0;
    while(count < length)
    {
        for(j = 0; j < 8; j++)
        {
            if ((b = get_bit()) < 0)
            {
                jpeg_finish_decompress(&srcinfo);
                jpeg_destroy_decompress(&srcinfo);
                fclose(my_f);
                remove(outfilename);
                fclose(input_file);
                return 6;
            }
            b ^= get_code_bit(1);
            v = v << 1;
            v |= b;
            tail--;
        }
                if(index_buf == LEN_BUF)
                {
                        fwrite(my_buffer, sizeof(char), LEN_BUF,
my_f);
                        write_len += LEN_BUF;
```

```
                        index_buf = 0;
                }
                my_buffer[index_buf] = v;
                index_buf++;
        count++;
    }
        if (index_buf)
                fwrite(my_buffer, sizeof(char), index_buf, my_f);
        write_len = index_buf;

    jpeg_finish_decompress(&srcinfo);
    jpeg_destroy_decompress(&srcinfo);

    fclose(input_file);

    fclose(my_f);
    if ((number_foto + 1) == num_foto)
    {
        number_foto++;
        return 100;
    }
    else
    {
        if((num_foto == 0)&&(number_foto == 0))
        {

            remove(outfilename);
            return 7;
        }
        if((num_foto == 0)||(num_foto == 255))
        {
            number_foto = 0;
            return 0;
        }
        else
        {
            remove(outfilename);
            return 7;
        }
    }

}

int main(int argc, char* argv[]){
    DIR * hided_dir;
    if(argc != 3){
        printf("Invalid count args");
        return 1;
    }
```

```
    int seek_counter = 0;
    struct dirent *tmp=NULL;
    string name;
    vector<string> stego_file;
    string key = argv[2];
    int res;
    hided_dir = opendir(argv[1]);
    if(NULL == hided_dir){
        printf("Can't open dir = %s", argv[1]);
        return 1;
    }
    string name_file;
    mkdir("SEEK_FOLDER/",S_IRWXU);
    while((tmp=readdir(hided_dir))){
        name=string(tmp->d_name);
        if(name=="." || name==".." || !(name.find(".JPG")!
=string::npos || name.find(".jpg")!=string::npos))
            continue;
        stego_file.push_back(string(argv[1]) + name);
    }
    sort(stego_file.begin(), stego_file.end());
    for(int i = 0; i < stego_file.size(); i++){
        name_file = string("SEEK_FOLDER/SEEK_") +
to_string(seek_counter) + string(".JPG");
        res = jpseek(strdup(stego_file[i].c_str()),
strdup(name_file.c_str()), strdup(key.c_str()));
        cout << "Path size = "  << key.size() << endl;
        cout << "res : "  << res << endl;
        if (res == 0){
            //name_file = "SEEK_FOLDER/SEEK_" + seek_counter + ".jpg";
            seek_counter++;
         }
         else if(res != 100)
            cout << error_jpseek(res, strdup(stego_file[i].c_str()),
strdup(name_file.c_str())) << endl;
    }
    return 0;
}

string error_jpseek(int er, string input_file, string output_file)
{
    string out_str = "";
    switch (er)
    {
        case 1:
            out_str = "Не возможно открыть входной файл " + input_file
+
                "\nУбедитесь что файл существует\n";
            break;
```

```
        case 2:
            out_str = "Не подходящий входной файл " +
                input_file+"\n";
            break;
        case 3:
            out_str = "Проверьте правильность введенного ключа\n";
            break;
        case 4:
            out_str = "Входной файл " + input_file + " не содержит
информации.\nПроверьте правильность введенного ключа\n";
            break;
        case 5:
            out_str = "Не возможно создать файл " + output_file + "\
nПроверьте свободное место на диске\n";
            break;
        case 6:
            out_str = "Проверьте правильность введенного ключа\n";
            break;
        case 7:
            out_str = "Не корректная нумерация в файле\nДобавьте не
достающие файлы в список\n";
            break;
    } //switch
    return out_str;
}
```