

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.02 –

Специализация –

Информационная безопасность
телекоммуникационных систем
Разработка защищенных
телекоммуникационных систем

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, РЕАЛИЗУЮЩЕЕ ШИФРОВАНИЕ И
СКРЫТИЕ СООБЩЕНИЙ С ПОМОЩЬЮ СТЕГАНОГРАФИЧЕСКОГО
МЕТОДА

Студент	_____	<u>Свинкин Роман Игоревич</u>
Руководитель	_____	<u>Кулагин О.В.</u>
Нормоконтролёр	_____	<u>Фатеев А.Г.</u>
Рецензент	_____	<u>Афанасьев А.С.</u>

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ С.Л. Зефилов

Работа защищена с отметкой _____ (протокол заседания ГЭК
от _____ № _____)

Секретарь ГЭК _____ А.Г. Фатеев

Пенза 2016

РЕФЕРАТ

Отчет содержит 162 с., 52 рис., 2 табл., 3 приложения.

СТЕГАНОГРАФИЯ, СТЕГАНОГРАФИЧЕСКИЙ КОНТЕЙНЕР, КРИПТОГРАФИЯ, ЗАШИФРОВАНИЕ, РАСШИФРОВАНИЕ, КЛЮЧ, КЛЮЧЕВОЕ ЗАПОМИНАЮЩЕЕ УСТРОЙСТВО, СКРЫТИЕ СООБЩЕНИЯ, ИЗВЛЕЧЕНИЕ СООБЩЕНИЯ

Объектом разработки является программное обеспечение, реализующее шифрование и скрытие сообщений с помощью стеганографического метода.

Целью работы является разработка программного обеспечения, реализующего шифрование и скрытие сообщений с помощью стеганографического метода.

В процессе выполнения дипломного проекта был произведен обзор цифровых стеганографических методов скрытия информации, разработаны алгоритмы и реализованы программы, осуществляющие зашифрование со скрытием и извлечение с расшифрованием информации одним из стеганографических методов, была проведена экспериментальная проверка функционирования программ.

Результатом выполнения дипломного проекта является программное обеспечение, реализующее защиту передаваемых сообщений, включающее в себя программы, осуществляющие зашифрование со скрытием и извлечение с расшифрованием информации одним из стеганографических методов.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

УДК 004.056.55

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Выпускающая кафедра:
Информационная безопасность систем
и технологий

Учебная группа 11ПТ1

УТВЕРЖДАЮ
Зав. кафедрой ИБСТ
к.т.н., доцент С.Л. Зефирова

«___» _____ 2016 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ
РАБОТУ**

студента группы 11ПТ1 Свинкина Романа Игоревича

**Специальность 10.05.02 – Информационная безопасность
автоматизированных систем**

Тема дипломного проекта: Программное обеспечение, реализующее
шифрование и скрытие сообщений с помощью стеганографического метода.

Руководитель ВКР

Инженер I категории АО «ПНИЭИ», к.т.н.

Кулагин О.В.

Нормоконтролер

к.т.н., доцент

Фатеев А.Г.

1 Цели и задачи ВКР

Цель работы – разработка программного обеспечения, реализующего шифрование сообщений с последующим скрыванием с помощью стеганографического метода, и извлечение с последующим расшифрованием.

2 Тактико-технические требования к выполнению ВКР

Технические требования:

- программное обеспечение должно быть реализовано в виде двух программ на языке программирования C++;
- передаваемое сообщение должно быть сначала зашифровано, а затем встроено в контейнер *.wav методом стеганографии LSB для последующей передачи с помощью первой программы;
- передаваемое сообщение должно быть зашифровано согласно алгоритму ГОСТ 28147-89 в режиме простой замены (блочного шифрования);
- на вход первой программы должны поступать текст передаваемого сообщения, ключевая информация и *.wav-файл в качестве контейнера для скрывания;
- на выходе первой программы должен быть *.wav-файл содержащий в себе скрытое зашифрованное сообщение;
- зашифрованное и скрытое сообщение должно быть извлечено и расшифровано при помощи второй программы;
- на вход второй программы должен поступать *.wav-файл, содержащий в себе скрытое зашифрованное сообщение и ключевая информация;
- на выходе второй программы должен быть текст передаваемого сообщения, идентичный тому, что поступал на вход первой программы.

В ВКР должны быть рассмотрены следующие вопросы:

- обзор цифровых стеганографических методов скрытия информации;
- разработка и описание алгоритма программы, реализующей шифрование и скрытие сообщений;
- разработка программы, реализующей шифрование и скрытие сообщений;
- разработка и описание алгоритма программы, реализующей извлечение и расшифрование скрытых сообщений;
- разработка программы, реализующей извлечение и расшифрование сообщений.

3 Требования к разрабатываемой документации

Отчет по ВКР выполняется в соответствии с требованиями ГОСТ 7.32.

Перечень и содержание графической части ВКР:

- криптосхема алгоритма зашифрования в режиме простой замены по ГОСТ 28147-89 – 1л., ф. А1 (плакат);
- криптосхема алгоритма расшифрования в режиме простой замены по ГОСТ 28147-89 – 1л., ф. А1 (плакат);
- структура *.wav-файла как контейнера – 1л., ф. А1 (плакат);
- алгоритм стеганографического скрытия информации – 1л., ф. А1 (плакат);
- алгоритм извлечения скрытой информации из контейнера – 1л., ф. А1 (плакат);
- демонстрация работы программного обеспечения.

Электронная копия отчета и графической части ВКР должна быть представлена на магнитном или оптическом носителе.

4 Сроки выполнения ВКР

Тема утверждена приказом ректора ПГУ № от «___»_____ 201_г.

Дата выдачи задания «___» _____ 201_г.

Время дипломного проектирования с _____ 201_г. по _____ 201_г.

Задание к исполнению принял «___» _____ 201_г.

Исполнитель ВКР _____ Свинкин Р.И.

СОДЕРЖАНИЕ

Введение.....	10
1 Обзор цифровых стеганографических методов скрытия информации.....	11
2 Структура файла WAV формата	23
3 Реализация программного обеспечения.....	28
3.1 Общее описание.....	28
3.2 Разработка и описание алгоритма программы, реализующей шифрование и скрытие сообщений.....	28
3.3 Разработка программы, реализующей шифрование и скрытие сообщений.....	48
3.3.1 Описание начальных переменных.....	48
3.3.2 Функция открытия и чтения файла ключа	52
3.3.3 Функция формирования КЗУ.....	54
3.3.4 Функция открытия файла открытого текста.....	56
3.3.5 Функция чтения и подготовки к шифрованию открытого текста.....	57
3.3.6 Функция шифрования.....	60
3.3.7 Функция подготовки зашифрованного текста к скрытию.....	67
3.3.8 Функция открытия файла контейнера.....	70
3.3.9 Функция подсчета необходимой информации по стеганографическому контейнеру.....	75
3.3.10 Функция чтения байт данных стеганографического контейнера.....	77
3.3.11 Функция определения последовательности байт стеганографического контейнера для скрытия	78
3.3.12 Функция скрытия зашифрованного текста и создания выходного файла.....	81
3.4 Разработка и описание алгоритма программы, реализующей извлечение и расшифрование скрытых сообщений.....	83

3.5 Разработка программы, реализующей извлечение и расшифрование сообщений.....	100
3.5.1 Описание начальных переменных.....	100
3.5.2 Функция открытия файла стеганографического контейнера.....	102
3.5.3 Функция подсчета необходимой информации по стеганографическому контейнеру.....	106
3.5.4 Функция чтения байт данных стеганографического контейнера.....	106
3.5.5 Функция определения последовательности байт стеганографического контейнера для извлечения.....	107
3.5.6 Функция извлечения зашифрованного текста.....	110
3.5.7 Функция открытия и чтения файла ключа.....	113
3.5.8 Функция формирования КЗУ.....	115
3.5.9 Функция подготовки зашифрованного текста к расшифрованию.....	116
3.5.10 Функция расшифрования зашифрованного текста.....	118
3.5.11 Функция создания выходного файла и записи в него расшифрованного текста.....	123
4. Экспериментальная проверка работоспособности программ.....	125
4.1 Проверка работоспособности программы зашифрования и скрытия сообщения	125
4.2 Проверка работоспособности программы извлечения и расшифрования сообщения	126
4.3 Реакция на ошибки.....	128
Заключение.....	133
Список использованных источников.....	134
Приложение А Код программы, реализующей шифрование и скрытие сообщений.....	135
Приложение Б Код программы, реализующей извлечение и расшифрование сообщений.....	150

Приложение В Протокол проверки ВКР на оригинальность в системе «Антиплагиат.ВУЗ».....	162
--	-----

ВВЕДЕНИЕ

В современную эпоху цифровых устройств и технологий информация – главный её ресурс. Информация окружает людей и сопровождает повсюду. Соответствующая конфиденциальная информация должна находиться в надежных руках и подлежать надежной защите. Ведь обнародование такой информации или попадание в руки злоумышленника (конкурентов, преступников, мошенников, террористических организаций или вражеской стороны) способно нанести огромный и непоправимый ущерб как на локальном, так и на государственном и даже мировом уровне. Это утверждение становится более значимым в условиях современной циркуляции информации в мире от отправителей к получателям. Ведь на пути следования информация может быть перехвачена, распознана, искажена, или уничтожена, что недопустимо.

Одним из способов защитить передаваемую информацию в недоверенной среде от лиц, доступ которым к ней не предусмотрен, является скрытие самого факта передачи данной информации. Этим занимается стеганография.

Данный проект посвящен разработке программного обеспечения, реализующего защиту передаваемых сообщений путем зашифрования и скрытия с помощью одного из стеганографических методов.

1 Обзор цифровых стеганографических методов скрытия информации

Целью стеганографии является скрытие факта передачи информации. Информация скрывается в непримечательный, не привлекающий внимания объект, и в открытом виде пересылается адресату. Наличие скрытой связи остается незаметным. Области применения стеганографии представлены на рисунке 1[1].



Рисунок 1 – Области применения стеганографии

Компьютерная стеганография — направление классической стеганографии, основанное на особенностях компьютерной платформы.

Примером компьютерной стеганографии могут служить:

- использование зарезервированных полей компьютерных форматов файлов;
- метод скрытия информации в неиспользуемых местах гибких дисков;

- метод использования особых свойств полей форматов, которые не отображаются на экране;
- использование особенностей файловых систем.

В последнее время приобрели популярность методы, когда скрытая информация передается через компьютерные сети с использованием особенностей работы протоколов передачи данных. Такие методы получили название «сетевая стеганография». Типичные методы сетевой стеганографии включают изменение свойств одного из сетевых протоколов. Кроме того, может использоваться взаимосвязь между двумя или более различными протоколами с целью более надежного скрытия передачи секретного сообщения.

Сетевая стеганография охватывает широкий спектр методов, в частности:

- WLAN стеганография основывается на методах, которые используются для передачи стеганограмм в беспроводных сетях (Wireless Local Area Networks). Практический пример WLAN стеганографии — система HICCUPS (Hidden Communication System for Corrupted Networks);
- LACK (Lost Audio Packets) стеганография — скрытие сообщений во время разговоров с использованием IP-телефонии. Например: использование пакетов, которые задерживаются, или намеренно повреждаются и игнорируются приемником или скрытие информации в полях заголовка, которые не используются [2].

В настоящее время методы компьютерной стеганографии развиваются по двум основным направлениям:

- методы, основанные на использовании специальных свойств компьютерных форматов;
- методы, основанные на избыточности аудио и визуальной информации.

В таблице 1 представлены сравнительные характеристики стеганографических методов [3].

Таблица 1 - Сравнительные характеристики стеганографических методов

Метод	Характеристика	Недостатки	Преимущества
Методы использования специальных свойств компьютерных форматов данных			
Методы использования зарезервированных для расширения полей компьютерных форматов данных	Поля расширения имеются во многих мультимедийных форматах, они заполняются нулевой информацией и не учитываются программой	Низкая степень скрытности, передача небольших объемов информации	Простота использования
Методы специального форматирования текстовых файлов			
Методы использования известного смещения слов, предложений, абзацев	Методы основаны на изменении положения строк и расстановки слов в предложении, что обеспечивается вставкой дополнительных пробелов между словами	Слабая производительность метода, передача небольших объемов информации Низкая степень скрытности	Простота использования.
Методы выбора определенных позиций букв (нулевой шифр)	Акростих - частный случай этого метода (например, начальные буквы каждой строки образуют сообщение)		
Методы использования специальных свойств полей форматов, не отображаемых на экране	Методы основаны на использовании специальных "невидимых", скрытых полей для организации сносок и ссылок (например, использование черного шрифта на черном фоне)		
Методы скрытия в неиспользуемых местах гибких дисков	Информация записывается в обычно неиспользуемых местах ГМД (например, в нулевой дорожке)	Слабая производительность метода, передача небольших объемов информации Низкая степень скрытности	Простота использования.
Методы использования имитирующих функций (mimic-function)	Метод основан на генерации текстов и является обобщением акростиха. Для тайного сообщения генерируется осмысленный текст, скрывающий само сообщение	Слабая производительность метода, передача небольших объемов информации Низкая степень скрытности	Результирующий текст не является подозрительным для систем мониторинга сети
Методы удаления идентифицирующего файл заголовка	Скрываемое сообщение шифруется и у результата удаляется идентифицирующий заголовок, оставляя только зашифрованные данные. Получатель заранее знает о передаче сообщения и имеет недостающий заголовок	Проблема скрытия решается только частично. Необходимо заранее передать часть информации получателю	Простота реализации.
Методы использования избыточности аудио и визуальной информации			
Методы использования избыточности цифровых фотографий, цифрового звука и цифрового видео	Младшие разряды цифровых отсчетов содержат очень мало полезной информации. Их заполнение дополнительной информацией практически не влияет на качество восприятия, что и дает возможность скрытия конфиденциальной информации	За счет введения дополнительной информации искажаются статистические характеристики цифровых потоков. Для снижения компрометирующих признаков требуется коррекция статистических характеристик	Возможность скрытой передачи большого объема информации. Возможность защиты авторского права, скрытого изображения товарной марки, регистрационных номеров и т.п.

Метод замены цветовой палитры.

Данный метод предназначен для скрытия текстовой информации в графических файлах, использующих цветовые палитры. В качестве контейнера, в данном случае, следует выбирать файлы, содержащие некоторый цвет в избытке. Например, это могут быть схемы, рисунки или отсканированный текст черного цвета на белом фоне. Фон, как правило, занимает большую часть изображения, т. е. имеется избыток белого цвета.

Вначале создаётся алфавит, содержащий символы, которые используются в файле сообщения, например все 33 буквы русского алфавита от А до Я, цифры от 0 до 9, специальные знаки и знаки пунктуации – всего 51 символ:

Код	0	1	2	3	4	...	49	50
Символ	А	Б	В	Г	Д	...	%	;

Далее проводится замена цветов палитры. Для этого первому 51 цвету палитры назначается черный цвет (цвет рисунка). Следующему 51 цвету палитры назначается белый цвет (цвет фона). Если рисунок содержит 3...5 цветов, то можно переназначить следующие группы по 51 цвету в соответствующие цвета. Изменённая палитра цветов будет иметь в шестнадцатеричном представлении следующий вид:

Черный цвет

Код	0	1	2	...	49	50
Цвет	00 00 00	00 00 00	00 00 00	...	00 00 00	00 00 00

Белый цвет

Код	51	52	53	...	100	101
Цвет	FF FF FF	FF FF FF	FF FF FF	...	FF FF FF	FF FF FF

Черный цвет в данном случае имеет нулевой уровень, что для одного байта соответствует 0 в десятичной системе счисления, «0000 0000» в двоичной и «00» в шестнадцатеричной, а белый цвет имеет уровень 255, что соответствует 255 в десятичной, «1111 1111» в двоичной и «FF» в шестнадцатеричной системе счисления. Для скрытия информации берётся первая точка изображения, анализируется её принадлежность к определённой

цветовой группе, например к группе чёрного цвета, затем этой точке присваивается код текущего символа из файла-сообщения с учётом выбранной цветовой группы. Например, для символа Б чёрной точке будет назначен цвет с кодом 1, а белой точке – цвет с кодом 52 [4].

При применении данного метода к двухцветным контейнерам объем скрываемой информации равен количеству точек (произведение ширины на высоту). Это один из самых емких методов скрытия информации в графических файлах, который позволяет оставлять картинку без изменения качества и объема. Его можно использовать для любого алфавита с числом символов не более 128. Однако данный метод достаточно легко выявляется средствами программного стеганоанализа [5].

Метод сортировки палитры.

В данном методе, в качестве контейнера используются файлы с индексированными цветами, содержащие монохромное изображение. Суть метода заключается в предварительной подготовке файла-контейнера. Палитра файла упорядочивается таким образом, чтобы цвета с соседними номерами минимально отличались друг от друга и равномерно изменялись от черного до белого цвета. После этого скрываемое сообщение привносится в младшие биты точек изображения, то есть искажаются номера цветов, а не сами цвета, но благодаря предварительно отсортированной палитре цвет точки заменяется на похожий. Следует отметить, что данный метод также достаточно легко выявляется средствами программного анализа.

Методы компьютерной стеганографии в JPEG-файлах.

Цветные изображения, представленные в цифровой форме, достаточно велики и занимают большой объём памяти (до нескольких мегабайт). Поэтому для их сжатия существует ряд методов. Так, форматы BMP и GIF используют алгоритмы сжатия без потерь, обеспечивающие точное восстановление исходного изображения. Существуют также алгоритмы сжатия с потерей (искажением) информации. Таким примером может служить формат JPEG (Joint Photographic Experts Group) [6].

В общем случае методы обработки (сжатия) изображений можно разделить на две группы: непосредственные и спектральные. При использовании непосредственных методов обработке подвергаются сами исходные изображения (пиксели). Спектральные методы основаны на применении дискретных унитарных преобразований Фурье, Адамара и др. При этом обрабатывается не исходное изображение, а соответствующие коэффициенты преобразования.

Алгоритм сжатия JPEG состоит из следующих этапов:

- преобразование изображения в оптимальное цветовое пространство;
- субдискретизация компонентов цветности посредством их усреднения;
- применение дискретных косинусных преобразований (разновидность преобразований Фурье) для уменьшения избыточности данных изображения;
- квантование коэффициентов преобразования с применением весовых функций, оптимизированных с учётом физиологических особенностей зрения;
- кодирование данных изображения (результатирующих коэффициентов) для удаления избыточности информации с применением алгоритма Хаффмана.

Возможность использования файловых форматов, построенных по схеме сжатия JPEG, для скрытия информации обусловлена их широким распространением при хранении и передаче графических изображений, в частности в сети Internet.

В JPEG-файлах могут использоваться следующие методы скрытия информации:

- дописывание данных скрываемой информации в конец файла;
- скрытие информации в косвенных данных файла;

- скрывание информации с использованием таблиц квантования;
- скрывание информации между блоками данных файла.

Методы скрывания информации в JPEG-файлах обладают достаточно высокой степенью защищённости от стегоанализа, так как возможность варьирования качества сжатого изображения в широком диапазоне не позволяет легко установить, являются ли возникающие в результате сжатия погрешности следствием скрывания данных или следствием использования высоких коэффициентов квантования.

Метод эхо – стеганографии.

Этот метод основан на особенностях функционирования человеческого уха, устойчив к воздействию типа фильтрации, повторной оцифровки, блочного редактирования, сжатия с потерей качества на аудио файл, содержащий скрытую информацию. При использовании метода эхо-стеганографии в сигнал вводятся изменения, которые ближе к естественным условиям воспроизведения, чем к случайному шуму, поэтому этот метод устойчив к воздействию алгоритмов сжатия с потерей данных. Каждое вносимое искажение, по отдельности, похоже на резонансы от стен, фурнитуры и т.п. Различие между сигналом-контейнером и стего-сигналом примерно такое же, как между прослушиванием компакт диска через наушники и прослушиванием его же через динамики. При прослушивании через звук слышен таким, как он был записан. А при прослушивании через динамики слышен записанный звук плюс эхо, обусловленное акустикой помещения. При корректном выборе параметров вводимого искажения можно получить сигнал, неотличимый от вышеизложенного примера с динамиками.

Метод эхо-стеганографии помещает сообщение в аудиофайл-контейнер с помощью вводимого «эха». Цифровые отсчеты определяются использованием четырех главных параметров эха: начальная амплитуда, величина ослабления, сдвиг двоичной единицы, сдвиг двоичного нуля (сдвиг двоичной единицы плюс дельта). Амплитуда эхо-сигнала – амплитуда эха при кодировании единичного

бита, величина ослабления – амплитуда эха при кодировании нулевого бита, дельта-сдвиг эхосигнала относительно сдвига при кодировании нулевого бита относительно сдвига при кодировании единичного бита. При уменьшении сдвига (задержки) между оригиналом и эхом, сигналы смешиваются. В некоторый момент человеческое ухо перестает различать оригинальный сигнал и его эхо, а воспринимает один искаженный сигнал. Определить данное значение задержки достаточно тяжело. Оно зависит от качества записи оригинала, типа звука, подвергнутого эхообразованию и от слушателя. В основном, данный эффект наступает при задержке в одну тысячную секунды для большинства звуков и слушателей.

Особенность применения данного метода состоит в том, что звуковые данные такого рода можно не только передавать в виде файлов, но и по радиоканалам с хорошим качеством, например в FM диапазоне. При записи аудиосигнала из эфира возможна расшифровка данных с помощью спектрального анализатора. Такой метод хорошо подходит при непрерывной передаче данных в эфир или в глобальную сеть [5].

Фазовое кодирование.

Данный метод заключается в замене фазы исходного звукового сегмента на опорную фазу, представляющую собой секретную информацию. Остальные сегменты фазы корректируются для сохранения определенной фазы между сегментами. С точки зрения отношения сигнала к шуму, фазовое кодирование является одним из наиболее эффективных методов кодирования. Когда происходит резкое изменение фазового соотношения между каждой частотной составляющей, шумы становятся заметными. В случае слабой модификации фазы изменения не смогут быть распознаны на слух. Фазовое кодирование происходит в несколько этапов:

- исходный аудиосигнал делится на более мелкие сегменты;
- с помощью дискретного преобразования Фурье создается матрица фаз;

- вычисляется разность фаз между соседними сегментами;
- в стегофайле сохраняются разности фаз, секретное сообщение встраивается только в фазу первого сегмента;
- с помощью новой фазы первого сегмента создается новая матрица фаз и разницы между ними;
- с помощью обратного дискретного преобразования Фурье с использованием исходной и новой матрицы величин сегменты аудиосигнала восстанавливаются;
- соединение сегментов.

Получатель должен знать длину сегмента, чтобы извлечь секретное сообщение из звукового файла с помощью дискретного преобразования Фурье.

Фазовое кодирование имеет основной недостаток, заключающийся в низкой скорости передачи данных из-за того, что секретное сообщение кодируется только на первом сегменте сигнала. Следовательно, этот метод используется только тогда, когда передается небольшое количество данных [7].

Метод замены наименьших значащих бит LSB (Least Significant Bits).

Данный метод стеганографии представляет собой метод при котором скрываемое сообщение (информация) разбивается на отдельные биты, которые в последующем, должны быть вставлены вместо младших бит в байтах файла, в котором планируется скрыть сообщение [8]. Для изображений это байты кодирующие цвет пикселей (точек), совокупность которых образует изображение. Так как заменяются именно младшие биты, изменения в цвете между исходным изображением и изображением в котором скрыто сообщение данным методом минимальны и являются неотличимыми для человеческого глаза. Число младших бит скрываемого сообщения в каждом байте изображения может быть разным, однако чем больше бит в каждом байте исходного изображения подлежит замене, тем сильнее изображение искажается визуально. Таким образом, наилучшим способом скрытия сообщения с точки зрения незаметности является замена как можно меньшего числа бит (1 – 2

бита) в байтах исходного изображения. Принцип метода LSB для изображений показан на рисунке 2.

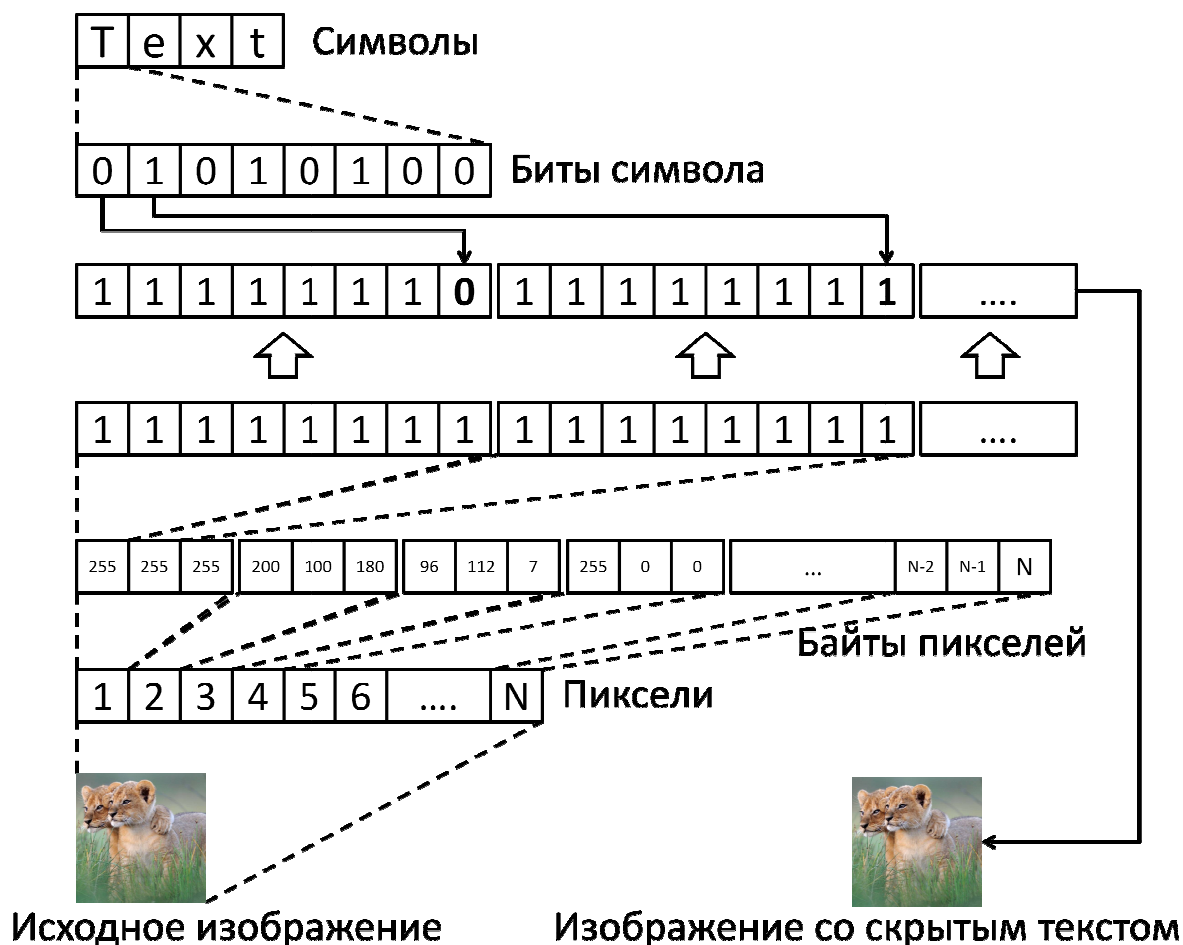


Рисунок 2 – Принцип метода LSB для изображений

Кроме того, число заменяемых бит напрямую влияет на вместимость изображения. Чем больше бит в каждом байте заменяется, тем больше вместимость изображения. Например, максимальная длина скрываемого сообщения в символах для 24-битного bmp изображения рассчитывается по формуле:

$$\text{Длина скрываемого сообщения} = \frac{a \times 3 \times \text{размер изображения}}{8}$$

, где:

- a – число заменяемых бит в каждом байте;

– размер изображения – число пикселей, содержащихся в изображении.

Сведение числа заменяемых бит в байтах исходного файла к минимальному минимизирует длину сообщения, которое туда способно уместиться, но делает скрытность для человеческого глаза максимальной.

Метод LSB для звуковых файлов аналогичен методам скрытия информации в графических файлах. Кроме этого возможен метод полной замены блока данных файла-контейнера на блок данных файла-сообщения. Применительно к звуковым файлам, этот метод становится актуален, благодаря тому что при попытке выявления файлов-контейнеров с привнесенной информацией из всего потока передаваемых данных, зачастую невозможно прослушать все файлы по причине больших временных затрат в отличие от графических изображений. Программа-анализатор также не выявит нарушений в структуре файла.

Данный метод не изменяет размер файла-контейнера и позволяет сохранять достаточно большой объем информации [5].

Основным недостатком данного метода является то, что он является слабоустойчивым к атакам и может быть использован только при отсутствии шума в канале передачи данных. Применение данного метода ограничено форматами, использующими сжатие без потери качества. Иначе говоря, файл со скрытой информацией должен передаваться в оригинале без малейших искажений для того, чтобы получатель на приемной стороне мог раскрыть предназначенное ему послание. Обнаружение наличия LSB осуществляется по аномальным характеристикам распределения значений диапазона младших битов отсчетов цифрового сигнала [5].

Метод расширенного спектра.

В аудио стеганографии метод расширенного спектра пытается передать секретные сведения по спектру частот звукового сигнала. Этот метод схож с методом LSB, который передает биты сообщения случайным образом по всему звуковому файлу. Тем не менее, в отличие от способа LSB, метод

расширенного спектра распространяет секретную информацию по спектру частот звукового файла, используя код, который не зависит от фактического сигнала. В результате конечный сигнал занимает полосу пропускания, которая размером больше, чем требуемый размер для передачи.

Метод расширенного спектра может внести вклад в повышение производительности по сравнению с методами LSB, фазового и четного кодирования путем умеренной скорости передачи данных и высоким уровнем устойчивости. Однако, метод расширенного спектра имеет один существенный недостаток – он может вносить шум в аудиофайл [7].

2 Структура файла WAV формата

Для реализации защиты передаваемой информации была изучена структура формата wav файла.

Waveform audio file format (Wav/Wave) – формат файла-контейнера для хранения записи оцифрованного аудиопотока (является подвидом RIFF формата для хранения потоковых мультимедийных данных). Данный контейнер, как правило, используется для хранения несжатого звука в формате импульсно-кодовой модуляции (pulse-code modulation – PCM).

Типичная структура wav-файла представлена на рисунке 3.

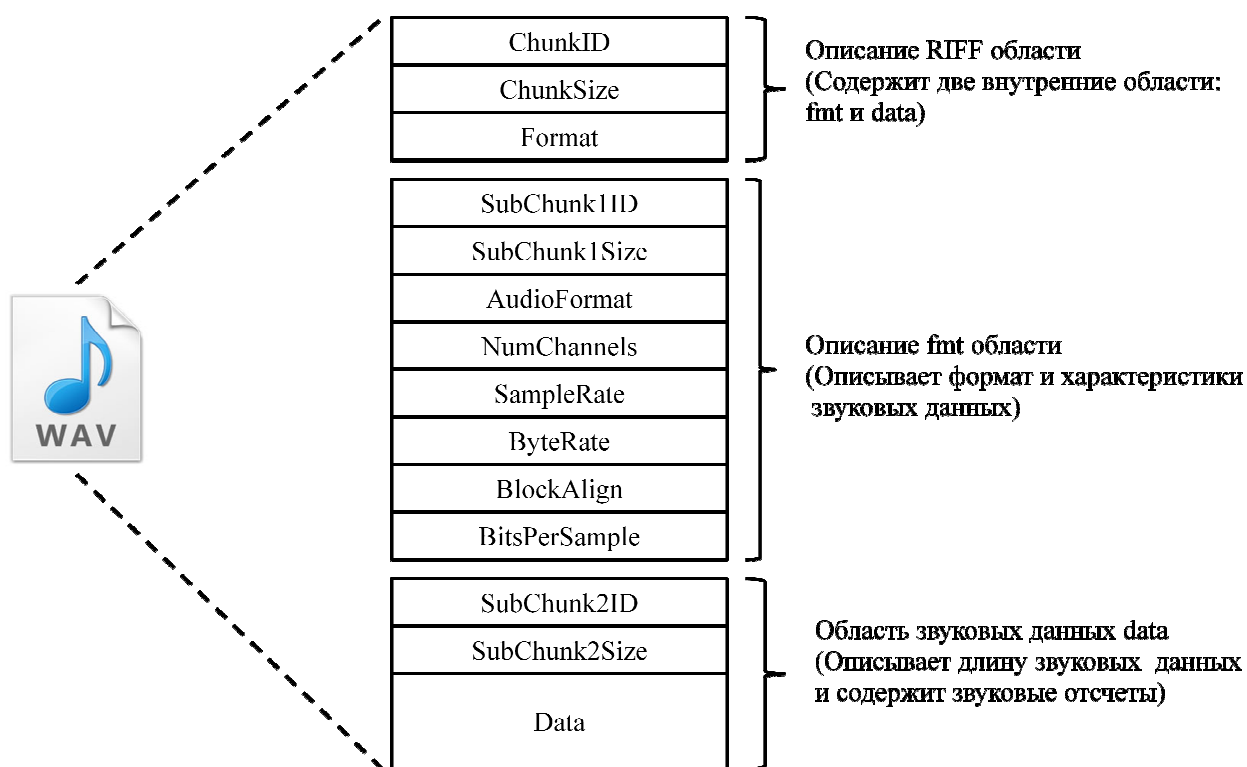


Рисунок 3 – Структура файла формата wav

Данная структура содержит последовательность полей, определяющих основные характеристики файла. Их описание приведено в таблице 2.

Файл формата wav может содержать дополнительные, редко используемые поля (информация об исполнителях, теги и т.п.). Как правило, данные поля располагаются перед полем SubChunk2ID либо после байт звуковых отсчетов (в конце файла). При работе с wav-файлом, необходимо

учитывать важную особенность, заключающуюся в том, что порядок байт полей структуры в файле может быть разным.

Таблица 2 – Описание полей структуры файла формата wav

Порядок байт	Смещение (в байтах)	Длина (в байтах)	Название поля	Описание
Прямой	0	4	ChunkID	Сигнатура формата RIFF. Может содержать единственное значение 0x52494646 (символы “RIFF”). Является началом RIFF области.
Обратный	4	4	ChunkSize	Оставшийся размер последовательности байт за исключением полей ChunkID и ChunkSize (размер файла минус 8 байт).
Прямой	8	4	Format	Сигнатура формата wav. Может содержать единственное значение 0x57415645 (символы “WAVE”).
Прямой	12	4	SubChunk1ID	Сигнатура области fmt. Может содержать единственное значение 0x666d7420 (символы “fmt”).
Обратный	16	4	SubChunk1Size	Оставшийся размер области fmt.
Обратный	20	2	AudioFormat	Формат кодирования аудиоданных. Для РСМ = 1 (Линейное квантование). Значения, отличающиеся от 1, обозначают некоторый формат сжатия.
Обратный	22	2	NumChannels	Количество каналов.
Обратный	24	4	SampleRate	Частота дискретизации, Гц.
Обратный	28	4	ByteRate	Количество байт, переданных за секунду воспроизведения.
Обратный	32	2	BlockAlign	Количество байт для одного звукового отсчета, включая все каналы.
Обратный	34	2	BitsPerSample	Количество бит, отводимых для хранения одного звукового отсчета.
Прямой	36	4	SubChunk2ID	Сигнатура области data. Может содержать единственное значение 0x64617461 (символы “data”).
Обратный	40	4	SubChunk2Size	Количество байт, отводимых для хранения всех звуковых отсчетов.
Обратный	44	-	Data	Байты звуковых отсчетов.

Для каждого поля установлен свой порядок байт (приведен в таблице 2) и может быть прямым (big-endian) и обратным (little-endian).

Байты звуковых отсчетов для одноканального wav – файла в области поля data идут последовательно друг за другом. Для двухканальных файлов байты чередуются: сначала идет байт/байты первого звукового отсчета для левого

канала, потом для правого, после чего идут байты второго звукового отсчета для левого и правого канала и т.д. Аналогично для других многоканальных файлов.

На рисунке 4 представлено распределение байт в области звуковых отсчетов для разных wav-файлов.

		Последовательность байт																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Моно, 8 бит																		...
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета		Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	...
Сtereo, 8 бит																		...
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета																		...
Моно, 16 бит																		...
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета		Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	...
Сtereo, 16 бит																		...
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета																		...
Моно, 24 бита																		...
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
№ звукового отсчета		Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	Л	П	...
Сtereo, 24 бита																		...

Рисунок 4 – Распределение байт в области звуковых отсчетов для разных wav-файлов (Л – левый канал, П – правый канал)

Структура wav-файла как контейнера на примере конкретного файла представлено на рисунке 5. Байты в области, где расположены звуковые отсчеты, представляют собой числа, кодирующие значение амплитуды звука в определенный момент времени. Число и продолжительность таких моментов времени не бесконечно и определяется частотой дискретизации (значение которой содержится в поле SampleRate). Амплитуда может выражаться числом, состоящим из 8, 16, 24 и т.д. бит.

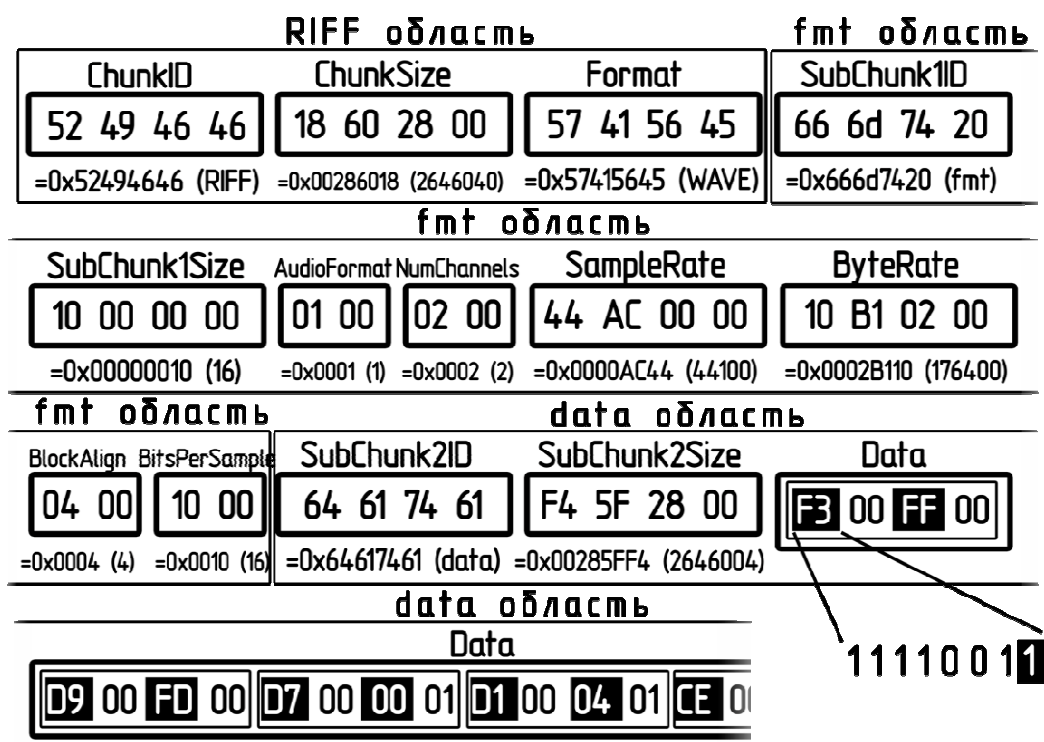


Рисунок 5 – Структура *wav-файла как контейнера

Значение данного числа определяет, сколько бит отводится на хранение звукового отсчета/амплитуды и содержится в поле BitsPerSample. Чем больше байт отводится для хранения звуковых отсчетов, тем более точно можно закодировать звуковую последовательность. Скрытие в файле-контейнере информации методом LSB предполагает изменение только младших бит в байтах звуковых отсчетов, в то время как остальная часть файла сохраняется в оригинале (на рисунке 5 изменяемые байты/биты выделены черным цветом). Это позволяет сделать файл до скрытия и после неотличимыми внешне, а факт того, что изменяются именно младшие биты – неотличимыми на слух.

Максимальный размер информации (в байтах), которую можно скрыть в файле-контейнере wav-формата методом стеганографии LSB определяется по формуле:

$$\text{Вместимость контейнера} = a \times \frac{\text{SubChunk2Size}}{\text{BitsPerSample} \times 8}$$

, где:

- a – число заменяемых бит байте;
- $SubChunk2Size$ – количество байт, отводимых для хранения всех звуковых отсчетов;
- $BitsPerSample$ – количество бит, отводимых для хранения одного звукового отсчета.

3 Реализация программного обеспечения

3.1 Общее описание

В качестве инструмента реализации программного обеспечения была использована среда Microsoft Visual C++ 2008 Express Edition, а в качестве языка программирования был использован язык C++. Программное обеспечение реализовано в виде двух консольных программ, для работы в ОС Windows.

Первая программа Hide_message.exe производит зашифрование сообщения согласно алгоритму ГОСТ 28147–89 в режиме простой замены [9] и его скрывание в файл-контейнер формата wav (с PCM кодировкой) с помощью стеганографического метода LSB с заменой одного младшего бита.

Вторая программа Seek_message.exe производит извлечение из файла-контейнера формата wav, полученного при помощи первой программы зашифрованного сообщения и расшифровывает его.

Минимальный размер текста сообщения, для зашифрования и скрывания сообщения с помощью первой программы – 4 символа. Максимальный размер текста сообщения, для зашифрования и скрывания сообщения с помощью первой программы зависит от файла-контейнера и определяется его размерами (вместимостью). В качестве символов текста скрываемого сообщения могут быть использованы любые символы таблицы ASCII.

3.2 Разработка и описание алгоритма программы, реализующей шифрование и скрывание сообщений

Был разработан алгоритм программы, реализующий шифрование текстовых сообщений с помощью алгоритма ГОСТ 28147–89[9] в режиме простой замены и их скрывание в контейнере *wav-файла стеганографическим методом LSB с заменой одного младшего бита, включающий в себя следующие функции:

- открытие и чтение файла ключа;
- формирование ключевого запоминающего устройства (КЗУ);

- открытие файла открытого текста;
- чтение и подготовка открытого текста к шифрованию;
- шифрование открытого текста;
- подготовка зашифрованного текста к скрыванию;
- открытие файла стеганографического контейнера;
- подсчет необходимой информации по стеганографическому контейнеру;
- чтение байт данных стеганографического контейнера;
- определение последовательности байт стеганографического контейнера для скрывания;
- скрывание зашифрованного текста и создание выходного файла.

Общий алгоритм программы представлен на рисунке 6.

В начале происходит инициализация начальных переменных для шифрования. Затем производится открытие и чтение файла ключа для последующего формирования КЗУ. После того, как КЗУ было сформировано, происходит открытие файла, содержащего открытый текст сообщения для передачи и его предварительная подготовка перед шифрованием. Далее идет непосредственно шифрование открытого текста. Когда текст зашифрован, объявляются начальные переменные для скрывания зашифрованного текста. Производится подготовка зашифрованного текста к скрыванию и открытие файла, содержащего стеганографический контейнер для скрывания. После открытия файла-контейнера происходит подсчет необходимой информации по данному файлу и чтение его байт данных. Когда байты данных файла-контейнера прочитаны, формируется последовательность номеров его байт в области данных для скрывания в них бит зашифрованного сообщения. Далее происходит скрывание бит зашифрованного текста в байтах данных файла контейнера, создается новый файл-контейнер, содержащий в себе скрытое зашифрованное сообщение. Программа выводит на экран сообщение об успешном завершении шифрования и скрывания.

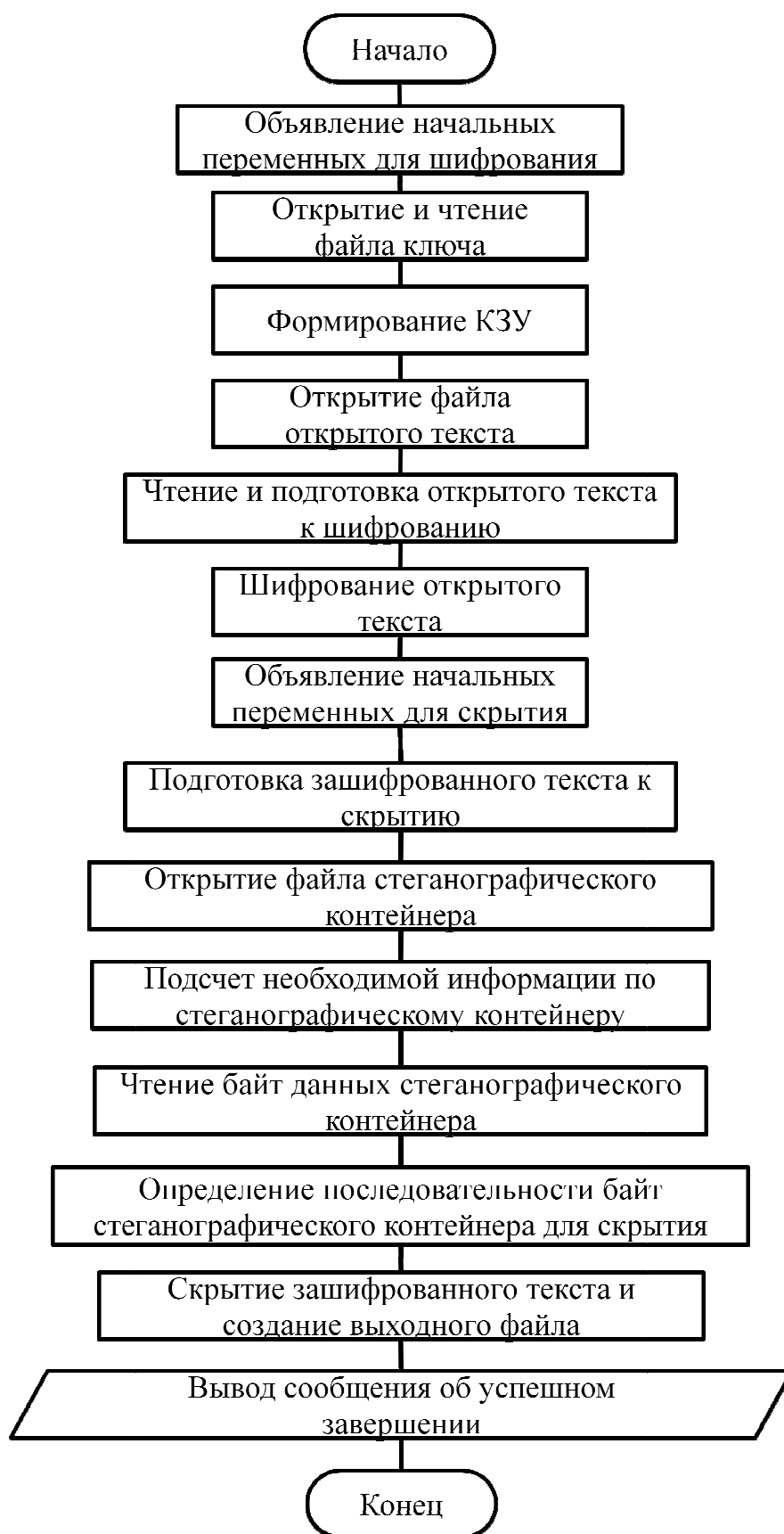


Рисунок 6 – Общий алгоритм программы, реализующей шифрование и скрывание сообщений

На рисунке 7 представлен алгоритм функции открытия и чтения файла ключа.

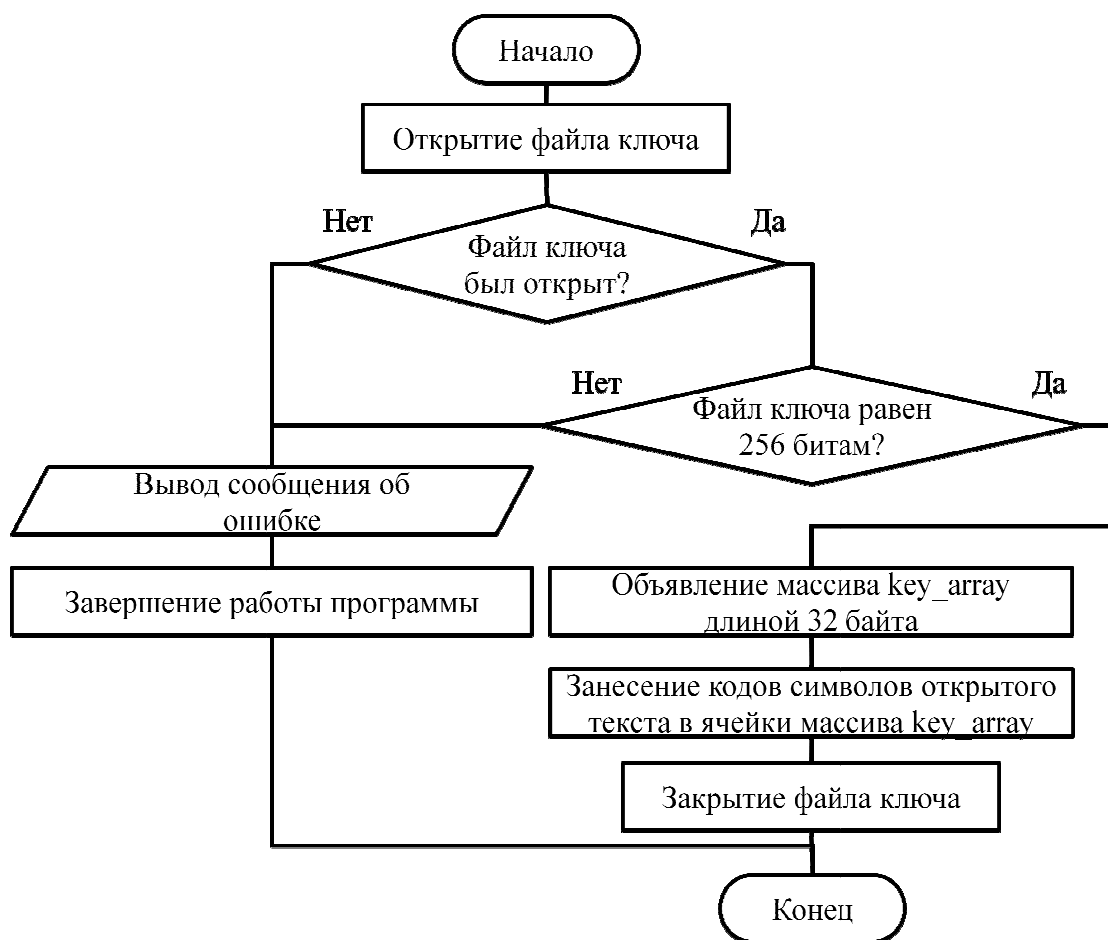


Рисунок 7 – Алгоритм функции открытия и чтения файла ключа

Происходит открытие файла ключа. Если по каким-либо причинам этого не происходит, функция выводит на экран сообщение об ошибке и программа завершает свою работу.

В случае если файл ключа успешно открыт происходит проверка длины ключа. Если ключ не содержит ровно 256 бит, функция выводит на экран сообщение об ошибке и программа завершает свою работу. В противном случае создается массив для хранения кодов символов, содержащихся в файле ключа, куда и происходит занесение данных кодов. После произведенных операций происходит закрытие файла ключа.

Алгоритм функции формирования КЗУ представлен на рисунке 8.

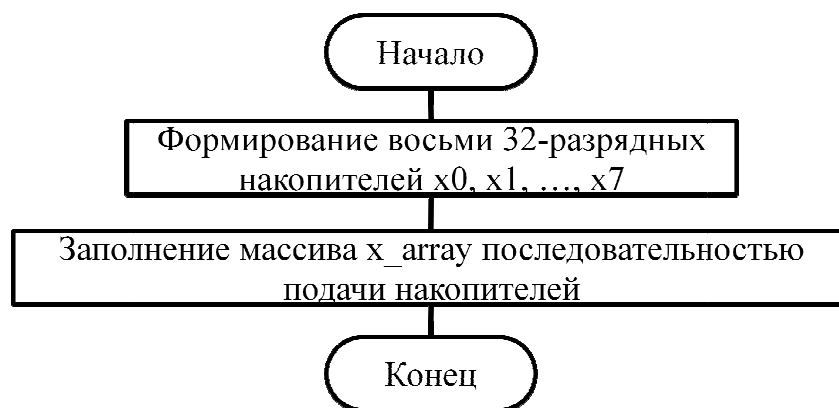


Рисунок 8 - Алгоритм функции формирования КЗУ

Функция распределяет 256 бит ключевой информации по 32 бита в переменные $x_0, x_1, \dots, x_7$. Таким образом, КЗУ представляет собой восемь 32 – разрядных накопителей. Алгоритм шифрования ГОСТ 28147-89[9] в режиме простой замены предполагает работу с накопителями в определенном порядке. В виду этого создается специальный массив x_array , ячейки которого заполняются значениями переменных $x_0, x_1, \dots, x_7$ установленным алгоритмом шифрования в следующем порядке: $X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0$.

Алгоритм функции открытия файла открытого текста представлен на рисунке 9.



Рисунок 9 – Алгоритм функции открытия файла открытого текста

Происходит открытие файла, содержащего открытый текст сообщения. Если файл не был открыт программа выводит на экран сообщение об ошибке и завершает работу. Если файл успешно открыт, определяется длина открытого текста в символах/байтах и заносится в переменную `text_length`. В случае если длина открытого текста составляет не менее четырех символов происходит закрытие файла открытого текста. В противном случае программа выводит на экран сообщение об ошибке и завершает работу. Необходимость наличия не менее четырех символов в файле открытого текста объясняется особенностью дополнения открытого текста до кратности 64 битам, которая будет приведена далее.

Функция чтения и подготовки открытого текста к шифрованию представлена на рисунке 10.

Алгоритм шифрования ГОСТ 28147-89[9] является блочным шифром и подразумевает разбиение открытого текста на блоки равной длины (по 64 бита каждый). Однако длины открытого текста может не хватить для того, чтобы последний блок был необходимой длины. Выполнение функции начинается с подсчета числа байт/символов, которых не хватает в открытом тексте для его кратности 64 битам. Данное число заносится в переменную `dop_length`. Затем по длине открытого текста вычисляется число блоков с учетом дополняющих символов.

Объявляются массивы `text_array`, `dop_array` и `crypt_array`. Первый предназначен для хранения кодов символов открытого текста с учетом дополняющих символов, второй – для хранения только кодов дополняющих символов, третий – для хранения блоков зашифрованного текста.

Происходит открытие файла, содержащего открытый текст сообщения и посимвольное чтение из него в ячейки массива `text_array`. Таким образом, массив `text_array` будет содержать в своих ячейках коды символов открытого текста без учета дополняющих символов. Файл, содержащий открытый текст закрывается.

Рисунок 10 – Функция чтения и подготовки открытого текста к шифрованию

Далее необходимо определить, какими символами будет дополняться открытый текст до кратности 64 битам. Данная операция происходит в цикле, который повторяется столько раз, сколько дополняющих символов не хватает (переменная `dur_length`). Если номер выполнения цикла (значение счетчика) четный, то код дополняющего символа определяется как код символа открытого текста с порядковым номером равным номеру выполнения цикла минус 1, в противном случае – код символа открытого текста с порядковым номером равным номеру выполнения цикла плюс 1. Очередной символ дополнения заносится в соответствующую ячейку массива `dur_array`.

После определения последовательности дополняющих символов происходит соединение (конкатенация) кодов символов открытого текста и символов дополнения. Полученная последовательность кодов символов подлежит зашифрованию и содержится в массиве `text_array`. Криптосхема алгоритма зашифрования в режиме простой замены по ГОСТ 28147-89[9] представлена на рисунке 11.

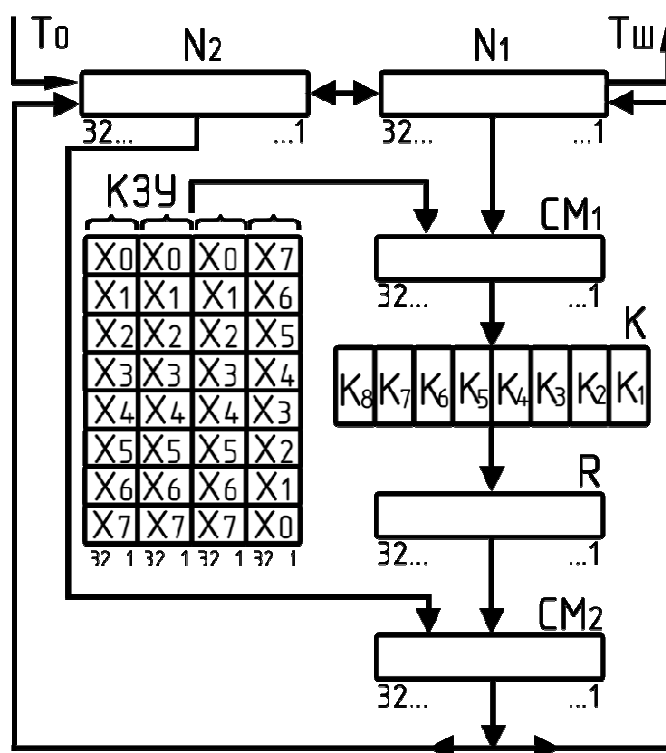


Рисунок 11– Криптосхема алгоритма зашифрования в режиме простой замены по ГОСТ 28147-89[9]

Алгоритм шифрования открытого текста представлен на рисунке 12.

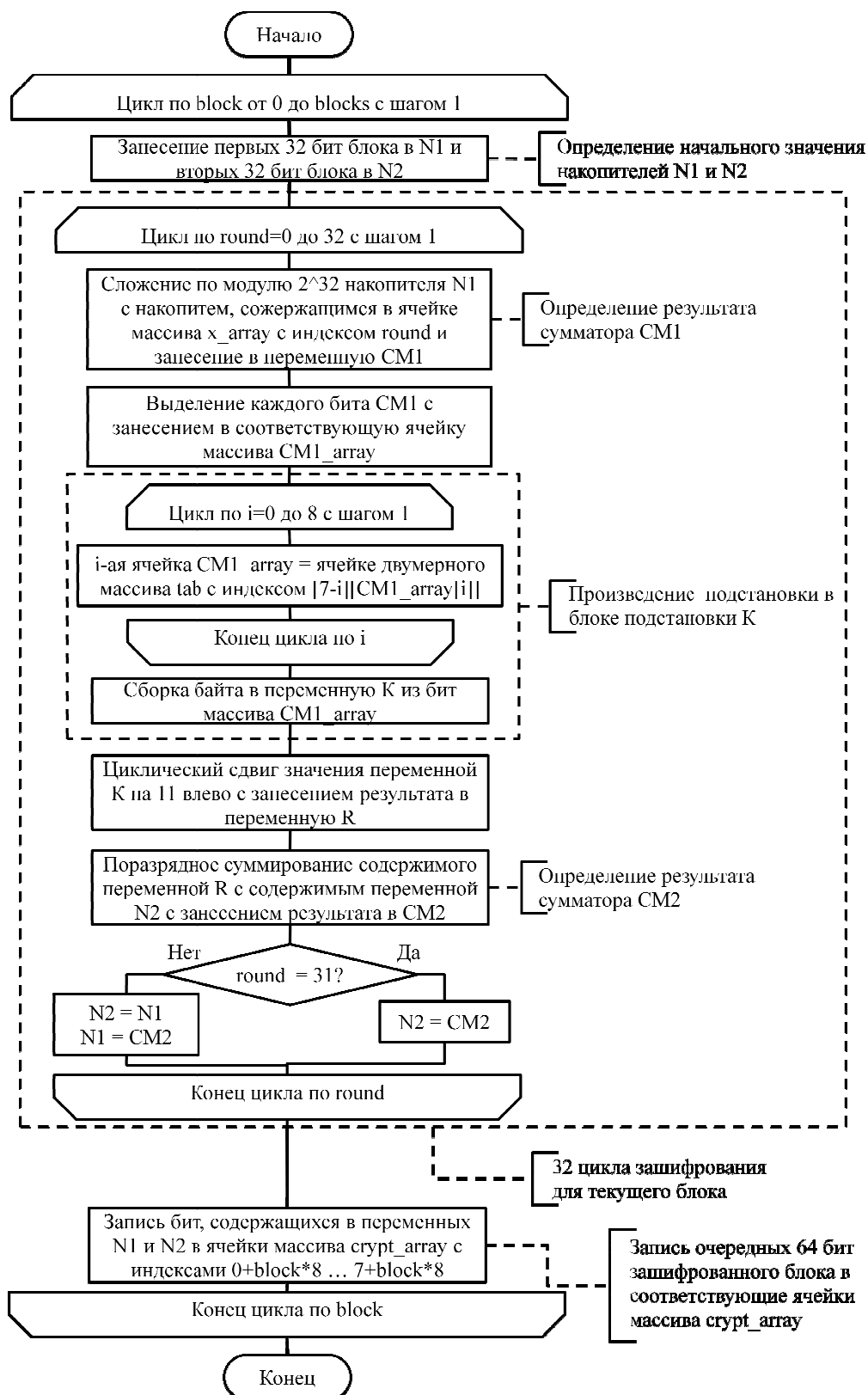


Рисунок 12 – Алгоритм шифрования открытого текста

Шифрование производится для каждого отдельного блока бит из массива текста `text_array`. Согласно алгоритму шифрования ГОСТ 28147-89[9] в режиме

простой замены шифрование каждого блока осуществляется в 32 цикла. Перед выполнением первого цикла формируется начальное заполнение накопителей N_1 и N_2 таким образом, что первая половина бит (32 бита) блока записываются в N_1 а вторая половина в N_2 . Далее идет выполнение циклов зашифрования.

Определяется результат сумматора CM_1 – значение накопителя N_1 складывается по модулю 2^{32} со значением накопителя, содержащимся в ячейке массива x_array с соответствующим номеру цикла индексом. Затем содержимое сумматора CM_1 разбивается на отдельные четверки бит, которые заносятся в соответствующие ячейки массива $CM1_array$.

Затем в узле подстановки происходит подстановка каждой четверки бит из ячеек массива $CM1_array$ в соответствии с таблицей замены. После подстановки четверки бит объединяются в 32-битное значение и оно заносится в переменную K . Содержимое переменной K претерпевает циклический сдвиг на 11 в сторону старших разрядов и заносится в переменную R .

Содержимое переменной R поразрядно складывается с содержимым N_2 по модулю 2 и, являясь результатом сумматора CM_2 , заносится в соответствующую переменную.

После произведенных операций производится проверка на то, является ли текущий цикл зашифрования для блока последним (32 – ым). В случае если это так, значение накопителя N_2 приравнивается к значению результата сумматора CM_2 . В противном случае значение накопителя N_2 приравнивается к значению N_1 , а тот, в свою очередь, приравнивается к значению результата сумматора CM_2 . На этом один из циклов зашифрования заканчивается. Данный цикл повторяется для каждого блока 32 раза. Важно отметить, что при зашифровании в циклах с 1 – го по 8 – ой, с 9 – го по 16 – ый и с 17 – го по 24 – ый накопители из массива x_array (КЗУ) подаются в прямом порядке – $X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7$. В циклах с 25 – го по 32 – ой порядок обратный – $X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0$.

После того, как 32 цикла зашифрования закончились накопители N_1 и N_2 содержат половины бит зашифрованного блока. Данные 64 бита записываются в соответствующие ячейки массива `crypt_array`.

Описанные процедуры повторяются для каждого блока и в конце выполнения функции шифрования массив `crypt_array` будет содержать блоки зашифрованного текста. После того, как текст зашифрован, он подлежит скрытию. Алгоритм стеганографического скрытия информации представлен на рисунке 13.



Рисунок 13 – Алгоритм стеганографического скрытия информации

Скрытие начинается с подготовки зашифрованного текста к скрытию, алгоритм которой приведен на рисунке 14.

Рисунок 14 – Алгоритм функции подготовки зашифрованного текста к скрытию

Для того, чтобы в дальнейшем извлечь скрытый зашифрованный текст необходимо знать его длину. Кроме того, после расшифрования извлеченного зашифрованного текста открытый текст может содержать дополняющие

символы, внедренные ранее. В итоге, чтобы получить идентичный открытый текст необходимо также знать число дополняющих символов. Однако, приемная сторона, при извлечении и расшифровании будет иметь только ключ и файл-контейнер со скрытой информацией. Поэтому, в файл – контейнер предполагается скрывать помимо самого зашифрованного сообщения еще и информацию о его длине и числе дополняющих символов. Для этих целей объявляются массивы `pole1`, `pole2` и `pole3`, предназначенные для хранения бит длины зашифрованного сообщения, бит самого зашифрованного сообщения и бит числа дополняющих символов соответственно.

Чтобы заполнить массив `pole1`, переменная `full_length` разбивается на отдельные биты и заносятся в его ячейки. Затем заполняется массив `pole2` битами блоков из массива `crypt_array`. Массив `pole3` заполняется битами переменной `dop_length`. После формирования данных массивов, они объединяются в массиве `array_to_hide` в последовательности `pole1`, `pole2`, `pole3`. Таким образом, массив `array_to_hide` содержит в себе информацию, необходимую для успешного извлечения и расшифрования сообщения в дальнейшем. Под хранение значения длины зашифрованного текста для части `pole1` отводится 4 байта, для хранения бит зашифрованного текста для части `pole2` отводится `full_length` байт, для хранения значения числа дополняющих символов для части `pole3` отводится 1 байт, так как число дополняющих символов может быть в диапазоне от 0 до 7 включительно. Структура массива `array_to_hide` приведена на рисунке 15.

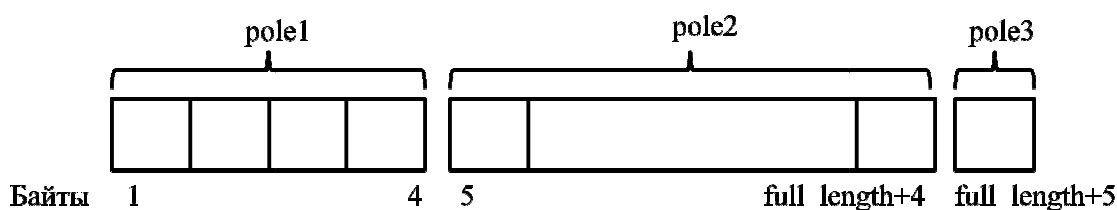


Рисунок 15 – Структура массива `array_to_hide`

Биты, содержащиеся в массиве `array_to_hide` подлежат скрытию.

На рисунке 16 представлен алгоритм функции открытия файла стеганографического контейнера.

Рисунок 16 – Алгоритм функции открытия файла стеганографического контейнера

Происходит открытие файла-контейнера. Если этого не произошло, программа выводит на экран сообщение об ошибке и завершает свою работу. Если файл успешно открыт происходит чтение первых 1000 байт в ячейки массива `wavheader_bigger`, после чего в цикле в данных ячейках происходит поиск четырех ячеек, последовательно, друг за другом содержащие коды символов `d`, `a`, `t`, `a`. Когда поиск завершен переменная `data_position` будет содержать номер байта, с которого начинается блок данных (позиция символа `d`).

Затем в цикле в ячейки массива `wavheader_real`, заносятся значения ячеек массива `wavheader_bigger` до `data_posotion` плюс 7 (первые байты файла – контейнера до байт звуковых отсчетов). Определяется число байт, отводимых для хранения звуковых отсчетов и длина заголовка файла-контейнера.

После происходит чтение байт заголовка с занесением в соответствующие переменные. `Wav` – файл может содержать блоки после блоков данных (звуковых отсчетов), поэтому далее происходит определение числа байт, идущих после блоков данных (звуковых отсчетов) и номера байта, с которого они начинаются в переменные `tail_length` и `tail_position` соответственно.

Файл-контейнер проверяется на соответствие формату. В случае если файл не содержит `RIFF`-контейнер или не является `WAVE`-файлом или не соответствует формату `PCM` (импульсно-кодовой модуляции) программа выводит сообщение об ошибке и завершает свою работу.

После выполнения данной функции идет функция подсчета необходимой информации по стеганографическому контейнеру, алгоритм которой представлен на рисунке 17.

Происходит определение числа отсчетов звуковых сигналов и общего числа байт, отведенных для их хранения. Вычисляется вместимость контейнера в байтах/символах. В случае, если общая длина зашифрованного текста из массива `array_to_hide` (включая `pole1`, `pole2` и `pole3`) больше чем вместимость

контейнера программа выводит на экран сообщение об ошибке и завершает свою работу.

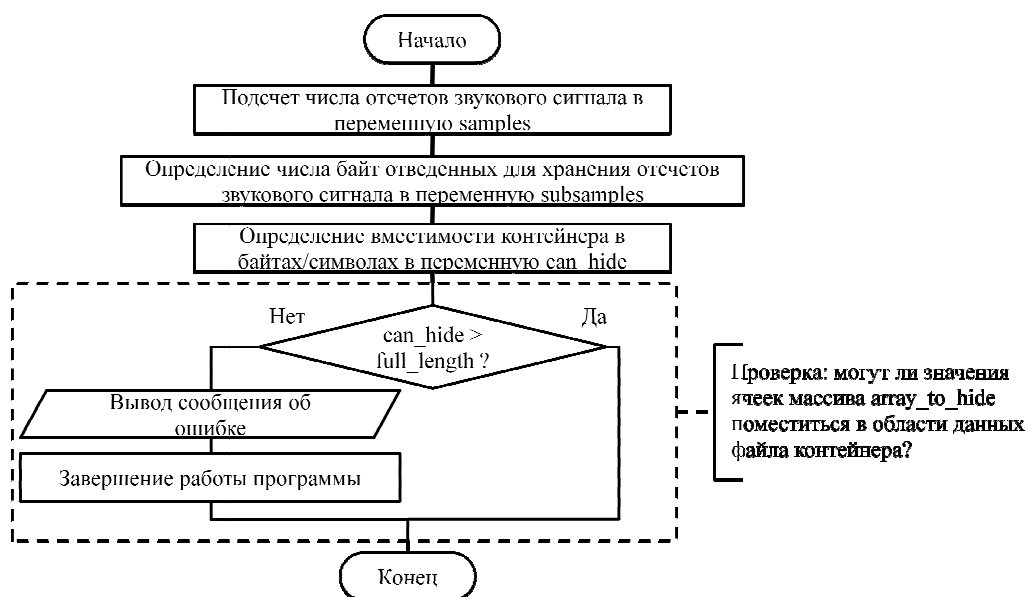


Рисунок 17 – Алгоритм функции подсчета необходимой информации по стеганографическому контейнеру

Алгоритм функции чтения байт данных стеганографического контейнера приведен на рисунке 18.

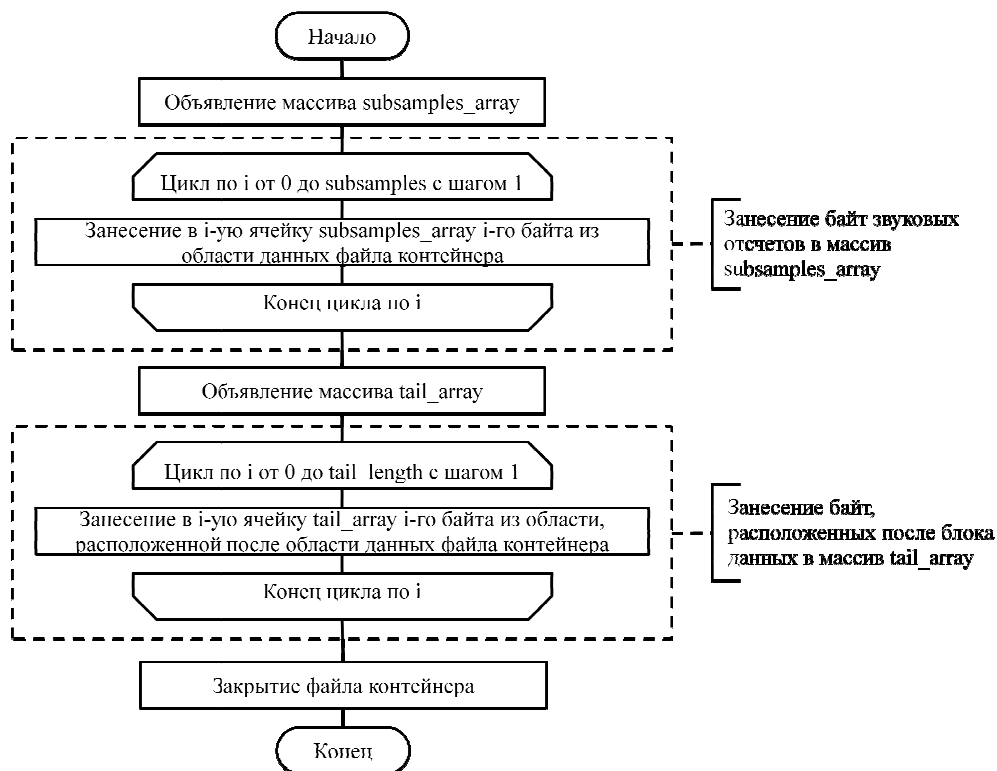


Рисунок 18 – Алгоритм функции чтения байт данных стеганографического контейнера

Из файла – контейнера в цикле производится чтение байт звуковых отсчетов в массив `subsamples_array`, а в массив `tail_array` производится чтение байт, расположенных после блока данных. Файл-контейнер закрывается.

Далее следует функция определения последовательности байт, подлежащих замене, алгоритм которой приведен на рисунке 19.

Происходит инициализация массивов `subsamples_nums`, `subsamples_nums2` и `subsamples_status`. Первые два предназначены для хранения последовательности номеров звуковых отсчетов 8 – битного и не 8 – битного файла – контейнера соответственно, в которые будут внедрены биты зашифрованного текста. Значения ячеек массивов `subsamples_nums2` и `subsamples_status` обнуляются.

Если число байт, отводимых под хранение звуковых отсчетов превышает 1000000, программа выводит на экран информационное сообщение «Подождите...».

В начальную ячейку массива `subsamples_nums` вносится значение определяемое как `subsamples` деленое на 2, а начальная ячейка массива `subsamples_status` увеличивается на единицу. Далее происходит цикл для определения значения каждой из последующих ячеек массива `subsamples_nums`.

В такте цикла происходит вычисление значения текущей по счету ячейки массива `subsamples_nums` как значение предыдущей ячейки плюс псевдослучайное число. Если значение текущей ячейки оказалось выше значения числа байт звуковых отсчетов (переменная `subsamples`) данное значение претерпевает деление по модулю `subsamples`. Далее проверяется, было ли такое значение сгенерировано ранее путем сравнения значения ячейки массива `subsamples_status` (имеющий индекс, равный сгенерированному значению) с нулем (для того чтобы избежать повторения значений последовательностей байт, подлежащих замене).

Рисунок 19 – Алгоритм функции определения последовательности байт,
подлежащих замене

Если в рассматриваемой ячейке массива `subsamples_status` находится значение, отличное от нуля, это значит, что сгенерированное число уже было сгенерировано ранее и необходимо сгенерировать новое число отличное от каждого, сгенерированного ранее. Поэтому снова происходит вычисление значения текущей по счету ячейки массива `subsamples_nums` как значение уже текущего сгенерированного значения плюс псевдослучайное число. Если значение текущей ячейки оказалось выше значения числа байт звуковых отсчетов (переменная `subsamples`) данное значение претерпевает деление по модулю `subsamples`.

Соответствующая ячейка массива `subsamples_status` уменьшается на единицу, а функция возвращается к месту первого сравнения значения ячейки массива `subsamples_status` с нулем.

Если значение рассматриваемой ячейки массива `subsamples_status` равно нулю, то это значит, что такого числа еще не было сгенерировано и значение данной ячейки увеличивается на единицу, чтобы в следующем такте цикла не было сгенерировано число, сгенерированное в данном такте цикла. Такт цикла завершается.

Таким образом, формируется содержимое массива `subsamples_nums`, содержащего в своих ячейках номера байт, определяющие последовательность байт в области данных 8-битного файла-контейнера, в которой будет происходить внедрение бит зашифрованного текста.

Также производится и формирование значения ячеек массива `subsamples_nums2`, который по своему назначению схож с `subsamples_nums` и отличается содержимым, предназначенным для скрытия в файлах, битность которых отличается от 8. Данный массив принимает из массива `subsamples_nums` только те значения, которые кратны числу байт, отводимых в файле – контейнере под один звуковой отсчет.

Алгоритм функции скрытия зашифрованного текста и создания выходного файла представлен на рисунке 20.

Рисунок 20 – Алгоритм функции скрывтия зашифрованного текста и создания
выходного файла

В цикле происходит внедрение младших бит зашифрованного текста в байты звуковых отсчетов файла-контейнера, хранящихся в массиве `subsamples_array`. В зависимости от битности, внедрение бит происходит в последовательности определенной в массивах `subsamples_nums` и `subsamples_nums2` для 8-битных и остальных файлов соответственно.

Затем создается выходной файл. В него из массива `wavheader_real` записываются байты заголовка оригинального файла – контейнера. После байт массива в файл записываются байты звуковых отсчетов (данных) из массива `subsamples_array`, содержащие биты зашифрованного текста. После байт данных записываются байты, содержащиеся в оригинальном файле-контейнере, расположенные после байт данных. Выходной файл закрывается.

Таким образом программа создает выходной wav-файл, идентичный своему оригинальному файлу, за исключением того, что в некоторых байтах данных младшие биты принадлежат зашифрованному тексту. Сформировался стеганографический контейнер, пригодный к дальнейшему извлечению и расшифрованию.

3.3 Разработка программы, реализующей шифрование и скрытие сообщений

3.3.1 Описание начальных переменных

В самом начале, вне тела функции `main()` происходит объявление структуры `audio`:

```
struct audio//структура для WAV-контейнера
{
    unsigned int chunkID;
    unsigned int chunkSize;
    unsigned int format;
    char subChunk1ID[4];
    unsigned int subChunk1Size;
    unsigned int audioFormat;
    unsigned int numChannels;
    unsigned int sampleRate;
    unsigned int byteRate;
    unsigned int blockAlign;
    unsigned int bitsPerSamle;
```

```

        char subChunk2ID[4];
        unsigned int subChunk2Size;
    } AUDIO;

```

В данной структуре содержатся переменные для хранения байт заголовка wav-файла, а именно:

- chunkID – метка, являющаяся началом области RIFF;
- chunkSize – оставшийся размер RIFF области (размер файла минус 8 байт);
- format – метка формата WAVE;
- subChunk1ID – метка области fmt;
- subchunk1Size – оставшийся размер области fmt;
- audioFormat – аудио формат;
- nuChannels – количество каналов;
- sampleRate – частота дискретизации;
- byteRate – количество байт, используемых за секунду воспроизведения;
- blockAlign – количество байт, используемых для кодирования одного звукового отсчета с учетом всех каналов;
- bitsPerSample – количество бит используемых для кодирования одного звукового отсчета;
- subChunk2ID – метка области звуковых данных;
- subchunk2Size – размер области звуковых данных.

В теле функции main() объявляются начальные переменные для зашифрования:

```

int i, j, round, block, k; //счетчики
long key_length=0; //длина ключа в символах/байтах
int text_length=0; //длина о.т в символах/байтах
int dop_length=0; //число символов, дополняющих о.т до кратности 64 битам
int blocks=0; //число блоков о.т по 64 бита
unsigned int x0, x1, x2, x3, x4, x5, x6, x7; //КЗУ
unsigned int x_array[32]; //массив в котором КЗУ выстроены в порядке нужном для зашифрования

```

```

unsigned int N1=0,N2=0;//накопители
unsigned int CM1=0,CM2=0;
unsigned int CM1_array[8]={0,0,0,0,0,0,0,0};//массив для разбиения
    CM1 на части по 4 бита
unsigned int K=0;//значение CM1 после подстановки
unsigned int R=0,r=0;
int tab[8][16] = {
{0x4,0xA,0x9,0x2,0xD,0x8,0x0,0xE,0x6,0xB,0x1,0xC,0x7,0xF,0x5,0x3},
    //k1
{ 0xE,0xB,0x4,0xC,0x6,0xD,0xF,0xA,0x2,0x3,0x8,0x1,0x0,0x7,0x5,0x9
    },//k2
{ 0x5,0x8,0x1,0xD,0xA,0x3,0x4,0x2,0xE,0xF,0xC,0x7,0x6,0x0,0x9,0xB
    },//k3
{ 0x7,0xD,0xA,0x1,0x0,0x8,0x9,0xF,0xE,0x4,0x6,0xC,0xB,0x2,0x5,0x3
    },//k4
{ 0x6,0xC,0x7,0x1,0x5,0xF,0xD,0x8,0x4,0xA,0x9,0xE,0x0,0x3,0xB,0x2
    },//k5
{ 0x4,0xB,0xA,0x0,0x7,0x2,0x1,0xD,0x3,0x6,0x8,0x5,0x9,0xC,0xF,0xE
    },//k6
{ 0xD,0xB,0x4,0x1,0x3,0xF,0x5,0x9,0x0,0xA,0xE,0x7,0x6,0x8,0x2,0xC
    },//k7
{ 0x1,0xF,0xD,0x0,0x5,0x7,0xA,0x4,0x9,0x2,0x3,0xE,0x6,0xB,0x8,0xC
    } //k8
};//таблица замен Идентификатор: id-GostR3411-94-TestParamSet

```

Данные переменные предназначены для:

- i, j, k – счетчики;
- round, block – счетчики для функции зашифрования;
- key_length – длина ключа в байтах;
- text_length – длина текста открытого сообщения в байтах;
- dop_length – число символов, дополняющих текст открытого сообщения до кратности 64 битам;
- blocks – число блоков открытого текста с учетом дополняющих символов (по 64 бита каждый);
- x0, x1, x2, x3, x4, x5, x6, x7 – накопители КЗУ;
- x_array – массив в котором содержимое накопителей КЗУ выстроено в порядке нужном для зашифрования;
- N1, N2 – накопители;
- CM1, CM2 – результаты сумматоров;

- CM1_array – массив, в котором значение переменной CM1, разделенное на части по 4 бита;
- K – результат подстановки;
- R – результат циклического сдвига;
- r – старшая часть для циклического сдвига;
- tab – таблица замены.

Кроме того, в теле функции main() происходит объявление начальных переменных для скрытия:

```
unsigned int tail_length=0; //длина хвостовой части
unsigned int tail_position=0; //позиция начала хвостовой части
unsigned int samples=0; //число сэмплов в файле
unsigned int subsamples=0; //число сэмплов * число каналов
unsigned int can_hide=0; //число символов/байт для скрытия в данном
    контейнере
unsigned int wavheader_bigger[1000]; //массив для хранения
    заголовка Wav
unsigned int data_position=0; //позиция слова data
unsigned int wavheader_length=0; //длина заголовка в байтах
int trash=0; //ненужные байты
```

Данные переменные предназначены для:

- tail_length – количество байт, расположенных в файле после блока данных;
- tail_position – позиция, с которой начинаются байты, расположенные в файле после блока данных;
- samples – количество звуковых отсчетов (включая все каналы);
- subsamples – количество отдельных звуковых отсчетов;
- wavheader_bigger – массив для хранения первых 1000 байт файла – контейнера;
- data_position – позиция начала последовательности байт d, a, t, a в заголовке файла – контейнера;
- wavheader_length – число байт, расположенных в файле – контейнере перед байтами звуковых отсчетов;
- trash – переменная предназначена для чтения ненужных байт.

Остальные переменные будут объявляться на основании значений уже инициализированных переменных.

3.3.2 Функция открытия и чтения файла ключа

В ходе разработки программы, она была условно поделена на функции, каждая из которых представлена своей последовательностью команд, выполняющих свое функциональное предназначение. Данные функции представлены в неявном виде.

Функция открытия и чтения файла ключа представлена последовательностью следующих команд:

```
//открытие файла ключа
FILE * ptrFile = fopen("key.txt", "rb");
if (ptrFile == NULL)
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    {
        system ("pause");
        return -1;
    }
}
else
{
    fseek(ptrFile, 0, SEEK_END); // переместить внутренний
    указатель в конец файла
    key_length = ftell(ptrFile); // вернуть текущее положение
    внутреннего указателя
    fclose (ptrFile); // закрыть файл
    if (key_length!=32)
    {
        cout<<"ОШИБКА! Файл ключа key.txt не равен 256 битам."<<endl;
        system ("pause");
        return -1;
    }
}
unsigned int *key_array = new unsigned int
    [key_length]; //динамический массив под хранение кодов символов
    ключа
ifstream keystream("key.txt", ios::binary);
if (!(keystream.is_open()))
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    system ("pause");
}
```

```

        return -1;
    }
    //чтение ключа
    for (i=0;i<key_length;i++)
    {
        key_array[i]=0;
        keystoream.read((char*)&key_array[i],1);
    }
    //закрытие
    keystoream.close();

```

Сначала происходит открытие файла ключа key.txt в режиме бинарного чтения для проверки на существование и для определения его длины. Если файл не существует, или открытия не произошло по каким – либо другим причинам, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл ключа key.txt не был открыт.» и завершает свою работу.

Если файл был открыт, внутренний указатель в открытом файле перемещается в его конец. Значение позиции указателя заносится в переменную key_length, которое означает число байт, содержащихся в файле ключа. Файл закрывается. Происходит проверка переменной key_length и если ее значение отличается от 32 (если ключ содержит не 256 бит) программа выводит на экран сообщение об ошибке «ОШИБКА! Файл ключа key.txt не равен 256 битам.» и завершает свою работу. Если значение key_length равно 32, функция продолжает свою работу. Объявляется динамический массив key_array длиной key_length ячеек. Происходит открытие файла ключа для чтения. Чтение ключа реализовано в цикле по i от 0 до key_length с шагом 1. Значение i – ой ячейки массива key_array обнуляется и в него записывается байт, считанный из файла ключа. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i. Файл закрывается.

В результате выполнения функции формируется массив key_array, в ячейках которого содержатся байты ключа.

3.3.3 Функция формирования КЗУ

Сначала выполняются команды:

```
//Заполнение КЗУ
x0=(key_array[0]<<24)+(key_array[1]<<16)+(key_array[2]<<8)+key_array[3];
x1=(key_array[4]<<24)+(key_array[5]<<16)+(key_array[6]<<8)+key_array[7];
x2=(key_array[8]<<24)+(key_array[9]<<16)+(key_array[10]<<8)+key_array[11];
x3=(key_array[12]<<24)+(key_array[13]<<16)+(key_array[14]<<8)+key_array[15];
x4=(key_array[16]<<24)+(key_array[17]<<16)+(key_array[18]<<8)+key_array[19];
x5=(key_array[20]<<24)+(key_array[21]<<16)+(key_array[22]<<8)+key_array[23];
x6=(key_array[24]<<24)+(key_array[25]<<16)+(key_array[26]<<8)+key_array[27];
x7=(key_array[28]<<24)+(key_array[29]<<16)+(key_array[30]<<8)+key_array[31];
```

Ключевое запоминающее устройство (КЗУ) представляет собой восемь 32-разрядных накопителей. Значение первого накопителя x_0 образуется из первых четырех байт ключа, расположенных в ячейках массива `key_array` следующим образом значение нулевой ячейки массива `key_array` побитово сдвигается на 24 позиции влево, значение первой ячейки массива сдвигается на 16 позиций влево, значение второй ячейки сдвигается на 8 позиций влево. Получившиеся значения логически суммируются между собой и со значением третьей ячейки массива `key_array` и записываются в накопитель x_0 . Следующие накопители суммируются аналогичным образом, за исключением того, что используются соответствующие накопителю четыре ячейки массива `key_array`. Так, восемь накопителей $x_0, \dots, x_7$ содержат в себе по четыре байта (32 бита) ключа, показано на рисунке 21.

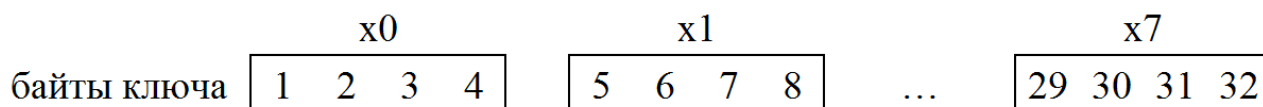


Рисунок 21 – Накопители КЗУ

Затем значения накопителей КЗУ выстаиваются в необходимой последовательности подачи для зашифрования:

```
//Заполнение Массива x_array последовательности подачи КЗУ
//ячейки 0-23
x_array[0]=x_array[8]=x_array[16]=x0;
x_array[1]=x_array[9]=x_array[17]=x1;
x_array[2]=x_array[10]=x_array[18]=x2;
x_array[3]=x_array[11]=x_array[19]=x3;
x_array[4]=x_array[12]=x_array[20]=x4;
x_array[5]=x_array[13]=x_array[21]=x5;
x_array[6]=x_array[14]=x_array[22]=x6;
x_array[7]=x_array[15]=x_array[23]=x7;
//ячейки 24-31
x_array[24]=x7;
x_array[25]=x6;
x_array[26]=x5;
x_array[27]=x4;
x_array[28]=x3;
x_array[29]=x2;
x_array[30]=x1;
x_array[31]=x0;
```

Массив `x_array` содержит 32 ячейки, по числу предстоящих циклов зашифрования. Алгоритм ГОСТ 28147-89[9] устанавливает, что в каждом из 32 циклов зашифрования будет использован определенный накопитель КЗУ. Необходимая последовательность значений накопителей КЗУ, используемых при зашифровании формируется в массиве `x_array`. Ячейки данного массива с индексами 0, 8, 16 приравниваются к значению накопителя `x0`. Ячейки с индексами 1, 9, 17 приравниваются к значению накопителя `x1`, ячейки с индексами 2, 10, 18 приравниваются к значению накопителя `x2` и т.д. Последние восемь ячеек заполняются в обратном порядке: 24 – ая принимает значение накопителя `x7`, 25 – ая принимает значение накопителя `x6`, ..., последняя принимает значение накопителя `x0`. Окончательное содержимое массива `x_array` показано на рисунке 22.

Номер ячейки	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значение накопителя	x0	x1	x2	x3	x4	x5	x6	x7	x0	x1	x2	x3	x4	x5	x6	x7

Номер ячейки	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Значение накопителя	x0	x1	x2	x3	x4	x5	x6	x7	x7	x6	x5	x4	x3	x2	x1	x0

Рисунок 22 – Содержимое массива x_array

3.3.4 Функция открытия файла открытого текста

Данная функция представлена последовательностью команд:

```
//2.1.открытие файла для определения длины
// открытие файла о.т.
FILE * ptrFile2 = fopen("text.txt","rb");
if (ptrFile2 == NULL)
{
    cout<<"ОШИБКА! Файл открытого текста text.txt не был открыт."
    <<endl;
    system ("pause");
    return -1;
}
else
{
    fseek(ptrFile2, 0, SEEK_END); // переместить внутренний
    указатель в конец файла
    text_length = ftell(ptrFile2); // вернуть текущее положение
    внутреннего указателя
    fclose (ptrFile2); // закрыть файл
}
//проверка длины о.т
if (text_length <4)//если в о.т меньше 4 символов
{
    cout<<"ОШИБКА! Файл открытого текста text.txt должен соержать
    не менее 4 символов."<<endl;
    system ("pause");
    return -1;
}
```

Происходит открытие файла text.txt содержащего текст открытого сообщения. Если файл не существует, или открытия не произошло по каким – либо другим причинам, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл открытого текста text.txt не был открыт.» и завершает свою работу.

Если файл был открыт, внутренний указатель в открытом файле перемещается в его конец. Значение позиции указателя заносится в переменную `text_length`, которое означает число байт, содержащихся в файле открытого текста. Файл закрывается. Происходит проверка переменной `text_length` и если ее значение меньше 4 (текст содержит менее 4 символов) программа выводит на экран сообщение об ошибке «ОШИБКА! Файл открытого текста `text.txt` должен содержать не менее 4 символов» и завершает свою работу.

3.3.5 Функция чтения и подготовки к шифрованию открытого текста

В начале происходит подсчет числа недостающих в открытом тексте символов до его кратности 64 битам (8 байтам):

```
//2.3 подсчет числа недостающих в о.т символов до кратности 64
    битам
if ((text_length%8)!=0)
    dop_length=8-text_length%8;
```

Если значение переменной `text_length` не кратно восьми, внести в переменную `dop_length` значение определяемое по формуле:

$$dop_length = 8 - text_length \bmod 8$$

Если значение переменной `text_length` кратно восьми, то переменная `dop_length` не претерпевает изменений и ее значение остается равным нулю (было задано при инициализации). Так переменная `dop_length` может иметь значение от 0 до 7 включительно.

Далее определяется количество блоков открытого текста с учетом дополняющих символов:

```
//2.4 определение числа блоков (по 64 бита)
blocks=(text_length+dop_length)/8;
```

Переменная `blocks` содержит число блоков по 64 бита каждый и вычисляется по формуле:

$$blocks = \frac{text\ length + dop\ length}{8}$$

Затем идет последовательность команд:

```
unsigned int *text_array = new unsigned int
    [text_length+dop_length]; //динамический массив под хранение
    кодов символов о.т + дополнения
unsigned int *dop_array = new unsigned int [dop_length];
    //динамический массив под хранение кодов символов дополнения
unsigned int *crypt_array = new unsigned int
    [text_length+dop_length]; //динамический массив под хранение
    кодов символов ш.т
//обнуление crypt_array
for (i=0;i<text_length+dop_length;i++)
{
    crypt_array[i]=0;
}
ifstream textstream("text.txt",ios::binary);
if (!(textstream.is_open()))
{
    cout<<"ОШИБКА! Файл открытого текста text.txt не был открыт."
    <<endl;
    system ("pause");
    return -1;
}
//чтение о.т
for (i=0;i<text_length+dop_length;i++)
{
    text_array[i]=0;
    if (i<text_length)
        textstream.read((char*)&text_array[i],1);
}
//закрытие
textstream.close();
```

Объявляются массивы `text_array`, `dop_array` и `crypt_array`. Массив `text_array` имеет длину `text_length` плюс `dop_length` ячеек и предназначен для хранения байт открытого текста и дополняющих символов. Массив `dop_array`

имеет длину `dop_length` ячеек и предназначен для хранения байт дополняющих символов. Массив `crypt_array` имеет длину `text_length` плюс `dop_length` ячеек и предназначен для хранения байт зашифрованного текста с учетом дополняющих символов.

Происходит открытие файла открытого текста для чтения. Если файл не существует, или открытия не произошло по каким – либо другим причинам, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл открытого текста `text.txt` не был открыт.» и завершает свою работу.

Если файл был открыт происходит чтение открытого текста. В цикле по `i` от 0 до `text_length` плюс `dop_length` с шагом 1 `i` – ая ячейка массива `text_array` обнуляется и если значение счетчика `i` меньше значения `text_length` в нее заносится значение считанного байта из файла открытого текста. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Файл закрывается.

После того, как байты открытого текста прочитаны, необходимо определить дополняющие символы:

```
//2.6 определение дополняющих символов
for (i=0;i<dop_length;i++)
{
    if (i%2==0)//если символ о.т четный
dop_array[i]=text_array[i]-1;
    else //иначе
        dop_array[i]=text_array[i]+1;
}
//2.7 внедрение дополняющих символов
j=0;
for (i=text_length;i<text_length+dop_length;i++)//в незаполненные
    ячейки массива text_array внести ячейки dop_array
{
    text_array[i]=dop_array[j];
    j++;
}
```

В цикле по `i` от 0 до `dop_length` с шагом 1 происходит определение очередного дополняющего символа. Если значение счетчика `i` делится на 2 без остатка `i` – ая ячейка массива `dop_array` принимает значение `i` – ой ячейки

массива `text_array` минус 1, в противном случае она принимает значение `i` – ой ячейки массива `text_array` плюс 1. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Массив `dop_array` теперь содержит байты дополняющих символов.

Далее переменная `j` обнуляется и в цикле по `i` от `text_length` до `text_length` плюс `dop_length` с шагом 1 `i` – ая ячейка массива `text_array` принимает значение `j` – ой ячейки массива `dop_array`, а переменная `j` увеличивается на единицу. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Таким образом, массив `text_array` теперь содержит символы открытого текста с учетом дополняющих символов. Теперь длина текста кратна 64 битам.

3.3.6 Функция шифрования

Данная функция представлена последовательностью следующих команд:

```
//3.ШИФРОВАНИЕ
//Для каждого блока..
for (block=0;block<blocks;block++)//начало поблочного цикла
{
//определение начального заполнения N1 и N2 (перед первым циклом
зашифрования)
N1=(text_array[0+8*block]<<24)+(text_array[1+8*block]<<16)+(text_a
rray[2+8*block]<<8)+text_array[3+8*block];
N2=(text_array[4+8*block]<<24)+(text_array[5+8*block]<<16)+(text_a
rray[6+8*block]<<8)+text_array[7+8*block];
//циклы 2...32 32-3
for (round=0;round<32/*32*/;round++)//со 2-го цикла 32-3
{
//подсчет CM1 - сложение по модулю 2^32
CM1=N1+x_array[round];//(V)
//заполнение CM1_array для подстановки(V)
CM1_array[0]=(CM1&0xf0000000)>>28;
CM1_array[1]=(CM1&0xf000000)>>24;
CM1_array[2]=(CM1&0xf00000)>>20;
CM1_array[3]=(CM1&0xf0000)>>16;
CM1_array[4]=(CM1&0xf000)>>12;
CM1_array[5]=(CM1&0xf00)>>8;
CM1_array[6]=(CM1&0xf0)>>4;
CM1_array[7]=CM1&0xf;
//произведение подстановки(V)
```

```

for (i=0;i<8;i++)//для каждой четверки бит CM1
{
    CM1_array[i]=tab[7-i][CM1_array[i]];//подстановка
}
//сборка массива CM1_array после подстановки в переменную K
K=(CM1_array[0]<<28)+(CM1_array[1]<<24)+(CM1_array[2]<<20)+(CM1_ar
ray[3]<<16)+(CM1_array[4]<<12)+(CM1_array[5]<<8)+(CM1_array[6]
<<4)+(CM1_array[7]);
//циклический сдвиг на 11 влево (V)
r=(K&0xffe00000)>>21;//старшая часть помещается в r и
сдвигается в конец младший
R=(K<<11)+r;//сдвиг числа на 11 влево + r
CM2=R^N2; //сложение по модулю 2 (XOR) (V)
// N1 -> N2, CM2 -> N1 (V)
if (round!=31)//для всех циклов кроме последнего 32-го
{
    N2=N1;
    N1=CM2;
}
else //для последнего цикла
{
    N2=CM2;
}
} //конец цикла 32-3
//запись N1,N2 в массив crypt_array
crypt_array[0+block*8]=(N1&0xff000000)>>24;
crypt_array[1+block*8]=(N1&0x00ff0000)>>16;
crypt_array[2+block*8]=(N1&0x0000ff00)>>8;
crypt_array[3+block*8]=(N1&0x000000ff);
crypt_array[4+block*8]=(N2&0xff000000)>>24;
crypt_array[5+block*8]=(N2&0x00ff0000)>>16;
crypt_array[6+block*8]=(N2&0x0000ff00)>>8;
crypt_array[7+block*8]=(N2&0x000000ff);
} //конец поблочного цикла

```

Шифрование происходит для каждого блока в цикле по block от 0 до blocks с шагом 1. Определяется заполнение переменных N1 и N2 соответствующих накопителям N_1 и N_2 :

```

//определение начального заполнения N1 и N2 (перед первым циклом
зашифрования)
N1=(text_array[0+8*block]<<24)+(text_array[1+8*block]<<16)+(text_a
rray[2+8*block]<<8)+text_array[3+8*block];
N2=(text_array[4+8*block]<<24)+(text_array[5+8*block]<<16)+(text_a
rray[6+8*block]<<8)+text_array[7+8*block];

```

N1 определяется как сумма значений ячеек массива `text_array` с индексами (0 плюс 8, умноженное на `block`) после битового сдвига на 24 влево, (1 плюс 8, умноженное на `block`) после битового сдвига на 16 влево, (2 плюс 8, умноженное на `block`) после битового сдвига на 8 влево, (3 плюс 8, умноженное на `block`). N2 определяется как сумма значений ячеек массива `text_array` с индексами (4 плюс 8, умноженное на `block`) после битового сдвига на 24 влево, (5 плюс 8, умноженное на `block`) после битового сдвига на 16 влево, (6 плюс 8, умноженное на `block`) после битового сдвига на 8 влево, (7 плюс 8, умноженное на `block`). Таким образом переменные N1 и N2 содержат первые и последние 32 бита i – го блока текста из массива `text_array` соответственно, показано на рисунке 23.

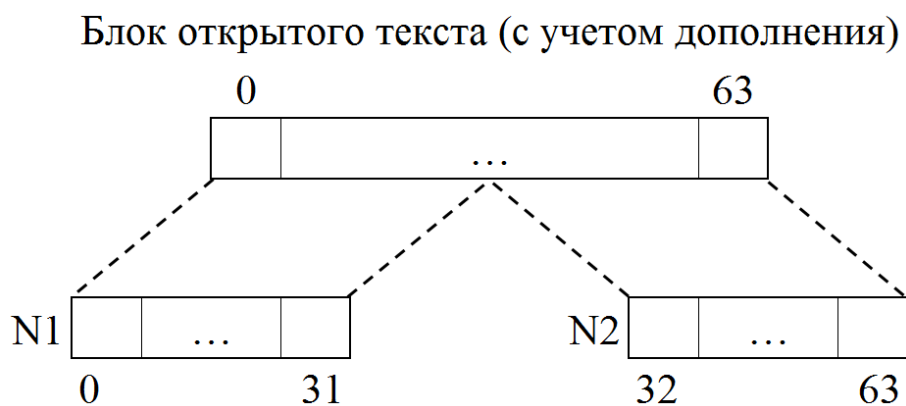


Рисунок 23 – Распределение бит блока по накопителям

После того как значения переменных N1 и N2 сформированы идет шифрование, которое реализовано в виде цикла по `round` от 0 до 32 с шагом 1. В такте цикла происходит подсчет переменной CM1 путем сложения по модулю 2^{32} значения N1 с содержимым ячейки массива `x_array` имеющей индекс `round` (Содержимое 32 ячеек `x_array` было определено ранее в функции формирования КЗУ). Получившееся значение переменной CM1 соответствует результату сумматора CM₁.

Далее значение CM1 должно пройти процедуру подстановки. Для этого по ячейкам массива `CM1_array` распределяются четверки бит переменной CM1:

```
//заполнение CM1_array для подстановки(V)
CM1_array[0]=(CM1&0xf0000000)>>28;
CM1_array[1]=(CM1&0xf0000000)>>24;
```

```

CM1_array[2]=(CM1&0xf00000)>>20;
CM1_array[3]=(CM1&0xf0000)>>16;
CM1_array[4]=(CM1&0xf000)>>12;
CM1_array[5]=(CM1&0xf00)>>8;
CM1_array[6]=(CM1&0xf0)>>4;
CM1_array[7]=CM1&0xf;

```

Нулевая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0xf0000000 побитово сдвинутое на 28 позиций вправо.

Первая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0f000000 побитово сдвинутое на 24 позиций вправо.

Вторая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x00f00000 побитово сдвинутое на 20 позиций вправо.

Третья ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x000f0000 побитово сдвинутое на 16 позиций вправо.

Четвертая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0000f000 побитово сдвинутое на 12 позиций вправо.

Пятая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x00000f00 побитово сдвинутое на 8 позиций вправо.

Шестая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x000000f0 побитово сдвинутое на 4 позиций вправо.

Седьмая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0000000f.

После заполнения ячеек массива CM1_array происходит подстановка:

```

//произведение подстановки(V)
for (i=0;i<8;i++)//для каждой четверки бит CM1

```

```
{
    CM1_array[i]=tab[7-i][CM1_array[i]]; //подстановка
}
```

В цикле по i от 0 до 8 с шагом 1 значение i – ой ячейки CM1_array становится равным начению ячейки двумерного массива tab (таблицы замены) с первым индексом 7 минус i и вторым индексом, определяемым содержимым i – ой ячейки массива CM1_array. Такт цикла по i завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

Затем получившиеся значения в ячейках массива CM1_array собираются воедино в переменную K:

```
//сборка массива CM1_array после подстановки в переменную K
K=(CM1_array[0]<<28)+(CM1_array[1]<<24)+(CM1_array[2]<<20)+(CM1_ar
    ray[3]<<16)+(CM1_array[4]<<12)+(CM1_array[5]<<8)+(CM1_array[6]
        <<4)+(CM1_array[7]);
```

Значение K получается путем суммирования нулевой ячейки CM1_array, сдвинутой побитово на 28 влево, первой ячейки CM1_array, сдвинутой побитово на 24 влево, второй ячейки CM1_array, сдвинутой побитово на 20 влево, третьей ячейки CM1_array, сдвинутой побитово на 16 влево, четвертой ячейки CM1_array, сдвинутой побитово на 12 влево, пятой ячейки CM1_array, сдвинутой побитово на 8 влево, шестой ячейки CM1_array, сдвинутой побитово на 4 влево и седьмой ячейки CM1_array.

Так переменная K содержит значение CM1 после подстановки. Данное значение согласно алгоритму шифрования ГОСТ 28147-89[9] в режиме простой замены должно претерпеть циклический сдвиг на 11 позиций в сторону старших разрядов в регистре R:

```
//циклический сдвиг на 11 влево
r=(K&0xffe00000)>>21; //старшая часть помещается в r и сдвигается в
    конец младший
R=(K<<11)+r; //сдвиг числа на 11 влево + r
```

Для реализации циклического сдвига старшие 11 бит переменной K умножаются на маску 0xffe00000 и побитово сдвигаются на 21 позицию вправо, результат записывается в переменную r. Значение переменной R определяется

как результат битового сдвига значения переменной K на 11 позиций влево сложенный с переменной r. Циклический сдвиг произведен.

Далее определяется значение переменной CM2 соответствующей результату сумматора CM₂:

```
CM2=R^N2; //сложение по модулю 2 (XOR)
```

Значение переменных R и N2 складываются по модулю 2 и получившиеся значения заносятся в переменную CM2.

После того как значение CM2 получено происходит проверка на то, является ли такт цикла по round последним (32 по счету):

```
// N1 -> N2, CM2 -> N1 (V)
if (round!=31)//для всех циклов кроме последнего 32-го
{
    N2=N1;
    N1=CM2;
}
else //для последнего цикла
{
    N2=CM2;
}
} //конец цикла 32-3
```

Если счетчик цикла round не равен 31 (такт цикла не последний) переменная N2 приравнивается к N1, а тот, в свою очередь становится равен CM2. В противном случае такт цикла будет последним и значение CM2 заносится в переменную N2, а значение переменной N1 останется неизменным.

На этом такт цикла по round завершается. Цикл завершается или повторяется в зависимости от значения счетчика round.

После того, как 32 цикла зашифрования произведены на выходе из этих циклов переменные N1 и N2 содержат биты зашифрованного текста блока. Для того чтобы сформировать блок зашифрованного текста была написана следующая последовательность команд:

```
//запись N1,N2 в массив crypt_array
crypt_array[0+block*8]=(N1&0xff000000)>>24;
crypt_array[1+block*8]=(N1&0x00ff0000)>>16;
crypt_array[2+block*8]=(N1&0x0000ff00)>>8;
crypt_array[3+block*8]=(N1&0x000000ff);
crypt_array[4+block*8]=(N2&0xff000000)>>24;
```

```

crypt_array[5+block*8]=(N2&0x00ff0000)>>16;
    crypt_array[6+block*8]=(N2&0x0000ff00)>>8;
    crypt_array[7+block*8]=(N2&0x000000ff);
} //конец поблочного цикла

```

Значение ячейки массива `crypt_array` с индексом (0 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N1` на маску `0xff000000` с последующим битовым сдвигом на 24 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (1 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N1` на маску `0x00ff0000` с последующим битовым сдвигом на 16 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (2 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N1` на маску `0x0000ff00` с последующим битовым сдвигом на 8 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (3 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N1` на маску `0x000000ff`.

Значение ячейки массива `crypt_array` с индексом (4 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0xff000000` с последующим битовым сдвигом на 24 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (5 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x00ff0000` с последующим битовым сдвигом на 16 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (6 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x0000ff00` с последующим битовым сдвигом на 8 позиции вправо.

Значение ячейки массива `crypt_array` с индексом (7 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x000000ff`.

Таким образом в восьми ячейках массива `crypt_array` формируются байты, принадлежащих блоку зашифрованного текста.

Такт цикла по `block` завершается. Цикл повторяется или завершается в зависимости от значения счетчика `block`.

В конце цикла по `block` массив `crypt_array` содержит в каждом своих восьми ячейках байты блока зашифрованного текста.

3.3.7 Функция подготовки зашифрованного текста к скрытию

Данная функция предназначена для формирования последовательности байт, готовых к непосредственному стеганографическому скрытию в файле – контейнере. Сначала в данной функции происходит заполнение массива `pole1` битами значения переменной `full_length`:

```
//ПРЕОБРАЗОВАНИЕ Ш.Т. В ВИД НУЖНЫЙ ДЛЯ СКРЫТИЯ
//заполнение массива pole1
for (i=0; i<32; i++)
{
    pole1[i]=0; //обнуление
    pole1[i]=(full_length>>(31-i))&0x1;
}
```

Так как под хранение максимального значения переменной `full_length` было решено отвести 4 байта, заполнение массива `pole1` происходит в 32 цикла по `i` с шагом 1. В такте цикла значение `i` – ой ячейки массива `pole1` обнуляется и приравнивается к значению переменной `full_length`, битово смещенной на 31 минус `i` позиций вправо и умноженному на маску `0x1`. Так очередной бит переменной `full_length` помещается в соответствующую ему ячейку массива `pole1`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`.

Затем происходит заполнение массива `pole2` битами значений, содержащихся в массиве `crypt_array`:

```
//заполнение массива pole2
```

```

for (i=0;i<text_length+dop_length;i++)
{
    pole2[0+i*8]=(crypt_array[i]>>7)&0x1;
    pole2[1+i*8]=(crypt_array[i]>>6)&0x1;
    pole2[2+i*8]=(crypt_array[i]>>5)&0x1;
    pole2[3+i*8]=(crypt_array[i]>>4)&0x1;
    pole2[4+i*8]=(crypt_array[i]>>3)&0x1;
    pole2[5+i*8]=(crypt_array[i]>>2)&0x1;
    pole2[6+i*8]=(crypt_array[i]>>1)&0x1;
    pole2[7+i*8]=crypt_array[i]&0x1;
}

```

В цикле по i от 0 до text_length плюс dop_length с шагом 1 ячейка массива pole2 с индексом $(0 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 7 позиций вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(1 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 6 позиций вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(2 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 5 позиций вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(3 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 4 позиции вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(4 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 3 позиции вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(5 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 2 позиции вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом $(6 \text{ плюс } i, \text{ умноженное на } 8)$ принимает значение результата битового сдвига на 1 позицию вправо и умножения на маску $0x1$ i – ой ячейки массива crypt_array .

Ячейка массива pole2 с индексом (7 плюс i, умноженное на 8) принимает значение результата умножения на маску 0x1 i – ой ячейки массива crypt_array.

Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i. Так происходит разбиение каждого значения, содержащегося в crypt_array (каждого байта зашифрованного текста) на отдельные биты, которые теперь содержатся в ячейках массива pole2.

Далее следует заполнение массива pole3:

```
//заполнение массива pole3
for (i=0;i<8;i++)
{
    pole3[i]=(dop_length>>(7-i))&0x1;
}
```

В цикле по i от 0 до 8 с шагом 1 значение i – ой ячейки массива pole3 становится равным результату битового сдвига на (7 минус i) позиций вправо и умножения на маску 0x1 значения переменной dop_length. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i. Теперь массив pole3 содержит в своих ячейках биты переменной dop_length (числа дополняющих символов).

После того, как массивы pole1, pole2 и pole3 сформированы, производится их соединение в один массив array_to_hide. Заполняются первые 4 байта (32 бита) из массива pole1:

```
//заполнение массива array_to_hide
//первые 4 байта
for (i=0;i<32;i++)
{
    array_to_hide[i]=pole1[i];
}
```

В цикле по i от 0 до 32 с шагом 1 в i – ю ячейку массива array_to_hide записывается значение i – ой ячейки массива pole1. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i.

После первых четырех байт в массив array_to_hide записываются биты массива pole2:

```
//байты [5 ... до предпоследнего]
for (i=32;i<(full_length*8)-8;i++)
```

```
{
    array_to_hide[i]=pole2[i-32];
}
```

В цикле по i от 32 до (full_length , умноженного на 8 и минус 8) с шагом 1 в i – ю ячейку массива `array_to_hide` записывается значение ячейки массива `pole2` с индексом i минус 32. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

Последний байт записывается из массива `pole3`:

```
//последний байт
for (i=(full_length*8)-8;i<(full_length*8);i++)
{
    array_to_hide[i]=pole3[i-((full_length*8)-8)];
}
```

В цикле по i от (full_length , умноженного на 8 и минус 8) до (full_length , умноженного на 8) с шагом 1 в i – ю ячейку массива `array_to_hide` записывается значение ячейки массива `pole3` с индексом (i минус (full_length , умноженного на 8 и минус 8)). Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

После описанных операций массив `array_to_hide` содержит биты длины зашифрованного текста, биты самого зашифрованного текста и биты числа символов дополнения. Данные биты подлежат скрытию в файле – контейнере.

3.3.8 Функция открытия файла контейнера

Объявляются переменные:

```
unsigned int wavheader_bigger[1000]; //массив для хранения
заголовка Wav
unsigned int data_position=0; //позиция слова data
unsigned int wavheader_length=0; //длина заголовка в байтах
int trash=0; // мусор
```

Далее открывается файл – контейнер `sound.wav` в режиме бинарного чтения:

```
//6.1 открытие файла контейнера для поиска позиции слова data
ifstream soundstream0("sound.wav",ios_base::binary);
if (!(soundstream0.is_open()))
{
    cout<<"ОШИБКА! Файл sound.wav не был открыт."<<endl;
    system ("pause");
}
```

```

        return -1;
    }

```

Если файл не был открыт, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл sound.wav не был открыт.» и завершает свою работу.

Если файл был открыт, происходит чтение байт из файла – контейнера:

```

//побайтное чтение в массив wavheader_bigger
for (i=0;i<1000;i++)
{
    wavheader_bigger[i]=0;
    soundstream0.read((char*)&wavheader_bigger[i],1);
}

```

В цикле по *i* от 0 до 1000 с шагом 1 значение *i* – ой ячейки массива `wavheader_bigger` обнуляется и в него заносится очередной байт из файла – контейнера. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика *i*.

Массив `wavheader_bigger` содержит первые 1000 байт файла – контейнера, в котором производится поиск байт слова `data`, означающих начало области звуковых данных:

```

//поиск в массиве wavheader_bigger байт слова data
for (i=0;i<1000;i++)
{
    data_position=i+1;
    if ((wavheader_bigger[i]==0x64)&&
        (wavheader_bigger[i+1]==0x61)&&(wavheader_bigger[i+2]==0x74)&&
        (wavheader_bigger[i+3]==0x61))
        break;
}

```

В цикле по *i* от 0 до 1000 с шагом 1 значение переменной `data_position` увеличивается на единицу. Происходит сравнение *i* – ой ячейки массива `wavheader_bigger` и последующих четырех со значениями `0x64`, `0x61`, `0x74`, `0x61` (символы `d`, `a`, `t`, `a`) соответственно, и если все они одновременно равны соответствующим значениям цикл прерывается. В противном случае такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика *i*. Переменная `data_position` содержит значение смещения относительно начала файла, с которого начинаются байты блока звуковых данных.

Затем объявляется и заполняется массив `wavheader_real`:

```

unsigned int *wavheader_real = new unsigned int
    [data_position+7]; // дин.массив длиной = длине заголовка (всё
    без сэмплов), содержащий байты заголовка
for (i=0; i<data_position+7; i++)
{
    wavheader_real[i]=wavheader_bigger[i];
}

```

Массив `wavheader_real` имеет `data_position` плюс 7 ячеек. В цикле по `i` от 0 до `data_position` плюс 7 с шагом 1 значение `i` – ой ячейки массива `wavheader_real` приравнивается к значению `i` – ой ячейки массива `wavheader_bigger`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Массив `wavheader_real` содержит байты файла – контейнера от начала до байт самих звуковых отсчетов.

Затем следует последовательность команд:

```

//определение длины даты (сэмплов)
AUDIO.subChunk2Size=wavheader_real[data_position+3]|wavheader_real
    [data_position+4]<<8|wavheader_real[data_position+5]<<16|wavhe
    ader_real[data_position+6]<<24;
//определение длины заголовка
wavheader_length=data_position+7;
//сброс каретки/возврат к началу файла
soundstream0.clear();
soundstream0.seekg(0);

```

Определяется значение переменной `subChunk2Size` как логическая сумма значений ячейки массива `wavheader_real` с индексом `data_position` плюс 3, ячейки массива `wavheader_real` с индексом `data_position` плюс 4 после битового сдвига на 8 позиций влево, ячейки массива `wavheader_real` с индексом `data_position` плюс 5 после битового сдвига на 16 позиций влево и ячейки массива `wavheader_real` с индексом `data_position` плюс 6 после битового сдвига на 24 позиции влево. В результате переменная `subChunk2Size` содержит значение длины области звуковых данных.

Переменная `wavheader_length` определяется как значение переменной `data_position` плюс 7 и содержит число байт, расположенных в файле – контейнере до байт звуковых отсчетов.

Позиция указателя перемещается в начало файла.

Затем происходит считывание из файла контейнера байт заголовка в соответствующие переменные структуры audio:

```
//чтение информации о контейнере в переменные структуры audio
soundstream0.read((char*)&AUDIO.chunkID,4);//Содержит символы
"RIFF" в ASCII кодировке (0x52494646 в big-endian
представлении). Является началом RIFF-цепочки.
soundstream0.read((char*)&AUDIO.chunkSize,4);//Это оставшийся
размер цепочки, начиная с этой позиции. Иначе говоря, это
размер файла - 8, то есть, исключены поля chunkID и chunkSize.
soundstream0.read((char*)&AUDIO.format,4);//Содержит символы
"WAVE" (0x57415645 в big-endian представлении)
soundstream0.read((char*)&AUDIO.subChunk1ID,4);//Содержит символы
"fmt " (0x666d7420 в big-endian представлении)
soundstream0.read((char*)&AUDIO.subChunk1Size,4);//16 для формата
PCM. Это оставшийся размер подцепочки, начиная с этой позиции.
soundstream0.read((char*)&AUDIO.audioFormat,2); //Аудио формат,
полный список можно получить здесь. Для PCM = 1 (то есть,
Линейное квантование). Значения, отличающиеся от 1, обозначают
некоторый формат сжатия.
soundstream0.read((char*)&AUDIO.numChannels,2);//Количество
каналов. Моно = 1, Стерео = 2 и т.д.
soundstream0.read((char*)&AUDIO.sampleRate,4);//Частота
дискретизации. 8000 Гц, 44100 Гц и т.д.
soundstream0.read((char*)&AUDIO.byteRate,4);//Количество байт,
переданных за секунду воспроизведения.
soundstream0.read((char*)&AUDIO.blockAlign,2);//Количество байт
для одного сэмпла, включая все каналы.
soundstream0.read((char*)&AUDIO.bitsPerSamle,2);//Количество бит в
сэмпле. Так называемая "глубина" или точность звучания. 8 бит,
16 бит и т.д.
soundstream0.read((char*)&AUDIO.subChunk2ID,4);//Содержит символы
"data" (0x64617461 в big-endian представлении)
soundstream0.read((char*)&trash,4);//Количество байт в области
данных.
```

В переменные chunkID, chunkSize, format, subChunk1ID, subChunk1Size, sampleRate, byteRate, subChunk2ID считывается по четыре соответствующих байта из заголовка, в переменные audioFormat, numChannels, blockAlign и bitsPerSamle по два. Число считываемых бит определяется структурой заголовка. Последним происходит считывание четырех байт в переменную trash. Это сделано для того, чтобы указатель переместился на 4 позиции право,

а значение переменной subChunk2Size, соответствующее этим четырем байтам, было определено ранее.

Следующая последовательность команд определяет длину и позицию начала байт, идущих после байт звуковых отсчетов:

```
//определение длины хвостовой части
tail_length=AUDIO.chunkSize+8-wavheader_length-
    AUDIO.subChunk2Size;
//определение позиции хвостовой части
tail_position=AUDIO.chunkSize+8-tail_length;
```

Переменная tail_length определяется по формуле:

$$tail\ length = chunkSize + 8 - wavheader\ length - subChunk2Size$$

Переменная tail_position определяется по формуле:

$$tail\ position = chunkSize + 8 - tail\ length$$

Далее происходит проверка на соответствие файла – контейнера:

```
//проверка на ошибки
//проверка формата RIFF
if (AUDIO.chunkID!=0x46464952)
{
    cout<<"ОШИБКА! Файл sound.wav не содержит RIFF
    контейнер."<<endl;
    system ("pause");
    return -1;
}
//проверка формата WAVE
if (AUDIO.format!=0x45564157)
{
    cout<<"ОШИБКА! Файл sound.wav не является WAVE файлом."<<endl;
    system ("pause");
    return -1;
}
//проверка на наличие формата PCM в audioFormat
if (AUDIO.audioFormat!=1)
{
```

```

        cout<<"ОШИБКА! Файл sound.wav не является форматом PCM WAVE
        файла."<<endl;
        system ("pause");
        return -1;
    }

```

Если значение переменной chunkID отличается от значения 0x46464952 (соответствующее символам RIFF), программа выводит на экран сообщение «ОШИБКА! Файл sound.wav не содержит RIFF контейнер.» и завершает свою работу. В противном случае проверяется значение переменной format. Если оно отличается от значения 0x45564157 (соответствующее символам WAVE), программа выводит на экран сообщение «ОШИБКА! Файл sound.wav не является WAVE файлом.» и завершает свою работу. В противном случае проверяется значение переменной audioFormat. Если оно отличается от значения 1 (соответствующее формату PCM), программа выводит на экран сообщение «ОШИБКА! Файл sound.wav не является форматом PCM WAVE файла.» и завершает свою работу. В противном случае, все три проверяемые переменные удовлетворяют необходимым условиям. Это значит, что файл – контейнер удовлетворяет требованиям и может быть использован для скрытия.

3.3.9 Функция подсчета необходимой информации по стеганографическому контейнеру

Данная функция предназначена для определения переменных, значения которых будут необходимы для скрытия информации в файле – контейнере:

```

//ПОДСЧЕТ НЕОБХОДИМОЙ ИНФОРМАЦИИ ПО КОНТЕЙНЕРУ
//определение числа звуковых отсчетов
samples = ((AUDIO.subChunk2Size)/AUDIO.blockAlign);
//определение числа байт для хранения звуковых отсчетов
subsamples = AUDIO.subChunk2Size;
//определение вместимости контейнера(в символах) при замене 1
    младшего бита
can_hide=samples/8/(AUDIO.blockAlign/AUDIO.numChannels); //универсальная ф-ла, подходит для моно и стерео

```

Определяется значение переменной samples, означающее число звуковых отсчетов (включая все имеющиеся каналы), содержащиеся в файле – контейнере по формуле:

$$samples = \frac{subChunk2Size}{blockAlign}$$

Определяется значение переменной `subsamples`, равное значению переменной `subChunk2Size` и обозначающее число байт, отводимых под хранение всех звуковых отсчетов, имеющихся в файле – контейнере. Кроме того, определяется вместимость файла – контейнера в переменную `can_hide` по формуле:

$$can\ hide = \frac{\frac{subsamples}{8}}{\frac{blockAlign}{numChannels}}$$

Затем, на основе полученного значения переменной `can_hide` производится проверка на возможность скрытия:

```
//уместится ли array_to_hide в контейнере?
if (full_length > can_hide)
{
    cout<<"ОШИБКА! Шифротекст ("<<full_length-5<<" байт) не
    помещается в контейнер."<<endl;
    cout<<"Контейнер может вместить только "<<can_hide-5<<" байт
    шифротекста."<<endl;
    system ("pause");
    return -1;
}
```

Если значение переменной `full_length` больше значения переменной `can_hide` программа выводит на экран сообщения: «ОШИБКА! Шифротекст (X байт) не помещается в контейнер.», «Контейнер может вместить только Y байт шифротекста.» (где X – значение переменной `full_length` минус 5, Y – значение переменной `can_hide` минус 5) и завершает свою работу. Если значение `full_length` не больше значения переменной `can_hide`, происходит перемещение внутреннего указателя файле – контейнере в область начала звуковых

отсчетов, значение начальной позиции которой содержится в переменной `wavheader_length`:

```
//переход каретки к данным
soundstream0.clear();
soundstream0.seekg(wavheader_length);
```

3.3.10 Функция чтения байт данных стеганографического контейнера

Производится чтение байт звуковых отсчетов:

```
//ЧТЕНИЕ СУБСЕМПЛОВ
unsigned int *subsamples_array = new unsigned int
    [subsamples]; //создание динамического массива для хранения
    байт субсэмпллов
for (i=0; i<subsamples; i++)
{
    subsamples_array[i]=0; //обнуление
    soundstream0.read((char*)&subsamples_array[i], 1);
}
```

Объявляется массив `subsamples_array` длиной `subsamples` ячеек для хранения байт звуковых отсчетов. В цикле по `i` от 0 до `subsamples` с шагом 1 значение `i` – ой ячейки массива `subsamples_array` обнуляется и в него считывается очередной байт из файла – контейнера из области данных, где расположены звуковые отсчеты. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`.

Также объявляется массив `tail_array` длиной `tail_length` ячеек для хранения байт, расположенных в файле контейнере после байт звуковых отсчетов и заполняется:

```
//чтение хвоста
unsigned int *tail_array = new unsigned int
    [tail_length]; //создание динамического массива для хранения
    байт хвоста
for (i=0; i<tail_length; i++)
{
    tail_array[i]=0; //обнуление
    soundstream0.read((char*)&tail_array[i], 1);
}
soundstream0.close();
```

В цикле по `i` от 0 до `tail_length` с шагом 1 значение `i` – ой ячейки массива `tail_array` обнуляется и в него считывается очередной байт из файла – контейнера

из области, расположенной после байт звуковых отсчетов. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика *i*.

3.3.11 Функция определения последовательности байт стеганографического контейнера для скрытия

В начале данной функции происходит инициализация необходимых массивов:

```
//ГЕНЕРАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ НОМЕРОВ СУБСЭМПЛОВ, ПОДЛЕЖАЩИХ  
ЗАМЕНЕ  
unsigned long int *subsamples_nums = new unsigned long int  
[subsamples]; //создание динамического массива для хранения  
номеров субсэмплов для 8 битных wav  
unsigned long int *subsamples_nums2 = new unsigned long int  
[subsamples]; //создание динамического массива для хранения  
номеров субсэмплов, где будут заменены младшие биты . для не 8  
битных wav  
unsigned int *subsamples_status = new unsigned int  
[subsamples]; //0 – субсэмпл свободен, 1 – занят
```

Инициализируется массив `subsamples_nums` длиной `subsamples` для хранения последовательности номеров звуковых отсчетов в которой будет происходить замена младших бит для 8 – битного файла – контейнера.

Инициализируется массив `subsamples_nums2` длиной `subsamples` для хранения последовательности номеров звуковых отсчетов в которой будет происходить замена младших бит для файла – контейнера, битность которого отличается от 8.

Инициализируется массив `subsamples_status` длиной `subsamples` для хранения значений, определяющих статусов номеров звуковых отсчетов в которой будет происходить замена младших бит (занят/ не занят). Это сделано для того, чтобы в дальнейшем, при скитии замена младших бит происходила в разных байтах звуковых отсчетов.

В случае, если число байт звуковых отсчетов больше 1000000 программа выводит информационное сообщение «Подождите...»:

```
if (subsamples>1000000)  
    cout<<"Подождите..."<<endl;
```

Далее ячейки массивов `subsamples_status` и `subsamples_nums` обнуляются в цикле по `i` от 0 до `subsamples` с шагом 1.

```
//обнуление subsamples_status, subsamples_nums2
for (i=0;i<subsamples;i++)
{
    subsamples_status[i]=0;
    subsamples_nums2[i]=0;
}
```

После происходит непосредственно генерация последовательности байт для скрытия. Определяется значение номера первого байта для скрытия:

```
//генератор
//начальное заполнение
subsamples_nums[0]=subsamples/2;
subsamples_status[subsamples_nums[0]]++;
```

Нулевая ячейка массива `subsamples_nums` становится равной значению переменной `subsamples` деленному на 2, а значение ячейки `subsamples_status` с индексом равным значению нулевой ячейки массива `subsamples_nums` увеличивается на единицу. Таким образом определен номер первого байта для скрытия, а в соответствующей ячейке массива `subsamples_status` для его установлено значение отличное от нуля. Это означает что данный номер байта для скрытия был только что занят и в дальнейшем следует избегать появления в массиве `subsamples_nums` такого же значения.

Далее происходит заполнение остальных ячеек массивов `subsamples_nums` и `subsamples_status`:

```
//далее заполнение массива
for (i=1;i<subsamples;i++)
{
    subsamples_nums[i]=subsamples_nums[i-1]+(0+rand()%subsamples);
    //если переполнение
    if (subsamples_nums[i]>subsamples)
        subsamples_nums[i]=subsamples_nums[i]%subsamples;
    again:
    if (subsamples_status[subsamples_nums[i]]!=0)
    {
        subsamples_nums[i]=subsamples_nums[i]+(0+rand()%subsamples);
        if (subsamples_nums[i]>subsamples)
            subsamples_nums[i]=subsamples_nums[i]%subsamples;
        subsamples_nums[i]--;
    }
}
```

```

        goto again;
    }
    subsamples_status[subsamples_nums[i]]++;
}

```

В цикле по i от 0 до `subsamples` с шагом 1 значение i – ой ячейки массива `subsamples_nums` определяется как значение предыдущей ячейки этого массива плюс случайное число, сгенерированное программой. Если получившееся значения оказалось больше значения `subsamples` (общего количества байт звуковых отсчетов) оно становится равным остатку от деления на значение `subsamples`. Проверяется значение соответствующей ячейки массива `subsamples_status` на равенство нулю. Если оно не равно нулю, необходимо сгенерировать другое число, так как данное уже имеется в массиве `subsamples_nums`. Значение i – ой ячейки массива `subsamples_nums` определяется как значение уже текущей ячейки (только что сгенерированное) плюс случайное число, сгенерированное программой. Если получившееся значения оказалось больше значения `subsamples` (общего количества байт звуковых отсчетов) оно становится равным остатку от деления на значение `subsamples` и уменьшается на единицу, после чего программа переходит на метку `again`. Если сгенерированное число не встречалось ранее, оно занесется в i – ю ячейку массива `subsamples_nums`, а значение ячейки массива `subsamples_status` с индексом, определяемым содержимым i – ой ячейки массива `subsamples_nums` увеличивается на единицу. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

После заполнения массива `subsamples_nums` заполняется массив `subsamples_nums2`:

```

//заполнение subsamples_nums2
j=0;
for (i=0;i<subsamples;i++)
{
    if
    ((subsamples_nums[i])%(AUDIO.blockAlign/AUDIO.numChannels)==0)
    {
        subsamples_nums2[j]=subsamples_nums[i];
        j++;
    }
}

```

```

    }
}

```

Переменная *j* равна перед выполнением цикла нулю. В цикле по *i* от 0 до *subsamples* с шагом 1 в случае если остаток от деления содержимого *i* – ой ячейки массива *subsamples_nums* на частное переменных *blockAlign* и *numChannels* равно нулю значение *j* – ой ячейки массива *subsamples_nums2* приравнивается к значению *i* – ой ячейки массива *subsamples_nums*, а переменная *j* увеличивается на единицу. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика *i*. Таким образом, массив *subsamples_nums2* содержит только те значения байт из массива *subsamples_nums*, которые являются младшими для файла – контейнера, битность которого отличается от 8.

3.3.12 Функция скрывтия зашифрованного текста и создания выходного файла

Производится замена младших бит в байтах звуковых отсчетов файла – контейнера, содержащихся в массиве *subsamples_array*:

```

//ЗАМЕНА МЛАДШИХ БИТ В СУБСЭМПЛАХ/БАЙТАХ
for (i=0;i<full_length*8;i++)
{
    //отброс старого младшего бита и внесение нового из
    array_to_hide
if (AUDIO.blockAlign==1)//для 8 бит
    subsamples_array[subsamples_nums[i]]=(subsamples_array[subsamples_nums[i]]&0xfe)|array_to_hide[i];
else //if (AUDIO.blockAlign==2)//для остальных
    subsamples_array[subsamples_nums2[i]]=(subsamples_array[subsamples_nums2[i]]&0xfe)|array_to_hide[i];
}

```

В цикле по *i* от 0 до *full_length*, умноженного на 8 с шагом 1 производится проверка переменной *blockAlign*. Если ее значение равно 1 значение ячейки массива *subsamples_array* с индексом, определяемым содержимым *i* – ой ячейки массива *subsamples_nums* умножается на маску 0xfe и логически суммируется со значением *i* – ой ячейки массива *array_to_hide* (младший бит байта

звукового отсчета заменяется битом подготовленного ранее зашифрованного сообщения).

Если значение переменной `blockAlign` отличается от 1 значение ячейки массива `subsamples_array` с индексом, определяемым содержимым `i` – ой ячейки массива `subsamples_nums2` умножается на маску `0xfe` и логически суммируется со значением `i` – ой ячейки массива `array_to_hide`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`.

После изменения младших бит следует создание выходного файла и запись в него содержимого:

```
//СОЗДАНИЕ ВЫХОДНОГО ФАЙЛА
ofstream ofstr("s_sound.wav", ios_base::out | ios::binary
    | ios_base::trunc); //создание выходного файла
//запись заголовка
for (i=0; i<wavheader_length; i++)
{
    ofstr.write((char*)&wavheader_real[i], 1);
}
//запись сэмплов
for (i=0; i<subsamples; i++)
{
    ofstr.write((char*)&subsamples_array[i], 1);
}
//запись хвоста если он есть
if (tail_length!=0)
    for (i=0; i<tail_length; i++)
    {
        ofstr.write((char*)&tail_array[i], 1);
    }
//закрытие файла
ofstr.close();
```

Создается файл `s_sound.wav` в режиме записи. В цикле по `i` от 0 до `wavheader_length` с шагом 1 записывается очередной байт заголовка, содержащийся в `i` – ой ячейке массива `wavheader_real`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Так в выходной файл записывается заголовок.

Затем в цикле по `i` от 0 до `subsamples` с шагом 1 записывается очередной байт звукового отсчета (имеющий на месте младшего бита новый бит

скрываемой информации), содержащегося в i – ой ячейке массива `subsamples_array`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . Так в выходной файл после байт заголовка записываются байты звуковых отсчетов.

После записи байт звуковых отсчетов следует запись последних байт, которые могут быть расположены в конце исходного файла – контейнера. Проверяется наличие таких байт путем сравнения значения переменной `tail_length` с нулем. Если оно не равно нулю, то выполняется цикл по i от 0 до `tail_length` с шагом 1. В цикле в выходной файл записывается значение i – ой ячейки массива `tail_array`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . Байты выходного файла окончательно сформированы и он закрывается.

После последовательного выполнения всех перечисленных функций программа выводит информационное сообщение: «Сообщение успешно зашифровано и скрыто!» и завершает свою работу. В результате, на выходе программы получился файл `s_sound.wav`, содержащий в байтах своих звуковых отсчетов младшие биты, являющиеся битами зашифрованного сообщения. В остальном файл полностью идентичен своему исходному файлу. Данный файл создается в каталоге, где находится программа создавшая его.

3.4 Разработка и описание алгоритма программы, реализующей извлечение и расшифрование скрытых сообщений

Был разработан алгоритм программы, реализующей извлечение сообщений, зашифрованных согласно алгоритму ГОСТ 28147-89[9] в режиме простой замены, скрытых в файле – контейнере формата `wav` и его расшифрование. Программа разделена на последовательно выполняемые условные функции:

- открытие файла стеганографического контейнера;
- подсчет необходимой информации по стеганографическому контейнеру;

- чтение байт данных стеганографического контейнера;
- определение последовательности байт стеганографического контейнера для извлечения;
- извлечение зашифрованного текста;
- открытие и чтение файла ключа;
- формирование КЗУ;
- подготовка зашифрованного текста к расшифрованию;
- расшифрование зашифрованного текста;
- создание выходного файла и запись в него расшифрованного текста.

Общий алгоритм программы представлен на рисунке 24.

В начале происходит инициализация начальных переменных для извлечения. Происходит открытие файла – контейнера, содержащего скрытый зашифрованный текст. На основании открытого файла подсчитывается необходимая информация, после чего происходит его побайтовое чтение. Определяется последовательность байт в области звуковых данных файла – контейнера и, согласно ей, извлекаются из них биты зашифрованного текста. Затем происходит инициализация начальных переменных для расшифрования. Производится открытие и чтение файла ключа для последующего формирования КЗУ. После того, как КЗУ было сформировано, происходит подготовка зашифрованного текста к расшифрованию и его расшифрование. После того, как текст расшифрован, создается выходной текстовый файл, содержащий в себе текст извлеченного и расшифрованного сообщения. Программа выводит на экран сообщение об успешном завершении извлечения и расшифрования.



Рисунок 24 – Общий алгоритм программы, реализующей извлечение и расшифрование сообщений

Алгоритм стеганографического извлечения информации представлен на рисунке 25.



Рисунок 25 – Алгоритм стеганографического извлечения информации

Алгоритм функции открытия стеганографического контейнера представлен на рисунке 26. Происходит открытие файла – контейнера. Если этого не произошло, программа выводит на экран сообщение об ошибке и завершает свою работу. Если файл успешно открыт происходит чтение первых 1000 байт в ячейки массива `wavheader_bigger`, после чего в цикле в данных ячейках происходит поиск четырех ячеек, последовательно, друг за другом содержащие коды символов `d`, `a`, `t`, `a`. Когда поиск завершен переменная `data_position` будет содержать номер байта, с которого начинается блок данных (позиция символа `d`).

Рисунок 26 – Алгоритм функции открытия стеганографического контейнера

Затем в цикле в ячейки массива `wavheader_real`, заносятся значения ячеек массива `wavheader_bigger` до `data_posotion` плюс 7 (первые байты файла – контейнера до байт звуковых отсчетов). Определяется число байт, отводимых для хранения звуковых отсчетов и длина заголовка файла – контейнера.

После происходит чтение байт заголовка с занесением в соответствующие переменные. Файл – контейнер проверяется на соответствие формату. В случае если файл не содержит RIFF – контейнер или не является WAVE – файлом или не соответствует формату PCM (импульсно – кодовой модуляции) программа выводит сообщение об ошибке и завершает свою работу.

После выполнения данной функции идет функция подсчета необходимой информации по стеганографическому контейнеру, алгоритм которой представлен на рисунке 27.



Рисунок 27 – Алгоритм функции подсчета необходимой информации по стеганографическому контейнеру

Происходит определение числа отсчетов звуковых сигналов и общего числа байт, отведенных для их хранения. Алгоритм функции чтения байт данных стеганографического контейнера представлен на рисунке 28.

Объявляется массив `subsamples_array` и в цикле в его ячейки заносятся байты звуковых отсчетов.

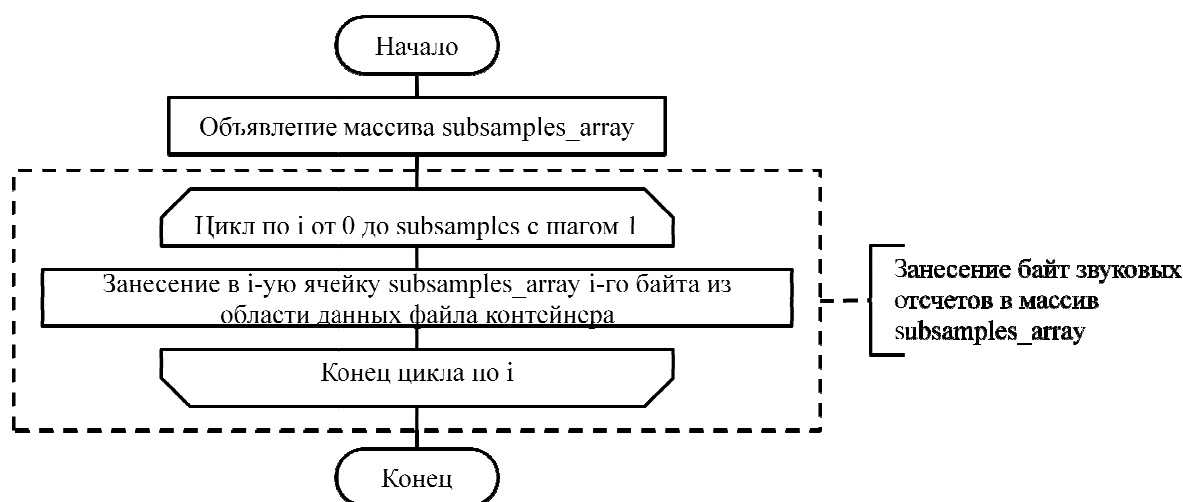


Рисунок 28 – Алгоритм функции чтения байт данных стеганографического контейнера

Алгоритм функции определения последовательности байт стеганографического контейнера для извлечения полностью идентичен алгоритму данной функции в программе, реализующей зашифрование и скрывание и представлен на рисунке 29.

Происходит инициализация массивов `subsamples_nums`, `subsamples_nums2` и `subsamples_status`. Первые два предназначены для хранения последовательности номеров звуковых отсчетов 8 – битного и не 8 – битного файла – контейнера соответственно, в которые будут внедрены биты зашифрованного текста. Значения ячеек массивов `subsamples_nums2` и `subsamples_status` обнуляются.

Если число байт, отводимых под хранение звуковых отсчетов превышает 1000000, программа выводит на экран информационное сообщение «Подождите...».

В начальную ячейку массива `subsamples_nums` вносится значение определяемое как `subsamples` деленное на 2, а начальная ячейка массива `subsamples_status` увеличивается на единицу. Далее происходит цикл для определения значения каждой из последующих ячеек массива `subsamples_nums`.

В такте цикла происходит вычисление значения текущей по счету ячейки массива `subsamples_nums` как значение предыдущей ячейки плюс псевдослучайное число.

Рисунок 29 – Алгоритм функции определения последовательности байт
стеганографического контейнера для извлечения

Если значение текущей ячейки оказалось выше значения числа байт звуковых отсчетов (переменная `subsamples`) данное значение претерпевает деление по модулю `subsamples`. Далее проверяется, было ли такое значение сгенерировано ранее путем сравнения значения ячейки массива `subsamples_status` (имеющий индекс, равный сгенерированному значению) с нулем (для того чтобы избежать повторения значений последовательностей байт, подлежащих замене). Если в рассматриваемой ячейке массива `subsamples_status` находится значение, отличное от нуля, это значит, что сгенерированное число уже было сгенерировано ранее и необходимо сгенерировать новое число отличное от каждого, сгенерированного ранее. Поэтому снова происходит вычисление значения текущей по счету ячейки массива `subsamples_nums` как значение уже текущего сгенерированного значения плюс псевдослучайное число. Если значение текущей ячейки оказалось выше значения числа байт звуковых отсчетов (переменная `subsamples`) данное значение претерпевает деление по модулю `subsamples`. Соответствующая ячейка массива `subsamples_status` уменьшается на единицу, а функция возвращается к месту первого сравнения значения ячейки массива `subsamples_status` с нулем.

Если значение рассматриваемой ячейки массива `subsamples_status` равно нулю, то это значит, что такого числа еще не было сгенерировано и значение данной ячейки увеличивается на единицу, чтобы в следующем такте цикла не было сгенерировано число, сгенерированное в данном такте цикла. Такт цикла завершается.

Таким образом, формируется содержимое массива `subsamples_nums`, содержащего в своих ячейках номера байт, определяющие последовательность байт в области данных 8 – битного файла – контейнера, в которой будет происходить извлечение бит зашифрованного текста.

Также производится и формирование значения ячеек массива `subsamples_nums2`, который по своему назначению схож с `subsamples_nums` а отличается содержимым, предназначенным для извлечения в файлах, битность

которых отличается от 8. Данный массив принимает из массива `subsamples_nums` только те значения, которые кратны числу байт, отводимых в файле – контейнере под один звуковой отсчет.

Так как число байт для хранения звуковых отсчетов в файле – контейнере до и после скртия одинаковое, значения переменных `subsamples`, вычисляемые в программах для зашифрования с последующим скртием и извлечения с последующим расшифроанием одинаковы. Это значит, что и последовательность байт для скртия/извлечения в данных программах генерируются одинаковые.

Алгоритм извлечения зашифрованного текста приведен на рисунке 30.

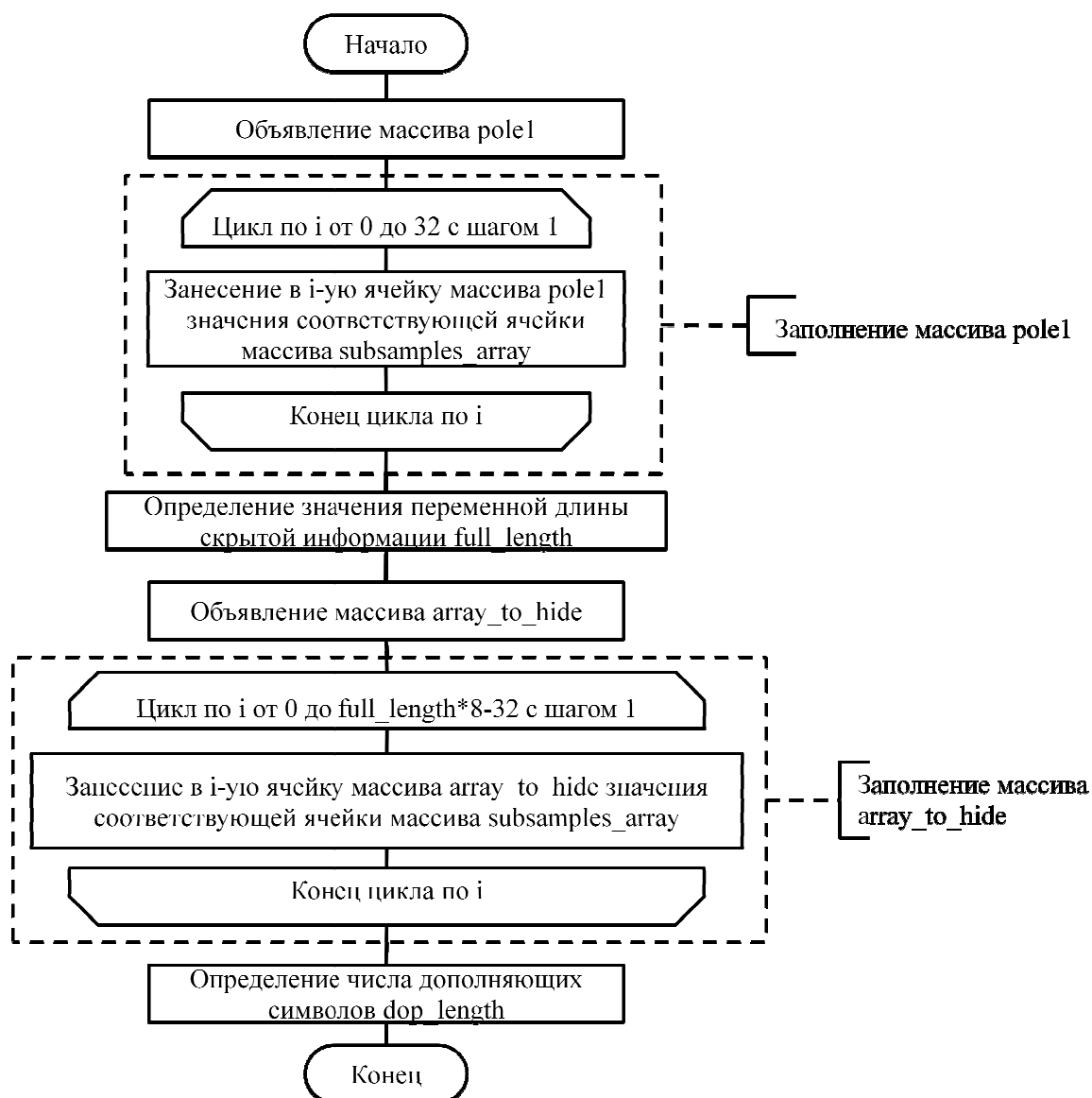


Рисунок 30 – Алгоритм извлечения зашифрованного текста

Для того, чтобы извлечь скрытый зашифрованный текст в оригинале, необходимо знать его длину. Кроме того, после расшифрования извлеченного зашифрованного текста открытый текст может содержать дополняющие символы, внедренные ранее. В итоге, чтобы получить идентичный открытый текст необходимо также знать число дополняющих символов.

Объявляется массив `pole1`, и заполняется значениями первых 32 ячеек массива `subsamples_array`. Из значений ячеек массива `pole1` формируется значения переменной `full_length` (длина зашифрованного текста с учетом значения его длины и числа дополняющих символов).

Объявляется массив `array_to_hide` и поочередно заполняется значениями из массива `subsamples_array` после игнорируя его первые 32 ячейки. Так массив `array_to_hide` будет содержать биты зашифрованного сообщения.

В конце функции подсчитывается значение числа дополняющих символов в переменную `dop_length`.

Алгоритм функции открытия и чтения файла ключа полностью идентичен алгоритму данной функции в программе, реализующей зашифрование и скрывание и представлен на рисунке 31.

Происходит открытие файла ключа. Если по каким-либо причинам этого не происходит, функция выводит на экран сообщение об ошибке и программа завершает свою работу.

В случае если файл ключа успешно открыт происходит проверка длины ключа. Если ключ не содержит ровно 256 бит, функция выводит на экран сообщение об ошибке и программа завершает свою работу. В противном случае создается массив для хранения кодов символов, содержащихся в файле ключа, куда и происходит занесение данных кодов. После произведенных операций происходит закрытие файла ключа.

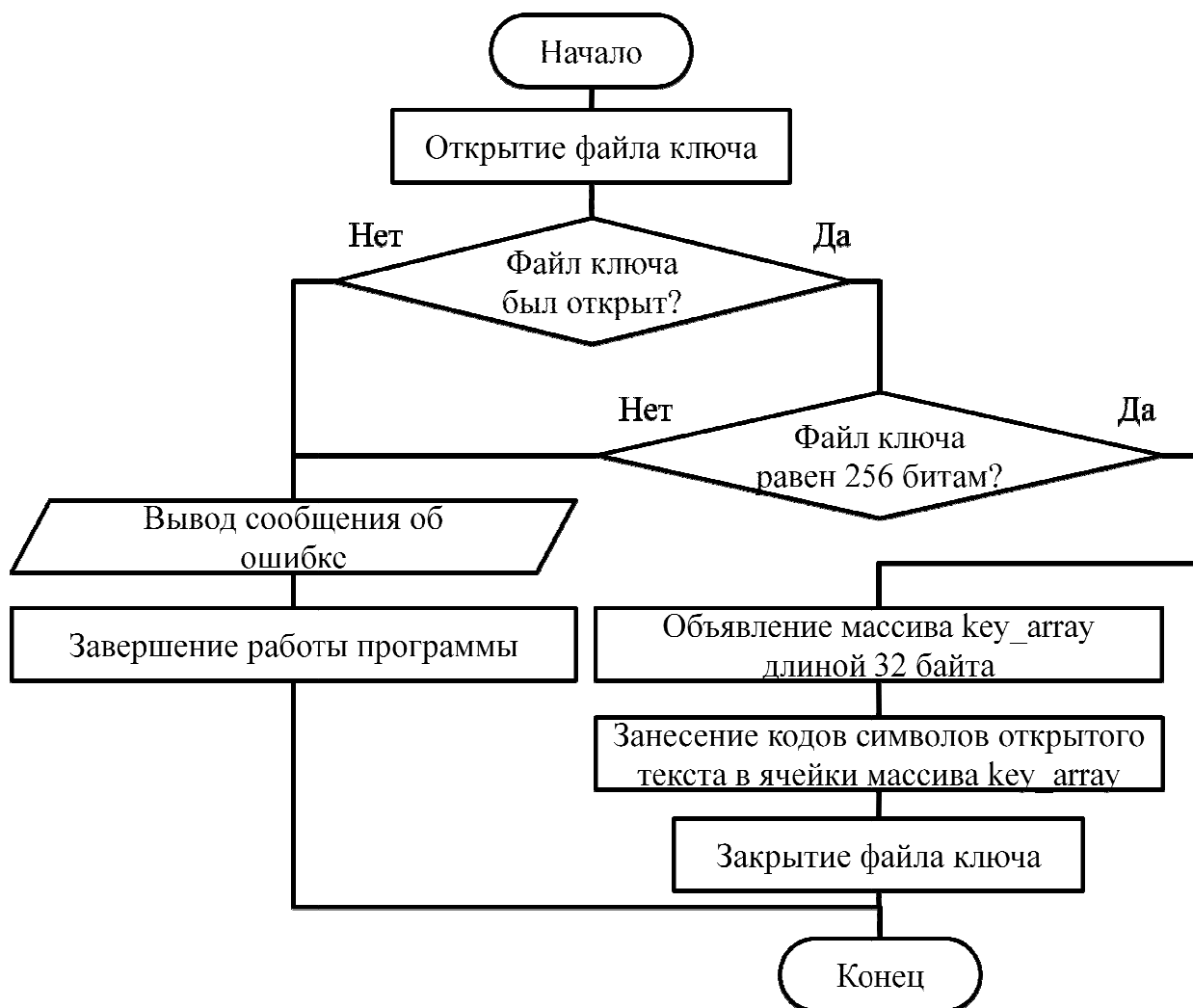


Рисунок 31 – Алгоритм функции открытия и чтения файла ключа
Алгоритм функции формирования КЗУ представлен на рисунке 32.



Рисунок 32 – Алгоритм функции формирования КЗУ

Функция распределяет 256 бит ключевой информации по 32 бита в переменные $x_0, x_1, \dots, x_7$. Таким образом, КЗУ представляет собой восемь 32-рядных накопителей.

Создается массив `x_array`, ячейки которого заполняются значениями переменных $x_0, x_1, \dots, x_7$ установленным алгоритмом шифрования в следующем порядке: $X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0, X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0$.

Алгоритм функции подготовки зашифрованного текста к расшифрованию приведен на рисунке 33.

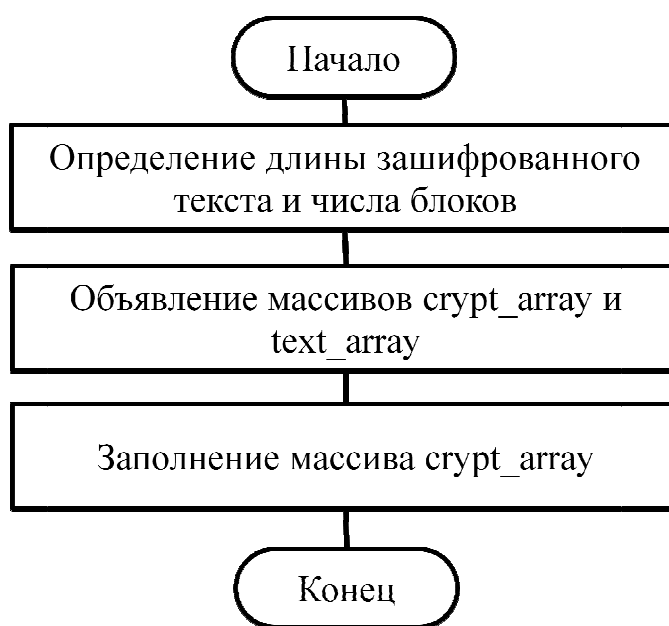


Рисунок 33 – Алгоритм функции подготовки зашифрованного текста к расшифрованию

Происходит вычисление длины текста зашифрованного сообщения (без байт его длины и числа дополняющих символов). Объявляются массивы `crypt_array` и `text_array`, массив `crypt_array` заполняется и в итоге содержит байты символов зашифрованного текста с учетом дополняющих символов.

Криптосхема алгоритма расшифрования в режиме простой замены по ГОСТ 28147-89[9] изображена на рисунке 34.

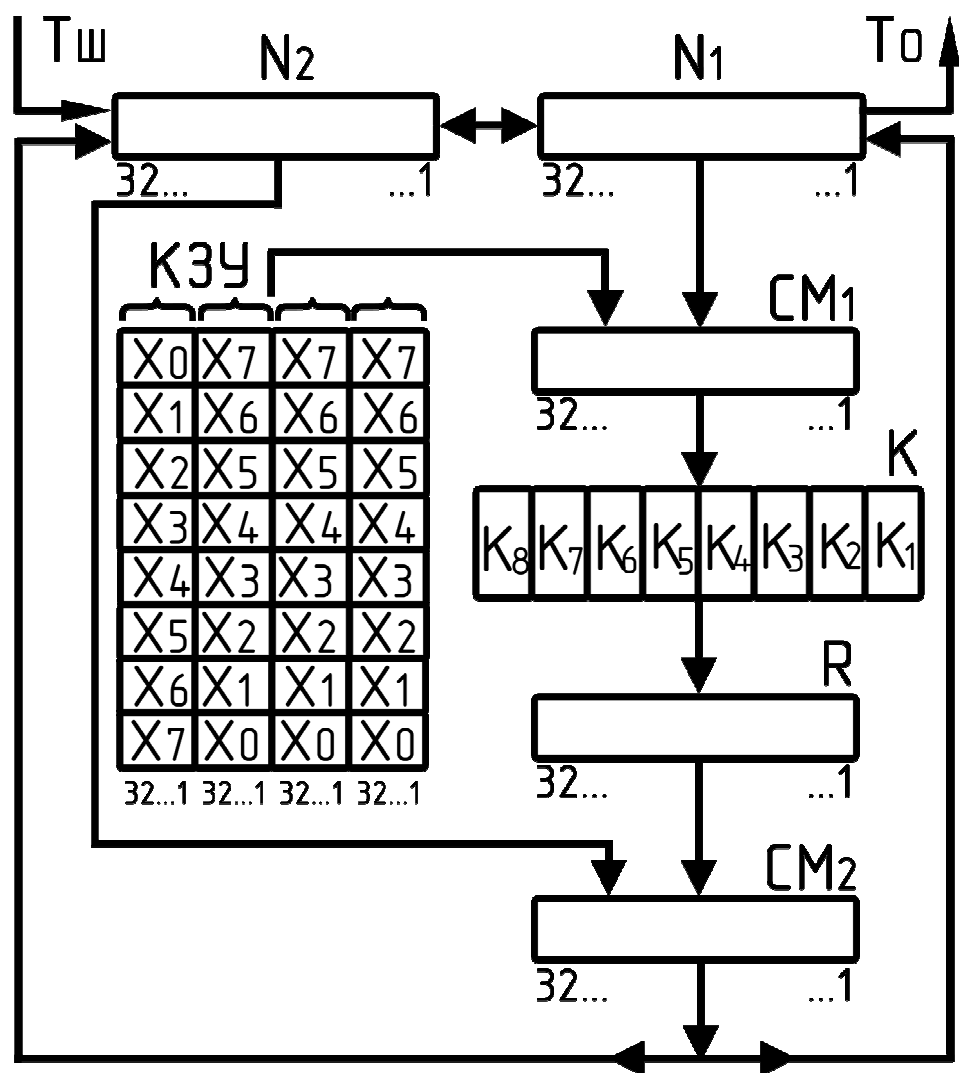


Рисунок 34 – Криптосхема алгоритма расшифрования в режиме простой замены по ГОСТ 28147-89[9]

Далее следует алгоритм функции расшифрования, алгоритм которой приведен на рисунке 35.

Расшифрование производится для каждого отдельного блока бит из массива текста `crypt_array`. Согласно алгоритму шифрования ГОСТ 28147-89[9] в режиме простой замены расшифрование каждого блока осуществляется в 32 цикла. Перед выполнением первого цикла формируется начальное заполнение накопителей N_1 и N_2 таким образом, что первая половина бит (32 бита) блока записываются в N_1 а вторая половина в N_2 . Далее идет выполнение циклов расшифрования.

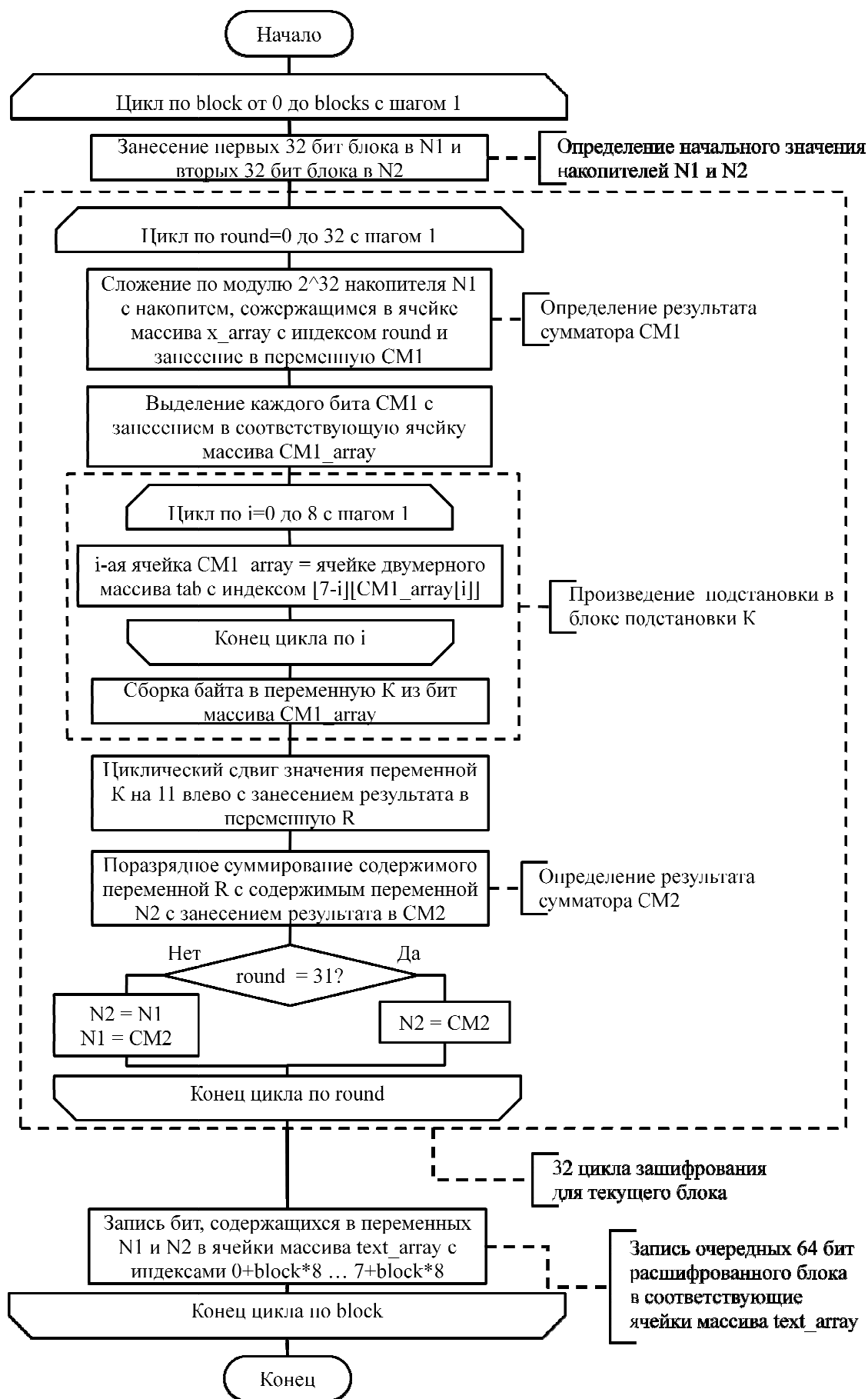


Рисунок 35 – Алгоритм функции расшифрования

Определяется результат сумматора CM_1 – значение накопителя N_1 складывается по модулю 2^{32} со значением накопителя, содержащимся в ячейке массива x_array с соответствующим номеру цикла индексом. Затем содержимое сумматора CM_1 разбивается на отдельные четверки бит, которые заносятся в соответствующие ячейки массива $CM1_array$.

Затем в узле подстановки происходит подстановка каждой четверки бит из ячеек массива $CM1_array$ в соответствии с таблицей замены. После подстановки четверки бит объединяются в 32 – битное значение и оно заносится в переменную K . Содержимое переменной K претерпевает циклический сдвиг на 11 в сторону старших разрядов и заносится в переменную R .

Содержимое переменной R поразрядно складывается с содержимым N_2 по модулю 2 и, являясь результатом сумматора CM_2 , заносится в соответствующую переменную.

После произведенных операций производится проверка на то, является ли текущий цикл расшифрования для блока последним (32-ым). В случае если это так, значение накопителя N_2 приравнивается к значению результата сумматора CM_2 . В противном случае значение накопителя N_2 приравнивается к значению N_1 , а тот, в свою очередь, приравнивается к значению результата сумматора CM_2 . На этом один из циклов расшифрования заканчивается. Данный цикл повторяется для каждого блока 32 раза. Важно отметить, что при расшифровании в циклах с 1 – го по 8 – ой накопители из массива x_array ($K3Y$) подаются в прямом порядке – $X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7$. В циклах с 9 – го по 16 – ый, с 17 – го по 24 – ый и с 25 – го по 32 – ой порядок обратный – $X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0$.

После того, как 32 цикла расшифрования закончились накопители N_1 и N_2 содержат половины бит зашифрованного блока. Данные 64 бита записываются в соответствующие ячейки массива $text_array$.

Описанные процедуры повторяются для каждого блока и в конце выполнения функции шифрования массив `text_array` будет содержать блоки расшифрованного текста с дополняющими символами.

Далее идет функция создания выходного файла и записи в него расшифрованного сообщения, алгоритм которой представлен на рисунке 36.

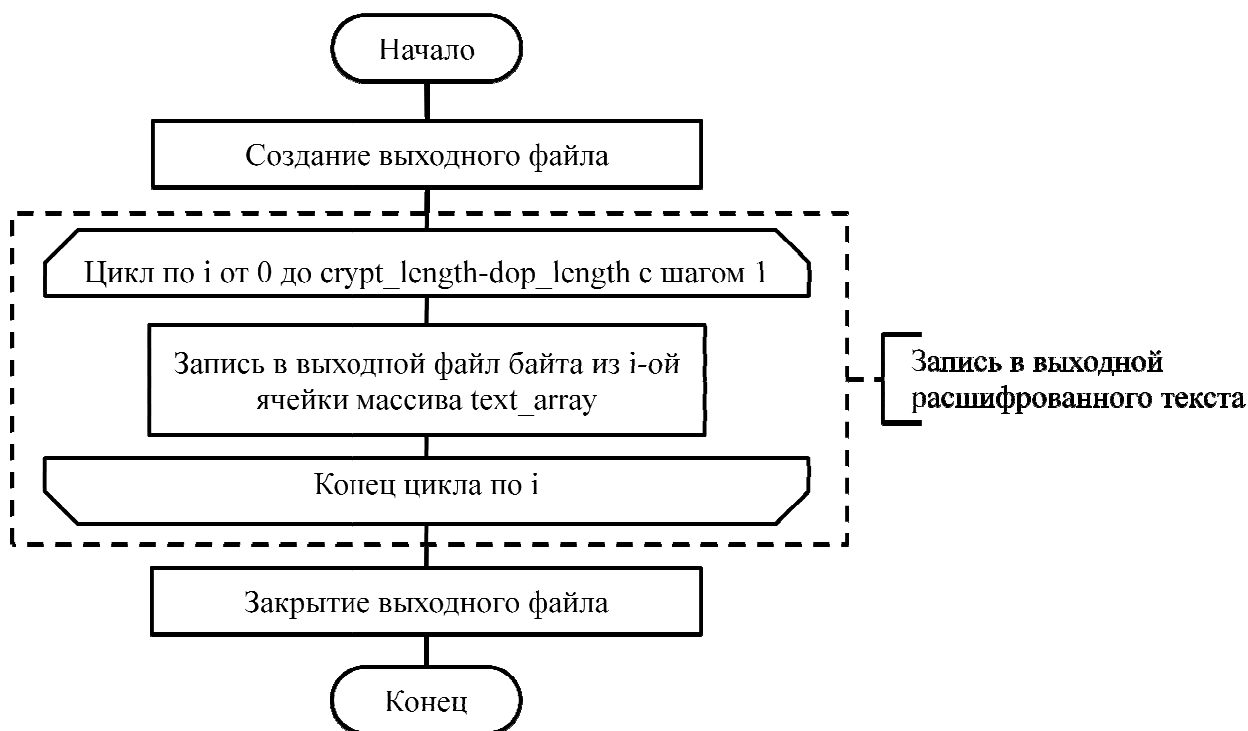


Рисунок 36 – Алгоритм функции создания выходного файла и записи в него расшифрованного сообщения

Создается выходной текстовый файл и в него поочередно записывается содержимое ячеек массива `text_array` за исключением последних байт, содержащих дополняющие символы, число которых определяется переменной `dop_length`. Файл закрывается. В результате созданный файл должен содержать в себе текст извлеченного и расшифрованного сообщения, который был зашифрован и скрыт с помощью первой разработанной программы.

3.5 Разработка программы, реализующей извлечение и расшифрование сообщений

3.5.1 Описание начальных переменных

В самом начале, вне тела функции `main()` происходит объявление структуры `audio`, полностью аналогичной одноименной структуре, содержащейся в первой программе.

В теле функции `main()` объявляются начальные переменные:

```
//Инициализация начальных переменных
int i, j, round, block; //счетчики
unsigned int samples=0;
unsigned int subsamples=0;
unsigned int full_length=0;
unsigned int dop_length=0;
unsigned int text_length=0;
```

Данные переменные предназначены для:

- `i, j, k` – счетчики;
- `samples` – количество звуковых отсчетов (включая все каналы);
- `subsamples` – количество бит, отводимых под хранение отдельных звуковых отсчетов;
- `text_length` – длина текста сообщения в байтах;
- `dop_length` – число символов, дополняющих текст сообщения до кратности 64 битам;
- `full_length` – длина текста сообщения в байтах с учетом дополняющих символов.

Далее в тексте программы объявляются следующие переменные:

```
long key_length=0; //длина ключа в символах/байтах
int crypt_length=0; //длина ш.т в символах/байтах
int blocks=0; //число блоков о.т по 64 бита
unsigned int x0, x1, x2, x3, x4, x5, x6, x7; //КЗУ
unsigned int x_array[32]; //массив в котором КЗУ выстроены в
    порядке нужном для зашифрования
unsigned int N1=0, N2=0; //накопители
unsigned int CM1=0, CM2=0;
unsigned int CM1_array[8]={0,0,0,0,0,0,0,0}; //массив для разбиения
    CM1 на части по 4 бита
```

```

unsigned int K=0; //значение CM1 после подстановки
unsigned int R=0, r=0;
int tab[8][16] = {
    {0x4, 0xA, 0x9, 0x2, 0xD, 0x8, 0x0, 0xE, 0x6, 0xB, 0x1, 0xC, 0x7, 0xF, 0x5, 0x3 }, //k1
    {0xE, 0xB, 0x4, 0xC, 0x6, 0xD, 0xF, 0xA, 0x2, 0x3, 0x8, 0x1, 0x0, 0x7, 0x5, 0x9 }, //k2
    {0x5, 0x8, 0x1, 0xD, 0xA, 0x3, 0x4, 0x2, 0xE, 0xF, 0xC, 0x7, 0x6, 0x0, 0x9, 0xB }, //k3
    {0x7, 0xD, 0xA, 0x1, 0x0, 0x8, 0x9, 0xF, 0xE, 0x4, 0x6, 0xC, 0xB, 0x2, 0x5, 0x3 }, //k4
    {0x6, 0xC, 0x7, 0x1, 0x5, 0xF, 0xD, 0x8, 0x4, 0xA, 0x9, 0xE, 0x0, 0x3, 0xB, 0x2 }, //k5
    {0x4, 0xB, 0xA, 0x0, 0x7, 0x2, 0x1, 0xD, 0x3, 0x6, 0x8, 0x5, 0x9, 0xC, 0xF, 0xE }, //k6
    {0xD, 0xB, 0x4, 0x1, 0x3, 0xF, 0x5, 0x9, 0x0, 0xA, 0xE, 0x7, 0x6, 0x8, 0x2, 0xC }, //k7
    {0x1, 0xF, 0xD, 0x0, 0x5, 0x7, 0xA, 0x4, 0x9, 0x2, 0x3, 0xE, 0x6, 0xB, 0x8, 0xC } //k8
}; //таблица замен Идентификатор: id-GostR3411-94-TestParamSet

```

Данные переменные предназначены для:

- round, block – счетчики для функции зашифрования;
- key_length – длина ключа в байтах;
- blocks – число блоков открытого текста с учетом дополняющих символов (по 64 бита каждый);

- x0, x1, x2, x3, x4, x5, x6, x7 – накопители КЗУ;
- x_array – массив в котором содержимое накопителей КЗУ выстроено в порядке нужном для зашифрования;

- N1, N2 – накопители;
- CM1, CM2 – результаты сумматоров;
- CM1_array – массив, в котором значение переменной CM1, разделенное на части по 4 бита;

- K – результат подстановки;
- R – результат циклического сдвига;
- r – старшая часть для циклического сдвига;
- tab – таблица замены.

Как видно, в целом предназначение перечисленных переменных совпадает с назначением одноименных переменных, содержащихся в первой программе. Остальные переменные будут объявляться на основании значений уже инициализированных переменных.

3.5.2 Функция открытия файла стеганографического контейнера

В ходе разработки программы, она была условно поделена на функции, каждая из которых представлена своей последовательностью команд, выполняющих свое функциональное предназначение. Данные функции представлены в неявном виде.

Функция открытия файла стеганографического контейнера аналогична одноименной функции, содержащейся в первой программе, за исключением отсутствия команд по определению смещения, с которого в файле – контейнере начинаются байты, идущие после байт звуковых данных и их длины.

Объявляются переменные:

```
unsigned int wavheader_bigger[1000]; //массив для хранения
заголовка Wav
unsigned int data_position=0; //позиция слова data
unsigned int wavheader_length=0; //длина заголовка в байтах
int trash=0; // мусор
```

Далее открывается файл – контейнер s_sound.wav, содержащий скрытое сообщение в режиме бинарного чтения:

```
//1.1 открытие файла контейнера для поиска позиции слова data
ifstream soundstream0("s_sound.wav",ios_base::binary);
if (!(soundstream0.is_open()))
{
    cout<<"ОШИБКА! Файл s_sound.wav не был открыт."<<endl;
    system ("pause");
    return -1;
}
```

Если файл не был открыт, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл s_sound.wav не был открыт.» и завершает свою работу. Если файл был открыт, происходит чтение байт из файла – контейнера:

```
//побайтное чтение в массив wavheader_bigger
for (i=0;i<1000;i++)
{
```

```

wavheader_bigger[i]=0;
soundstream0.read((char*)&wavheader_bigger[i],1);
}

```

В цикле по i от 0 до 1000 с шагом 1 значение i – ой ячейки массива `wavheader_bigger` обнуляется и в него заносится очередной байт из файла – контейнера. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

Массив `wavheader_bigger` содержит первые 1000 байт файла – контейнера, в котором производится поиск байт слова `data`, означающих начало области звуковых данных:

```

//поиск в массиве wavheader_bigger байт слова data
for (i=0;i<1000;i++)
{
    data_position=i+1;
    if ((wavheader_bigger[i]==0x64)&&(wavheader_bigger[i+1]==0x61)
    &&(wavheader_bigger[i+2]==0x74)&&(wavheader_bigger[i+3]==0x61)
    )
        break;
}

```

В цикле по i от 0 до 1000 с шагом 1 значение переменной `data_position` увеличивается на единицу. Происходит сравнение i – ой ячейки массива `wavheader_bigger` и последующих четырех со значениями 0x64, 0x61, 0x74, 0x61 (символы d, a, t, a) соответственно, и если все они одновременно равны соответствующим значениям цикл прерывается. В противном случае такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . Переменная `data_position` содержит значение смещения относительно начала файла, с которого начинаются байты блока звуковых данных.

Затем объявляется и заполняется массив `wavheader_real`:

```

unsigned int *wavheader_real = new unsigned int
[data_position+7]; // дин.массив длиной = длине заголовка (всё
без сэмплов), содержащий байты заголовка
for (i=0;i<data_position+7;i++)
{
    wavheader_real[i]=wavheader_bigger[i];
}

```

Массив `wavheader_real` имеет `data_position` плюс 7 ячеек. В цикле по `i` от 0 до `data_position` плюс 7 с шагом 1 значение `i` – ой ячейки массива `wavheader_real` приравнивается к значению `i` – ой ячейки массива `wavheader_bigger`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`. Массив `wavheader_real` содержит байты файла – контейнера от начала до байт самих звуковых отсчетов.

Затем следует последовательность команд:

```
//определение длины даты (сэмплов)
AUDIO.subChunk2Size=wavheader_real[data_position+3]|wavheader_real
    [data_position+4]<<8|wavheader_real[data_position+5]<<16|wavhe
    ader_real[data_position+6]<<24;
//определение длины заголовка
wavheader_length=data_position+7;
//сброс каретки/возврат к началу файла
soundstream0.clear();
soundstream0.seekg(0);
```

Определяется значение переменной `subChunk2Size` как логическая сумма значений ячейки массива `wavheader_real` с индексом `data_position` плюс 3, ячейки массива `wavheader_real` с индексом `data_position` плюс 4 после битового сдвига на 8 позиций влево, ячейки массива `wavheader_real` с индексом `data_position` плюс 5 после битового сдвига на 16 позиций влево и ячейки массива `wavheader_real` с индексом `data_position` плюс 6 после битового сдвига на 24 позиции влево. В результате переменная `subChunk2Size` содержит значение длины области звуковых данных.

Переменная `wavheader_length` определяется как значение переменной `data_position` плюс 7 и содержит число байт, расположенных в файле – контейнере до байт звуковых отсчетов.

Позиция указателя перемещается в начало файла.

Затем происходит считывание из файла контейнера байт заголовка в соответствующие переменные структуры `audio`:

```
//1.2 чтение информации о контейнере в переменные структуры audio
soundstream0.read((char*)&AUDIO.chunkID,4); //Содержит символы
    "RIFF" в ASCII кодировке (0x52494646 в big-endian
    представлении). Является началом RIFF-цепочки.
```

```

soundstream0.read((char*)&AUDIO.chunkSize,4); //Это оставшийся
    размер цепочки, начиная с этой позиции. Иначе говоря, это
    размер файла - 8, то есть, исключены поля chunkId и chunkSize.
soundstream0.read((char*)&AUDIO.format,4); //Содержит символы
    "WAVE" (0x57415645 в big-endian представлении)
soundstream0.read((char*)&AUDIO.subChunk1ID,4); //Содержит символы
    "fmt " (0x666d7420 в big-endian представлении)
soundstream0.read((char*)&AUDIO.subChunk1Size,4); //16 для формата
    PCM. Это оставшийся размер подцепочки, начиная с этой позиции.
soundstream0.read((char*)&AUDIO.audioFormat,2); //Аудио формат,
    полный список можно получить здесь. Для PCM = 1 (то есть,
    Линейное квантование). Значения, отличающиеся от 1, обозначают
    некоторый формат сжатия.
soundstream0.read((char*)&AUDIO.numChannels,2); //Количество
    каналов. Моно = 1, Стерео = 2 и т.д.
soundstream0.read((char*)&AUDIO.sampleRate,4); //Частота
    дискретизации. 8000 Гц, 44100 Гц и т.д.
soundstream0.read((char*)&AUDIO.byteRate,4); //Количество байт,
    переданных за секунду воспроизведения.
soundstream0.read((char*)&AUDIO.blockAlign,2); //Количество байт
    для одного сэмпла, включая все каналы.
soundstream0.read((char*)&AUDIO.bitsPerSamle,2); //Количество бит в
    сэмпле. Так называемая "глубина" или точность звучания. 8 бит,
    16 бит и т.д.
soundstream0.read((char*)&AUDIO.subChunk2ID,4); //Содержит символы
    "data" (0x64617461 в big-endian представлении)
soundstream0.read((char*)&trash,4); //Количество байт в области
    данных.

```

В переменные chunkID, chunkSize, format, subChunk1ID, subChunk1Size, sampleRate, byteRate, subChunk2ID считывается по четыре соответствующих байта из заголовка, в переменные audioFormat, numChannels, blockAlign и bitsPerSamle по два. Число считываемых бит определяется структурой заголовка. Последним происходит считывание четырех байт в переменную trash. Это сделано для того, чтобы указатель переместился на 4 позиции право, а значение переменной subChunk2Size, соответствующее этим четырем байтам, было определено ранее.

3.5.3 Функция подсчета необходимой информации по стеганографическому контейнеру

Данная функция предназначена для определения переменных, значения которых будут необходимы для извлечения информации из файла – контейнера:

```
//определение числа сэмплов (V)
samples = ((AUDIO.subChunk2Size)/AUDIO.blockAlign);
//определение числа сабсэмплов (V)
subsamples = AUDIO.subChunk2Size; //samples * AUDIO.numChannels;
старая ф-ла неверная
```

Определяется значение переменной *samples*, означающее число звуковых отсчетов (включая все имеющиеся каналы), содержащиеся в файле – контейнере по формуле:

$$samples = \frac{subChunk2Size}{blockAlign}$$

Определяется значение переменной *subsamples*, равное значению переменной *subChunk2Size* и обозначающее число байт, отводимых под хранение всех звуковых отсчетов, имеющихся в файле – контейнере.

В конце функции внутренний указатель в файле контейнере перемещается на позицию, определяемую значением переменной *wavheader_length* к месту, откуда начинаются байты звуковых отсчетов:

```
//переход каретки к данным
soundstream0.clear();
soundstream0.seekg(wavheader_length);
```

3.5.4 Функция чтения байт данных стеганографического контейнера

Данная функция представлена последовательностью следующих команд:

```
unsigned int *subsamples_array = new unsigned int
[subsamples]; //создание динамического массива для хранения
байт субсэмплов
for (i=0; i<subsamples; i++)
{
    subsamples_array[i]=0; //обнуление
    soundstream0.read((char*)&subsamples_array[i], 1);
```

}

Объявляется массив `subsamples_array` длиной `subsamples` ячеек для хранения байт звуковых отсчетов. В цикле по `i` от 0 до `subsamples` с шагом 1 значение `i` – ой ячейки массива `subsamples_array` обнуляется и в него считывается очередной байт из файла – контейнера из области данных, где расположены звуковые отсчеты. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика `i`.

3.5.5 Функция определения последовательности байт стеганографического контейнера для извлечения

В начале данной функции происходит инициализация необходимых массивов:

```
//ГЕНЕРАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ НОМЕРОВ СУБСЭМПЛОВ, ПОДЛЕЖАЩИХ  
ЗАМЕНЕ  
unsigned long int *subsamples_nums = new unsigned long int  
[subsamples];  
unsigned long int *subsamples_nums2 = new unsigned long int  
[subsamples];  
unsigned int *subsamples_status = new unsigned int  
[subsamples]; //0 – субсемпл свободен, 1 – занят
```

Инициализируется массив `subsamples_nums` длиной `subsamples` для хранения последовательности номеров звуковых отсчетов в которой была произведена замена младших бит для 8 – битного файла – контейнера с помощью первой программы.

Инициализируется массив `subsamples_nums2` длиной `subsamples` для хранения последовательности номеров звуковых отсчетов в которой была произведена замена младших бит для файла – контейнера, битность которого отличается от 8 с помощью первой программы.

Инициализируется массив `subsamples_status` длиной `subsamples` для хранения значений, определяющих статусов номеров звуковых отсчетов в которой была произведена замена младших бит (занят/ не занят) с помощью первой программы. Это сделано для того, чтобы в дальнейшем, при извлечении замена младших бит происходила в разных байтах звуковых отсчетов.

В случае, если число байт звуковых отсчетов больше 1000000 программа выводит информационное сообщение «Подождите...»:

```
if (subsamples>1000000)
    cout<<"Подождите..."<<endl;
```

Далее ячейки массивов `subsamples_status` и `subsamples_nums` обнуляются в цикле по `i` от 0 до `subsamples` с шагом 1:

```
//обнуление subsamples_status, subsamples_nums2
for (i=0;i<subsamples;i++)
{
    subsamples_status[i]=0;
    subsamples_nums2[i]=0;
}
```

После происходит непосредственно генерация последовательности байт для извлечения. Определяется значение номера первого байта для извлечения:

```
//генератор
//начальное заполнение
subsamples_nums[0]=subsamples/2;
subsamples_status[subsamples_nums[0]]++;
```

Нулевая ячейка массива `subsamples_nums` становится равной значению переменной `subsamples` деленному на 2, а значение ячейки `subsamples_status` с индексом равным значению нулевой ячейки массива `subsamples_nums` увеличивается на единицу. Таким образом определен номер первого байта для извлечения, а в соответствующей ячейке массива `subsamples_status` для его установлено значение отличное от нуля. Это означает что данный номер байта для извлечения был только что занят и в дальнейшем следует избегать появления в массиве `subsamples_nums` такого же значения.

Далее происходит заполнение остальных ячеек массивов `subsamples_nums` и `subsamples_status`:

```
//далее заполнение массива
for (i=1;i<subsamples;i++)
{
    subsamples_nums[i]=subsamples_nums[i-1]+(0+rand()%subsamples);
    //если переполнение
    if (subsamples_nums[i]>subsamples)
        subsamples_nums[i]=subsamples_nums[i]%subsamples;
    again:
    if (subsamples_status[subsamples_nums[i]]!=0)
```

```

{
    subsamples_nums[i]=subsamples_nums[i]+(0+rand()%subsamples);
    if (subsamples_nums[i]>subsamples)
        subsamples_nums[i]=subsamples_nums[i]%subsamples;
    subsamples_nums[i]--;
    goto again;
}
subsamples_status[subsamples_nums[i]]++;
}

```

В цикле по i от 0 до `subsamples` с шагом 1 значение i – ой ячейки массива `subsamples_nums` определяется как значение предыдущей ячейки этого массива плюс случайное число, сгенерированное программой. Если получившееся значения оказалось больше значения `subsamples` (общего количества байт звуковых отсчетов) оно становится равным остатку от деления на значение `subsamples`. Проверяется значение соответствующей ячейки массива `subsamples_status` на равенство нулю. Если оно не равно нулю, необходимо сгенерировать другое число, так как данное уже имеется в массиве `subsamples_nums`. Значение i – ой ячейки массива `subsamples_nums` определяется как значение уже текущей ячейки (только что сгенерированное) плюс случайное число, сгенерированное программой. Если получившееся значения оказалось больше значения `subsamples` (общего количества байт звуковых отсчетов) оно становится равным остатку от деления на значение `subsamples` и уменьшается на единицу, после чего программа переходит на метку `again`. Если сгенерированное число не встречалось ранее, оно занесется в i – ю ячейку массива `subsamples_nums`, а значение ячейки массива `subsamples_status` с индексом, определяемым содержимым i – ой ячейки массива `subsamples_nums` увеличивается на единицу. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

После заполнения массива `subsamples_nums` заполняется массив `subsamples_nums2`:

```

//заполнение subsamples_nums2
j=0;
for (i=0;i<subsamples;i++)
{

```

```

if ((subsamples_nums[i])%(AUDIO.blockAlign/AUDIO.numChannels)==0)
{
    subsamples_nums2[j]=subsamples_nums[i];
    j++;
}
}

```

Переменная *j* равна перед выполнением цикла нулю. В цикле по *i* от 0 до *subsamples* с шагом 1 в случае если остаток от деления содержимого *i* – ой ячейки массива *subsamples_nums* на частное переменных *blockAlign* и *numChannels* равно нулю значение *j* – ой ячейки массива *subsamples_nums2* приравнивается к значению *i* – ой ячейки массива *subsamples_nums*, а переменная *j* увеличивается на единицу. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика *i*. Таким образом, массив *subsamples_nums2* содержит только те значения байт из массива *subsamples_nums*, которые являются младшими для файла – контейнера, битность которого отличается от 8.

3.5.6 Функция извлечения зашифрованного текста

Объявляется массив *pole1* длиной 32 ячейки для хранения бит длины скрытого зашифрованного сообщения и заполняется:

```

//5.ПОЛУЧЕНИЕ МАССИВА Ш.Т
unsigned int pole1[32];
//заполнение массива pole1
for (i=0;i<32;i++)
{
    if (AUDIO.blockAlign==1)//для 8 бит
        pole1[i]=subsamples_array[subsamples_nums[i]]&0x01;
    else//для 16 бит
        pole1[i]=subsamples_array[subsamples_nums2[i]]&0x01;
}

```

В цикле по *i* от 0 до 32 с шагом 1 в случае если переменная *blockAlign* равна 1 (файл – контейнер 8 - битный) значение *i* – ой ячейки массива *pole1* становится равным значению ячейки массива *subsamples_array* с индексом, определяемым значением *i* – ой ячейки массива *subsamples_nums* и логически умноженному на маску 0x01. В противном случае значение *i* – ой ячейки массива *pole1* становится равным значению ячейки массива *subsamples_array* с

индексом, определяемым значением i – ой ячейки массива `subsamples_nums2` и логически умноженному на маску `0x01`. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i .

Далее на основании содержимого ячеек массива `pole1` получается значения длины скрытого зашифрованного сообщения:

```
//ПОЛУЧЕНИЕ full_length из pole1;
full_length=((pole1[31])&0x1)|((pole1[30]<<1)&0x2)
|((pole1[29]<<2)&0x4)|((pole1[28]<<3)&0x8)
|((pole1[27]<<4)&0x10)|((pole1[26]<<5)&0x20)
|((pole1[25]<<6)&0x40)|((pole1[24]<<7)&0x80)
|((pole1[23]<<8)&0x100)|((pole1[22]<<9)&0x200)
|((pole1[21]<<10)&0x400)|((pole1[20]<<11)&0x800)
|((pole1[19]<<12)&0x1000)|((pole1[18]<<13)&0x2000)
|((pole1[17]<<14)&0x4000)|((pole1[16]<<15)&0x8000)
|((pole1[15]<<16)&0x10000)|((pole1[14]<<17)&0x20000)
|((pole1[13]<<18)&0x40000)|((pole1[12]<<19)&0x80000)
|((pole1[11]<<20)&0x100000)|((pole1[10]<<21)&0x200000)
|((pole1[9]<<22)&0x400000)|((pole1[8]<<23)&0x800000)
|((pole1[7]<<24)&0x1000000)|((pole1[6]<<25)&0x2000000)
|((pole1[5]<<26)&0x4000000)|((pole1[4]<<27)&0x8000000)
|((pole1[3]<<28)&0x10000000)|((pole1[2]<<29)&0x20000000)
|((pole1[1]<<30)&0x40000000)|((pole1[0]<<31)&0x80000000);
```

Значение переменной `full_length` определяется как логическая сумма значений ячеек массива `pole1`, битово сдвинутых на соответствующее число позиций влево и умноженных на соответствующую маску. Теперь переменная `full_length` содержит длину (число байт) скрытого зашифрованного сообщения.

Далее объявляется массив `array_to_hide` длиной `full_length*8` минус 32 ячеек и заполняется:

```
unsigned int *array_to_hide = new unsigned int [full_length*8-
32]; //динамический массив под хранение бит символов ш.т +
последний бит служебной информации
//ПОЛУЧЕНИЕ массива pole2+pole3
for (i=0;i<full_length*8-32;i++)
{
    if (AUDIO.blockAlign==1)//для 8 бит
        array_to_hide[i]=subsamples_array[subsamples_nums[i+32]]&0x1;
    else //if (AUDIO.blockAlign==2)//для 16 бит
        array_to_hide[i]=subsamples_array[subsamples_nums2[i+32]]&0x1;
}
```

В цикле по i от 0 до 32 с шагом 1 в случае если переменная `blockAlign` равна 1 (файл – контейнер 8 - битный) значение i – ой ячейки массива `array_to_hide` становится равным значению ячейке массива `subsamples_array` с индексом, определяемым значением ячейки массива `subsamples_nums` с индексом (i плюс 32) и логически умноженным на маску 0x1. В противном случае значение i – ой ячейки массива `array_to_hide` становится равным значению ячейке массива `subsamples_array` с индексом, определяемым значением ячейки массива `subsamples_nums2` с индексом (i плюс 32) и логически умноженным на маску 0x1. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . Таким образом, массив `array_to_hide` будет содержать в своих ячейках биты символов зашифрованного сообщения и 8 бит , отводимых под хранение числа дополняющих символов, которое определяется далее:

```
//получение dop_length из pole3
dop_length=(array_to_hide[full_length*8-33]&0x1)
| ((array_to_hide[full_length*8-34]<<1)&0x2)
| ((array_to_hide[full_length*8-35]<<2)&0x4)
| ((array_to_hide[full_length*8-36]<<3)&0x8)
| ((array_to_hide[full_length*8-37]<<4)&0x10)
| ((array_to_hide[full_length*8-38]<<5)&0x20)
| ((array_to_hide[full_length*8-39]<<6)&0x40)
| ((array_to_hide[full_length*8-40]<<7)&0x80);
```

Переменная `dop_length` вычисляется как значение логической суммы ячейки массива `array_to_hide` с индексом (`full_length`, умноженное на 8 и минус 33) умноженной на маску 0x1, ячейки массива `array_to_hide` с индексом (`full_length`, умноженное на 8 и минус 34) битово сдвинутой на 1 позицию влево и умноженной на маску 0x2, ячейки массива `array_to_hide` с индексом (`full_length`, умноженное на 8 и минус 35) битово сдвинутой на 2 позицию влево и умноженной на маску 0x4, ячейки массива `array_to_hide` с индексом (`full_length`, умноженное на 8 и минус 36) битово сдвинутой на 3 позицию влево и умноженной на маску 0x8, ячейки массива `array_to_hide` с индексом (`full_length`, умноженное на 8 и минус 37) битово сдвинутой на 4 позицию влево и умноженной на маску 0x10, ячейки массива `array_to_hide` с индексом

(full_length, умноженное на 8 и минус 38) битово сдвинутой на 5 позицию влево и умноженной на маску 0x20, ячейки массива array_to_hide с индексом (full_length, умноженное на 8 и минус 39) битово сдвинутой на 6 позицию влево и умноженной на маску 0x40, ячейки массива array_to_hide с индексом (full_length, умноженное на 8 и минус 40) битово сдвинутой на 7 позицию влево и умноженной на маску 0x80. Таким образом, переменная dop_length будет содержать значение числа дополняющих символов.

3.5.7 Функция открытия и чтения файла ключа

Данная функция является полностью идентичной одноименной функции, содержащейся в первой программе и представлена последовательностью следующих команд:

```
//открытие файла ключа
FILE * ptrFile = fopen("key.txt", "rb");
if (ptrFile == NULL)
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    {
        system ("pause");
        return -1;
    }
}
else
{
    fseek(ptrFile, 0, SEEK_END); // переместить внутренний
    указатель в конец файла
    key_length = ftell(ptrFile); // вернуть текущее положение
    внутреннего указателя
    fclose (ptrFile); // закрыть файл
    if (key_length!=32)
    {
        cout<<"ОШИБКА! Файл ключа key.txt не равен 256 битам."<<endl;
        system ("pause");
        return -1;
    }
}
unsigned int *key_array = new unsigned int
    [key_length]; //динамический массив под хранение кодов символов
    ключа
ifstream keystream("key.txt", ios::binary);
if (!(keystream.is_open()))
```

```

{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    system ("pause");
    return -1;
}
//чтение ключа
for (i=0;i<key_length;i++)
{
    key_array[i]=0;
    keystoream.read((char*)&key_array[i],1);
}
//закрытие
keystoream.close();

```

Сначала происходит открытие файла ключа key.txt в режиме бинарного чтения для проверки на существование и для определения его длины. Если файл не существует, или открытия не произошло по каким – либо другим причинам, программа выводит на экран сообщение об ошибке «ОШИБКА! Файл ключа key.txt не был открыт.» и завершает свою работу.

Если файл был открыт, внутренний указатель в открытом файле перемещается в его конец. Значение позиции указателя заносится в переменную key_length, которое означает число байт, содержащихся в файле ключа. Файл закрывается. Происходит проверка переменной key_length и если ее значение отличается от 32 (если ключ содержит не 256 бит) программа выводит на экран сообщение об ошибке «ОШИБКА! Файл ключа key.txt не равен 256 битам.» и завершает свою работу. Если значение key_length равно 32, функция продолжает свою работу. Объявляется динамический массив key_array длиной key_length ячеек. Происходит открытие файла ключа для чтения. Чтение ключа реализовано в цикле по i от 0 до key_length с шагом 1. Значение i – ой ячейки массива key_array обнуляется и в него записывается байт, считанный из файла ключа. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i. Файл закрывается.

В результате выполнения функции формируется массив key_array, в ячейках которого содержатся байты ключа.

3.5.8 Функция формирования КЗУ

Данная функция представлена последовательностью следующих команд:

```
//Заполнение КЗУ
x0=(key_array[0]<<24)+(key_array[1]<<16)+(key_array[2]<<8)+key_array[3];
x1=(key_array[4]<<24)+(key_array[5]<<16)+(key_array[6]<<8)+key_array[7];
x2=(key_array[8]<<24)+(key_array[9]<<16)+(key_array[10]<<8)+key_array[11];
x3=(key_array[12]<<24)+(key_array[13]<<16)+(key_array[14]<<8)+key_array[15];
x4=(key_array[16]<<24)+(key_array[17]<<16)+(key_array[18]<<8)+key_array[19];
x5=(key_array[20]<<24)+(key_array[21]<<16)+(key_array[22]<<8)+key_array[23];
x6=(key_array[24]<<24)+(key_array[25]<<16)+(key_array[26]<<8)+key_array[27];
x7=(key_array[28]<<24)+(key_array[29]<<16)+(key_array[30]<<8)+key_array[31];
```

КЗУ представляет собой восемь 32 – разрядных накопителей. Значение первого накопителя x_0 образуется из первых четырех байт ключа, расположенных в ячейках массива `key_array` следующим образом значение нулевой ячейки массива `key_array` побитово сдвигается на 24 позиции влево, значение первой ячейки массива сдвигается на 16 позиций влево, значение второй ячейки сдвигается на 8 позиций влево. Получившиеся значения логически суммируются между собой и со значением третьей ячейки массива `key_array` и записываются в накопитель x_0 . Следующие накопители суммируются аналогичным образом, за исключением того, что используются соответствующие накопителю четыре ячейки массива `key_array`. Так, восемь накопителей $x_0, \dots, x_7$ содержат в себе по четыре байта (32 бита) ключа.

Затем значения накопителей КЗУ выставляются в необходимой последовательности подачи для зашифрования:

```
//Заполнение Массива x_array последовательности подачи КЗУ
//ячейки 8-31
x_array[24]=x_array[8]=x_array[16]=x7;
x_array[25]=x_array[9]=x_array[17]=x6;
x_array[26]=x_array[10]=x_array[18]=x5;
```

```

x_array[27]=x_array[11]=x_array[19]=x4;
x_array[28]=x_array[12]=x_array[20]=x3;
x_array[29]=x_array[13]=x_array[21]=x2;
x_array[30]=x_array[14]=x_array[22]=x1;
x_array[31]=x_array[15]=x_array[23]=x0;
//ячейки 0-7
x_array[0]=x0;
x_array[1]=x1;
x_array[2]=x2;
x_array[3]=x3;
x_array[4]=x4;
x_array[5]=x5;
x_array[6]=x6;
x_array[7]=x7;

```

Массив `x_array` содержит 32 ячейки, по числу предстоящих циклов расшифрования. Алгоритм ГОСТ 28147-89[9] предполагает, что в каждом из 32 циклов расшифрования будет использован определенный накопитель КЗУ. Необходимая последовательность значений накопителей КЗУ, используемых при расшифровании формируется в массиве `x_array` и отличается от той, что применяется при зашифровании. Ячейки данного массива с индексами 24, 8, 16 приравниваются к значению накопителя `x7`. Ячейки с индексами 25, 9, 17 приравниваются к значению накопителя `x6`, ячейки с индексами 26, 10, 18 приравниваются к значению накопителя `x5` и т.д. Первые восемь ячеек заполняются в прямом порядке: 0 – ая принимает значение накопителя `x0`, 1 – ая принимает значение накопителя `x1`, ..., 7 – ая принимает значение накопителя `x7`.

3.5.9 Функция подготовки зашифрованного текста к расшифрованию

В начале функции определяется длина зашифрованного текста без учета четырех байт означающих длину скрытой информации и байта числа дополняющих символов. Определяется число блоков зашифрованного текста:

```

//2.1.определения длины (V)
crypt_length=full_length-5;//от full_length отнять 4 первых байта
и последний
//2.3 определение числа блоков (по 64 бита) (V)
blocks=(crypt_length)/8;
Создаются массивы crypt_array и text_array длиной crypt_length
ячеек:

```

```
//2.4 создание массивов (V)
unsigned int *crypt_array = new unsigned int
    [crypt_length]; //динамический массив под хранение кодов
    символов ш.т+дополнения
unsigned int *text_array = new unsigned int
    [crypt_length]; //динамический массив под хранение кодов
    символов о.т+дополнения
```

Первый массив нужен для хранения байт символов текста с учетом дополняющих символов до расшифрования. Второй нужен для хранения байт символов текста с учетом дополняющих символов после расшифрования.

Заполнение массива `crypt_array` происходит следующим образом:

```
//2.5 заполнение массивов (V)
//crypt_array должен содержать коды/байты символов а не биты!
j=0;
for (i=0;i<crypt_length;i++)
{
    crypt_array[i]=((array_to_hide[j+0]<<7)&0x80)+
        ((array_to_hide[j+1]<<6)&0x40)+
        ((array_to_hide[j+2]<<5)&0x20)+
        ((array_to_hide[j+3]<<4)&0x10)+
        ((array_to_hide[j+4]<<3)&0x8)+
        ((array_to_hide[j+5]<<2)&0x4)+
        ((array_to_hide[j+6]<<1)&0x2)+
        ((array_to_hide[j+7])&0x1);
    j=j+8;
}
```

Перед циклом значение `j` обнуляется. В цикле по `i` от 0 до `crypt_length` с шагом 1 значение `i` – ой ячейки массива `crypt_array` становится равным сумме значений ячеек массива `array_to_hide` с индексом `j` плюс 0, битово сдвинутым на 7 позиций влево и умноженным на маску `0x80`, с индексом `j` плюс 1, битово сдвинутым на 6 позиций влево и умноженным на маску `0x40`, с индексом `j` плюс 2, битово сдвинутым на 5 позиций влево и умноженным на маску `0x20`, с индексом `j` плюс 3, битово сдвинутым на 4 позиции влево и умноженным на маску `0x10`, с индексом `j` плюс 4, битово сдвинутым на 3 позиции влево и умноженным на маску `0x8`, с индексом `j` плюс 5, битово сдвинутым на 2 позиции влево и умноженным на маску `0x4`, с индексом `j` плюс 6, битово сдвинутым на 1 позицию влево и умноженным на маску `0x2`, с индексом `j` плюс

6, умноженным на маску 0x1. Такт цикла завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . После выполнения цикла массив `crypt_array` содержит байты зашифрованного сообщения с учетом дополняющих символов.

3.5.10 Функция расшифрования зашифрованного текста

Расшифрование происходит для каждого блока в цикле по `block` от 0 до `blocks` с шагом 1. Определяется заполнение переменных N_1 и N_2 соответствующих накопителям N_1 и N_2 :

```
//3.РАСШИФРОВАНИЕ
//Для каждого блока..
for (block=0;block<blocks;block++)//начало поблочного цикла
{
//определение начального заполнения N1 и N2 (перед первым циклом
зашифрования)
N1=(crypt_array[0+8*block]<<24)+(crypt_array[1+8*block]<<16)+(crypt_
t_array[2+8*block]<<8)+crypt_array[3+8*block]; //(V)
N2=(crypt_array[4+8*block]<<24)+(crypt_array[5+8*block]<<16)+(crypt_
t_array[6+8*block]<<8)+crypt_array[7+8*block]; //(V)
```

N_1 определяется как сумма значений ячеек массива `crypt_array` с индексами (0 плюс 8, умноженное на `block`) после битового сдвига на 24 влево, (1 плюс 8, умноженное на `block`) после битового сдвига на 16 влево, (2 плюс 8, умноженное на `block`) после битового сдвига на 8 влево, (3 плюс 8, умноженное на `block`). N_2 определяется как сумма значений ячеек массива `crypt_array` с индексами (4 плюс 8, умноженное на `block`) после битового сдвига на 24 влево, (5 плюс 8, умноженное на `block`) после битового сдвига на 16 влево, (6 плюс 8, умноженное на `block`) после битового сдвига на 8 влево, (7 плюс 8, умноженное на `block`). Таким образом переменные N_1 и N_2 содержат первые и последние 32 бита i – го блока зашифрованного текста из массива `crypt_array` соответственно.

После того как значения переменных N_1 и N_2 сформированы идет расшифрование, которое реализовано в виде цикла по `round` от 0 до 32 с шагом 1:

```
for (round=0;round<32;round++)
{
//подсчет CM1 - сложение по модулю 2^32
```

```
CM1=N1+x_array[round]; //(V)
```

В такте цикла происходит подсчет переменной CM1 путем сложения по модулю 2^{32} значения N1 с содержимым ячейки массива x_array имеющей индекс round (Содержимое 32 ячеек x_array было определено ранее в функции формирования КЗУ). Получившееся значение переменной CM1 соответствует результату сумматора CM₁.

Далее значение CM1 должно пройти процедуру подстановки. Для этого по ячейкам массива CM1_array распределяются четверки бит переменной CM1:

```
//заполнение CM1_array для подстановки(V)
CM1_array[0]=(CM1&0xf0000000)>>28;
CM1_array[1]=(CM1&0xf000000)>>24;
CM1_array[2]=(CM1&0xf00000)>>20;
CM1_array[3]=(CM1&0xf0000)>>16;
CM1_array[4]=(CM1&0xf000)>>12;
CM1_array[5]=(CM1&0xf00)>>8;
CM1_array[6]=(CM1&0xf0)>>4;
CM1_array[7]=CM1&0xf;
```

Нулевая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0xf0000000 побитово сдвинутое на 28 позиций вправо.

Первая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0f000000 побитово сдвинутое на 24 позиций вправо.

Вторая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x00f00000 побитово сдвинутое на 20 позиций вправо.

Третья ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x000f0000 побитово сдвинутое на 16 позиций вправо.

Четвертая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0000f000 побитово сдвинутое на 12 позиций вправо.

Пятая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x00000f00 побитово сдвинутое на 8 позиций вправо.

Шестая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x000000f0 побитово сдвинутое на 4 позиций вправо.

Седьмая ячейка CM1_array принимает значение результата логического умножения значения CM1 на маску 0x0000000f.

После заполнения ячеек массива CM1_array происходит подстановка:

```
//произведение подстановки(V)
for (i=0;i<8;i++)//для каждой четверки бит CM1
{
    CM1_array[i]=tab[7-i][CM1_array[i]];//подстановка
}
```

В цикле по i от 0 до 8 с шагом 1 значение i – ой ячейки CM1_array становится равным начению ячейки двумерного массива tab (таблицы замены) с первым индексом 7 минус i и вторым индексом, определяемым содержимым i – ой ячейки массива CM1_array. Такт цикла по i завершается. Цикл повторяется или завершается в зависимости от значения счетчика i. Заполнение массива таблицы замены в программе зашифрования и скрытия и программе извлечения и расшифрования одинаковое.

Затем получившиеся значения в ячейках массива CM1_array собираются воедино в переменную K:

```
//сборка массива CM1_array после подстановки в переменную K
K=(CM1_array[0]<<28)+(CM1_array[1]<<24)+(CM1_array[2]<<20)+(CM1_ar
    ray[3]<<16)+(CM1_array[4]<<12)+(CM1_array[5]<<8)+(CM1_array[6]
        <<4)+(CM1_array[7]);
```

Значение K получается путем суммирования нулевой ячейки CM1_array, сдвинутой побитово на 28 влево, первой ячейки CM1_array, сдвинутой побитово на 24 влево, второй ячейки CM1_array, сдвинутой побитово на 20 влево, третьей ячейки CM1_array, сдвинутой побитово на 16 влево, четвертой ячейки CM1_array, сдвинутой побитово на 12 влево, пятой ячейки CM1_array,

сдвинутой побитово на 8 влево, шестой ячейки CM1_array, сдвинутой побитово на 4 влево и седьмой ячейки CM1_array.

Так переменная K содержит значение CM1 после подстановки. Данное значение циклически сдвигается на 11 позиций в сторону старших разрядов в регистре R:

```
//циклический сдвиг на 11 влево
r=(K&0xffe00000)>>21;//старшая часть помещается в r и сдвигается в
    конец младший
R=(K<<11)+r;//сдвиг числа на 11 влево + r
```

Для реализации циклического сдвига старшие 11 бит переменной K умножаются на маску 0xffe00000 и побитово сдвигаются на 21 позицию вправо, результат записывается в переменную r. Значение переменной R определяется как результат битового сдвига значения переменной K на 11 позиций влево сложенный с переменной r. Циклический сдвиг произведен.

Далее определяется значение переменной CM2 соответствующей результату сумматора CM₂:

```
CM2=R^N2; //сложение по модулю 2 (XOR)
```

Значение переменных R и N2 складываются по модулю 2 и получившиеся значения заносятся в переменную CM2.

После того как значение CM2 получено происходит проверка на то, является ли такт цикла по round последним (32 по счету):

```
// N1 -> N2, CM2 -> N1 (V)
if (round!=31)//для всех циклов кроме последнего 32-го
{
    N2=N1;
    N1=CM2;
}
else //для последнего цикла
{
    N2=CM2;
}
} //конец цикла 32-3
```

Если счетчик цикла round не равен 31 (такт цикла не последний) переменная N2 приравнивается к N1, а тот, в свою очередь становится равен

CM2. В противном случае такт цикла будет последним и значение CM2 заносится в переменную N2, а значение переменной N1 останется неизменным.

На этом такт цикла по round завершается. Цикл завершается или повторяется в зависимости от значения счетчика round.

После того, как 32 цикла расшифрования произведены на выходе из этих циклов переменные N1 и N2 содержат биты расшифрованного текста блока. Для того чтобы сформировать блок расшифрованного текста была написана следующая последовательность команд:

```
//запись N1,N2 в массив crypt_array
text_array[0+block*8]=(N1&0xff000000)>>24;
text_array[1+block*8]=(N1&0x00ff0000)>>16;
text_array[2+block*8]=(N1&0x0000ff00)>>8;
text_array[3+block*8]=(N1&0x000000ff);
text_array[4+block*8]=(N2&0xff000000)>>24;
text_array[5+block*8]=(N2&0x00ff0000)>>16;
    text_array[6+block*8]=(N2&0x0000ff00)>>8;
    text_array[7+block*8]=(N2&0x000000ff);
} //конец поблочного цикла
```

Значение ячейки массива text_array с индексом (0 плюс block, умноженный на 8) становится равным результату логического умножения значения переменной N1 на маску 0xff000000 с последующим битовым сдвигом на 24 позиции вправо.

Значение ячейки массива text_array с индексом (1 плюс block, умноженный на 8) становится равным результату логического умножения значения переменной N1 на маску 0x00ff0000 с последующим битовым сдвигом на 16 позиции вправо.

Значение ячейки массива text_array с индексом (2 плюс block, умноженный на 8) становится равным результату логического умножения значения переменной N1 на маску 0x0000ff00 с последующим битовым сдвигом на 8 позиции вправо.

Значение ячейки массива text_array с индексом (3 плюс block, умноженный на 8) становится равным результату логического умножения значения переменной N1 на маску 0x000000ff.

Значение ячейки массива `text_array` с индексом (4 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0xff000000` с последующим битовым сдвигом на 24 позиции вправо.

Значение ячейки массива `text_array` с индексом (5 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x00ff0000` с последующим битовым сдвигом на 16 позиции вправо.

Значение ячейки массива `text_array` с индексом (6 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x0000ff00` с последующим битовым сдвигом на 8 позиции вправо.

Значение ячейки массива `text_array` с индексом (7 плюс `block`, умноженный на 8) становится равным результату логического умножения значения переменной `N2` на маску `0x000000ff`.

Таким образом в восьми ячейках массива `text_array` формируются байты, принадлежащих блоку зашифрованного текста.

Такт цикла по `block` завершается. Цикл повторяется или завершается в зависимости от значения счетчика `block`.

В конце цикла по `block` массив `text_array` содержит в каждом своих восьми ячейках байты блока расшифрованного текста. Данная функция в целом схожа с функцией зашифрования, использованной в первой программе.

3.5.11 Функция создания выходного файла и записи в него расшифрованного текста

Данная функция представлена последовательностью команд:

```
//ЗАПИСЬ В ФАЙЛ
//создание файла o.t
ofstream textstream("text.txt", ios_base::out | ios::binary
    | ios_base::trunc); //создание выходного файла
for (i=0; i<crypt_length-dop_length; i++)
{
    char symbol= text_array[i];
```

```
    textstream.write((char*)&symbol,1);  
}  
textstream.close();
```

Создается выходной текстовый файл text.txt. В цикле по i от 0 до `crypt_length` минус `dop_length` с шагом 1 переменная `symbol` приравнивается к значению i – ой ячейки массива `text_array` и записывается в выходной текстовый файл. Такт цикла по i завершается. Цикл повторяется или завершается в зависимости от значения счетчика i . Файл закрывается. Таким образом, текстовый файл будет содержать текст извлеченного и расшифрованного сообщения без дополняющих символов, то есть в оригинале, полностью идентичный тому, что поступал на вход первой программы.

4. Экспериментальная проверка работоспособности программ

4.1 Проверка работоспособности программы зашифрования и скрывает сообщения

В каталоге с исполняемым файлом программы Hide_message.exe были созданы текстовый файл ключа key.txt, файл сообщения text.txt и помещен файл – контейнер, в который необходимо скрыть зашифрованное сообщение, показано на рисунке 37.



Рисунок 37 – Размещение программы Hide_message.exe, файла сообщения, файла ключа и файла – контейнера в одном каталоге

Файл – контейнер представляет собой 16 – битный, одноканальный (моно) файл формата wav, звуковые отсчеты в котором кодированы методом импульсно – кодовой модуляции (PCM). Он был предварительно переименован в sound.wav, так как это необходимое условие для работы программы. В файл ключа key.txt была введена последовательность из 32 символов (256 бит). В файл сообщения text.txt была введена последовательность символов, показанная на рисунке 38.

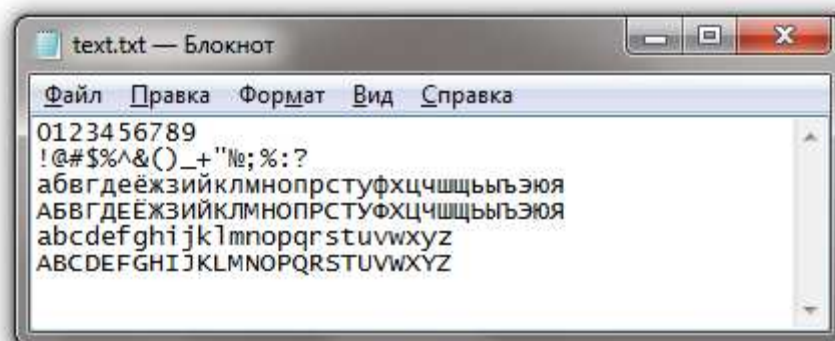


Рисунок 38 – Содержимое файла сообщения text.txt

Программа Hide_message.exe была запущена и в появившемся окне было выведено информирующее пользователя сообщение: «Сообщение зашифровано и скрыто!». Это значит, что текст сообщения был успешно зашифрован и скрыт в файле – контейнере s_sound.wav, который был создан по окончании работы программы, показано на рисунке 39.

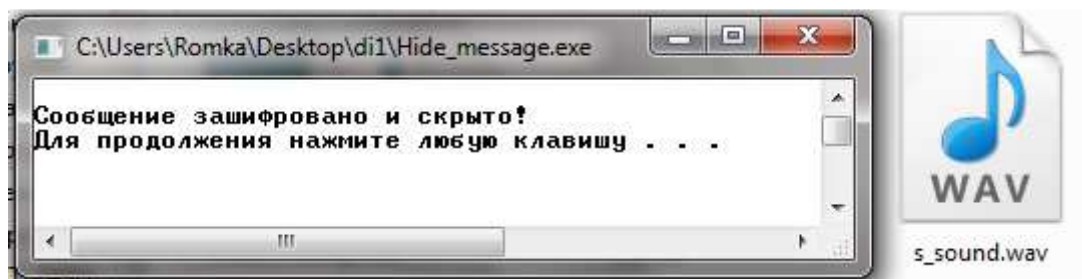


Рисунок 39 – Окно программы Hide_message.exe и выходной файл – контейнер s_sound.wav

Ознакомившись с файлами sound.wav и s_sound.wav можно прийти к выводу, что они абсолютно идентичны по звучанию, имеют одинаковый размер, число каналов, битность, частоту дискретизации и другие параметры. Сравнивая содержимое структуры двух файлов можно заметить, что младшие биты в некоторых байтах файлов отличаются друг от друга в области расположения байт звуковых отсчетов. Порядок последовательности таких байт определяется соответствующим алгоритмом, описанным ранее. Таким образом, на выходе программы Hide_message.exe получился файл s_sound.wav, содержащий в себе зашифрованный и скрытый текст сообщения.

4.2 Проверка работоспособности программы извлечения и расшифрования сообщения

В каталог с исполнительным файлом программы Seek_message.exe были созданы текстовый файл ключа key.txt и помещен файл – контейнер s_sound.wav, полученный при помощи первой программы показано на рисунке 40.



Рисунок 40 - Размещение программы Seek_message.exe, файла ключа и файла – контейнера в одном каталоге

Содержимое файла ключа (ключ) key.txt в каталогах первой и второй программ одинаковое. Программа Seek_message.exe была запущена и в появившемся окне было выведено информирующее пользователя сообщение: «Сообщение извлечено и расшифровано.», а в каталоге появился файл text.txt, показано на рисунке 41. Это значит, что текст сообщения был успешно извлечен из файла – контейнера s_sound.wav и расшифрован.

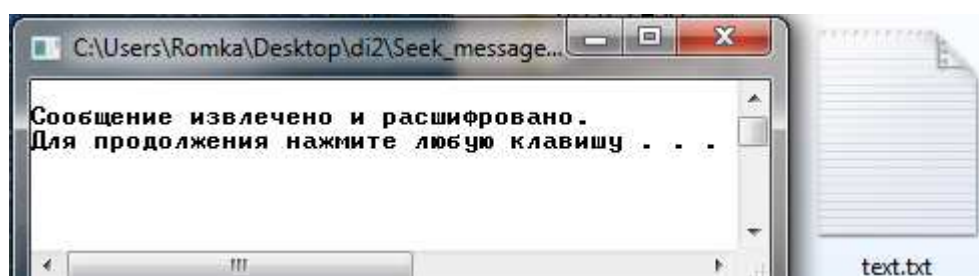


Рисунок 41 - Окно программы Seek_message.exe и выходной файл сообщения text.txt

Получившийся в результате файл text.txt был открыт, показано на рисунке 42.

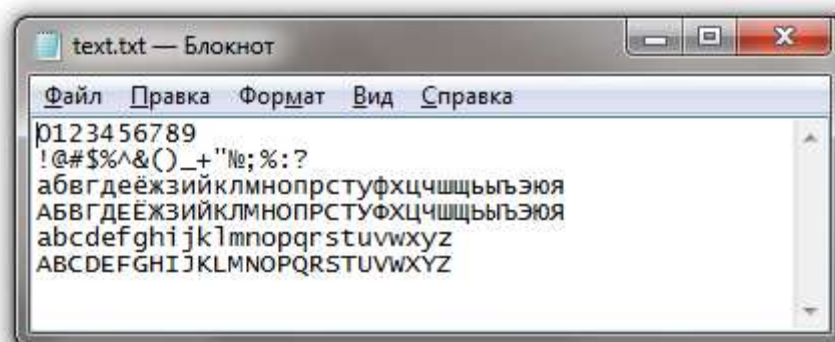


Рисунок 42 – Содержимое файла сообщения text.txt после расшифрования и извлечения

Текст сообщения, содержащийся в данном файле полностью совпадает с тем, что был введен в одноименный файл в первой программе. Это позволяет сделать вывод о том, что программы Hide_message.exe и Seek_message.exe функционируют верно, в соответствии с требованиями.

Подобный эксперимент с файлами – контейнерами, отличными по битности и количеству каналов показали аналогичный результат.

Также была смоделирована ситуация, когда ключи, подаваемые на вход программ отличаются. Для этого в каталоге, где располагается программа Seek_message.exe в файле ключа key.txt был изменен всего один символ. Программа Seek_message.exe была вновь запущена и в результате был создан файл text.txt, содержимое которого показано на рисунке 43.

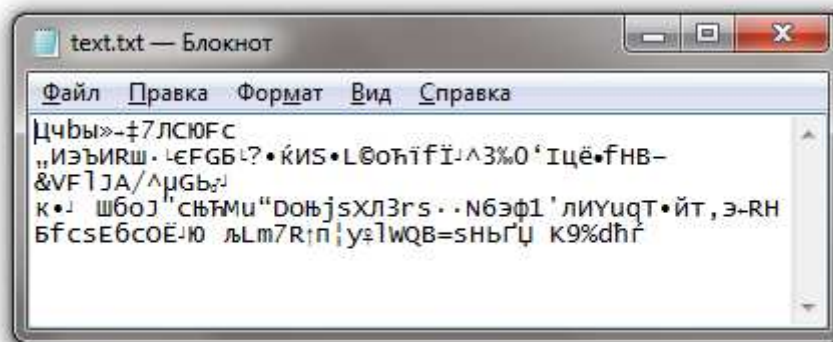


Рисунок 43 – Содержимое файла сообщения text.txt после расшифрования и извлечения с использованием неверного ключа

Как видно, получившийся текст полностью отличается от того, что подавался на вход первой программы с оригинальным ключом.

4.3 Реакция на ошибки

Представленными программами предусмотрены реакции на различного рода ошибки и невыполнение их требований. Это реакции на отсутствие требуемых файлов в каталоге, несоответствие их содержимого или содержимого файла – контейнера предъявляемым требованиям, или невозможность скрытия текста сообщения (для первой программы).

Для проверки реакции на отсутствие файла ключа, программа Hide_message.exe была запущена, а файл key.txt в ее каталоге был удален.

Реакция в программах Hide_message.exe и Seek_message.exe на данную ошибку одинаковая и представлена на рисунке 44. Программы выводят на экран сообщение «ОШИБКА! Файл ключа key.txt не был открыт» и завершают свою работу.



Рисунок 44 – Реакция программ Hide_message.exe и Seek_message.exe на отсутствие файла ключа key.txt

Файл ключа, может быть по ошибке заполнен не полностью или могут быть введены лишние символы. Для этого в файле ключа key.txt был намеренно удален символ. Реакция программ на данную ошибку предполагает вывод на экран сообщения «ОШИБКА! Файл ключа key.txt равен 256 битам» с завершением работы и показана на рисунке 45.

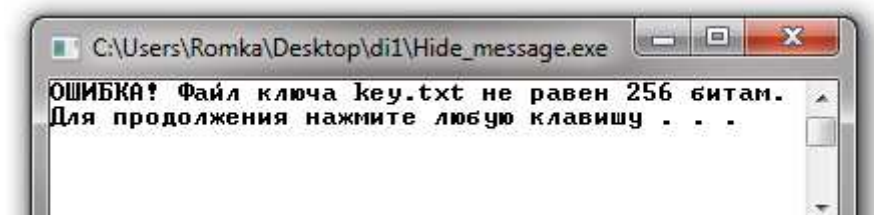


Рисунок 45 - Реакция программ Hide_message.exe и Seek_message.exe на несоответствие длины ключа в файле key.txt

Реакция программы Hide_message.exe на отсутствие файла сообщения text.txt в каталоге предполагает вывод на экран сообщения «ОШИБКА! Файл открытого текста text.txt не был открыт» с завершением работы и показана на рисунке 46.

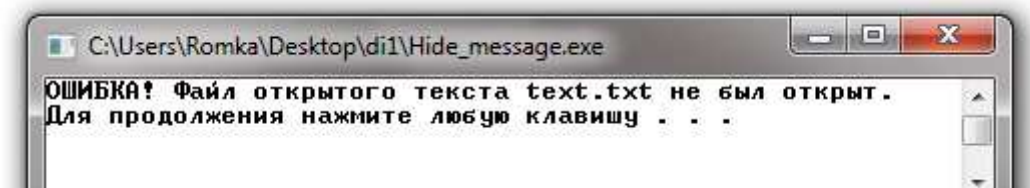


Рисунок 46 - Реакция программы Hide_message.exe на отсутствие в каталоге файла сообщения text.txt

Текст сообщения для зашифрования и скрытия должен содержать не менее четырех символов. Для проверки реакции программы Hide_message.exe на несоответствие приведенному требованию текста сообщения, был создан файл text.txt, в который было введено три символа. В результате программа вывела на экран сообщение «ОШИБКА! Файл открытого текста text.txt должен содержать не менее 4 символов.» и завершила свою работу, показано на рисунке 47.

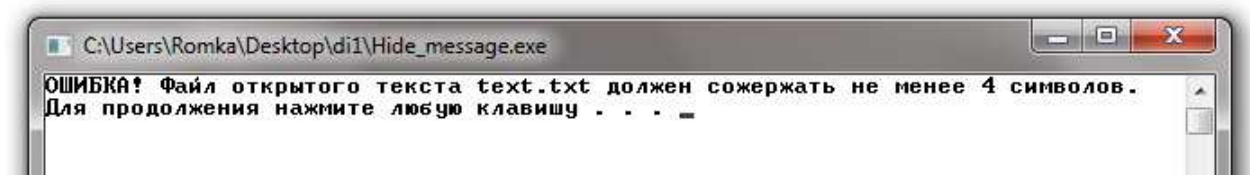


Рисунок 47 - Реакция программы Hide_message.exe несоответствие длины текста в файле сообщения text.txt

Была смоделирована ситуация, когда в каталог с программой не был помещен файл – контейнер, нужный для скрытия/извлечения в программах Hide_message.exe и Seek_message.exe соответственно. Для этого в соответствующих каталогах не были помещены файлы контейнеры sound.wav/s_sound.wav. В результате программа Hide_message.exe при запуске вывела на экран сообщение «ОШИБКА! Файл sound.wav не был открыт.» и завершила свою работу, показано на рисунке 48.



Рисунок 48 – Реакция программы Hide_message.exe на отсутствия в каталоге файла – контейнера sound.wav

Программа Seek_message.exe при запуске вывела на экран сообщение «ОШИБКА! Файл s_sound.wav не был открыт.» и завершила свою работу, показано на рисунке 49.

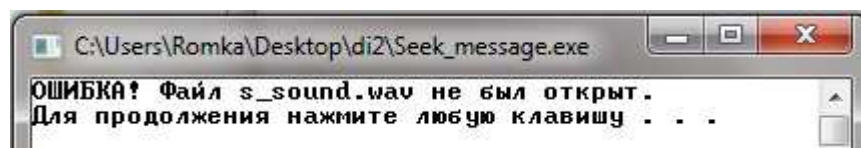


Рисунок 49 - Реакция программы Seek_message.exe на отсутствия в каталоге файла – контейнера s_sound.wav

Файл – контейнер может не соответствовать требованиям формата или метода кодировки звуковых отсчетов. Для проверки реакции на подобные ошибки в каталоге программ Hide_message.exe был помещен сторонний файл (например, обычный текстовый файл) и переименован в sound.wav. Так как этот файл на самом деле не является файлом формата wav (а следовательно не содержит RIFF контейнер) программа выводит на экран сообщение «ОШИБКА! Файл sound.wav не содержит RIFF контейнер.» и завершает работу.

Однако наличие RIFF контейнера свойственно не только файлам формата wav. Таким файлом может оказаться файл формата avi, rmid и другие. Для проверки реакции программы на ошибку, когда файл, содержащий RIFF контейнер не является файлом формата wav, в ее каталоге был создан файл sound.wav, путем переименования из файла формата avi (который содержит RIFF контейнер, но отличается от формата wav). Программа была запущена, вывела на экран сообщение «ОШИБКА! Файл sound.wav не является WAVE файлом.» и завершила работу. Результаты показаны на рисунке 50.

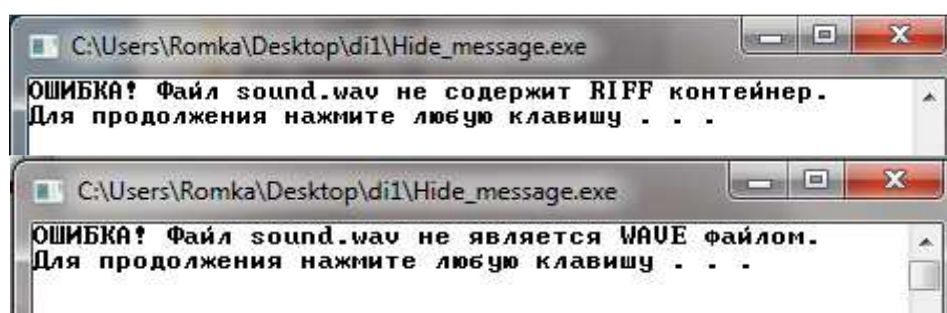


Рисунок 50 – Реакция программы Hide_message.exe на несоответствие требованиям формата файла

Для проверки реакции программы на несоответствие файла sound.wav требованию наличия метода/формата кодировки РСМ (импульсно – кодовой модуляции) в каталоге был размещен данный файл с другим методом

кодировки. Программа была запущена и вывела на экран сообщение «ОШИБКА! Файл sound.wav не является форматом PCM WAVE файла.» и завершила работу. Результаты показаны на рисунке 51.

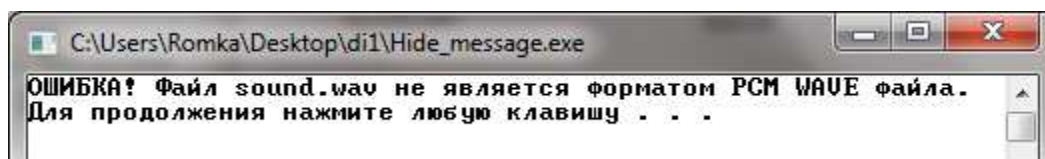


Рисунок 51 – Реакция программы Hide_message.exe на несоответствие требованиям метода кодировки звуковых отсчетов в файле sound.wav

Таким образом, программа Hide_message.exe не будет производить шифрование и скрывание сообщения в файл, не соответствующий предъявляемым требованиям. Реакции на несоответствие файла s_sound.wav предъявляемым требованиям в программе Seek_message.exe аналогичные.

В завершении была смоделирована ситуация, когда текст зашифрованного сообщения не помещается в файл – контейнер. Для этого, в файл text.txt был помещен текст длиной 3121 символов (заведомо больший, чем область звуковых данных в файле – контейнере sound.wav). Программа была запущена и вывела на экран сообщение «ОШИБКА! Шифротекст (3128 байт) не помещается в контейнер. Контейнер может вместить только 2041 байт шифротекста», после чего завершила свою работу, показано на рисунке 52.

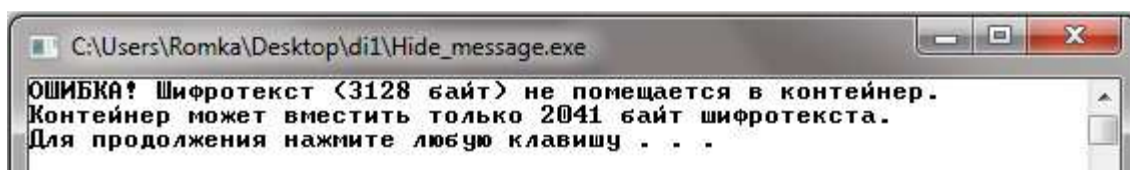


Рисунок 52 - Реакция программы Hide_message.exe на невозможность скрыть зашифрованный текст в файле sound.wav

Таким образом, программа при выводе сообщения об ошибке информирует пользователя о количестве символов зашифрованного текста, которые могут быть скрыты в предлагаемом файле – контейнере для скрывания.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта был произведен обзор цифровых стеганографических методов скрытия информации. Разработаны и описаны алгоритмы программ, осуществляющих зашифрование со скрытием сообщений в файл – контейнер и извлечение с расшифрованием сообщений из данного файла, произведена их программная реализация. Проведена экспериментальная проверка данных программ.

В результате выполнения дипломного проекта было создано программное обеспечение, реализующее защиту передаваемых сообщений в соответствии со всеми требованиями технического задания.

Таким образом, цель дипломного проекта достигнута, требования технического задания на выпускную квалифицированную работу были выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Грибунин В. Г., Оков И. Н., Туринцев И. В. Цифровая стеганография // М.: Солон-Пресс, 2009. — 265 с.
- 2 Коркач И.В., Пирогова Ю.И. Использование технологий IP-телефонии для скрытой передачи информации. (рус.) // Информационная безопасность человека, общества, государства.. — 2012. — Т. 9, № 2. — С. 124–128.
- 3 Барсуков В.С., Романцов А.П. Компьютерная стеганография вчера, сегодня, завтра. Технологии информационной безопасности 21 века // Журнал "специальная техника" № 4-5 1998 г.
- 4 Метод замены цветовой палитры [Электронный ресурс]. Режим доступа: http://www.nestego.ru/2012/07/blog-post_31.html, свободный.
- 5 Кустов В.Н., Федчук А.А. Методы встраивания скрытых сообщений //Конфидент. 2000. №3. С. 34-37.
- 6 Быков С.Ф. Алгоритм сжатия JPEG с позиции компьютерной стеганографии //Конфидент. 2000. №3. С. 26-33.
- 7 Нигматуллин Э. В., Ковырзина К. С. Обзор методов цифровой аудио стеганографии // Научное сообщество студентов XXI столетия. Технические науки: сб. ст. по мат. XLII междунар. студ. науч.-практ. конф. № 5(41). URL: [http://sibac.info/archive/technic/5\(41\).pdf](http://sibac.info/archive/technic/5(41).pdf)
- 8 Замена наименее значащего бита или LSB [Электронный ресурс]. Режим доступа: <http://www.nestego.ru/2012/07/lbsb.html>, свободный.
- 9 ГОСТ 28147–89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – М.: Изд-во стандартов, 1989. – 26с.

ПРИЛОЖЕНИЕ А

(обязательное)

Код программы, реализующей шифрование и скрывание сообщений

```
//Hide_message.exe
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct audio//структура для WAV-контейнера
{
    unsigned int chunkID;    //Содержит символы "RIFF" в ASCII
                             //кодировке (0x52494646 в big-endian представлении). Является
                             //началом RIFF-цепочки.
    unsigned int chunkSize; //Это оставшийся размер цепочки,
                             //начиная с этой позиции. Иначе говоря, это размер файла - 8, то
                             //есть, исключены поля chunkID и chunkSize.
    unsigned int format;    //Содержит символы "WAVE" (0x57415645
                             //в big-endian представлении)
    char subChunk1ID[4];    //Содержит символы "fmt " (0x666d7420
                             //в big-endian представлении)
    unsigned int subChunk1Size; //16 для формата PCM. Это
                             //оставшийся размер подцепочки, начиная с этой позиции.
    unsigned int audioFormat; //Аудио формат, полный список
                             //можно получить здесь. Для PCM = 1 (то есть, Линейное квантование).
                             //Значения, отличающиеся от 1, обозначают некоторый формат сжатия.
    unsigned int numChannels; //Количество каналов. Моно = 1,
                             //Стерео = 2 и т.д.
    unsigned int sampleRate; //Частота дискретизации. 8000 Гц,
                             //44100 Гц и т.д.
    unsigned int byteRate; //Количество байт, переданных за
                             //секунду воспроизведения.
    unsigned int blockAlign; //Количество байт для одного сэмпла,
                             //включая все каналы.
    unsigned int bitsPerSample; //Количество бит в сэмпле. Так
                             //называемая "глубина" или точность звучания. 8 бит, 16 бит и т.д.
    char subChunk2ID[4];    //Содержит символы "data" (0x64617461
                             //в big-endian представлении)
    unsigned int subChunk2Size; //Количество байт в области
                             //данных.
    unsigned int data;
} AUDIO;
```

```

int main ()
{
    std::locale::global(std::locale(""));
    int i,j,round,block,k;//счетчики
    long key_length=0;//длина ключа в символах/байтах
    int text_length=0;//длина о.т в символах/байтах
    int dop_length=0;//число символов, дополняющих о.т до
кратности 64 битам
    int blocks=0;//число блоков о.т по 64 бита
    unsigned int x0,x1,x2,x3,x4,x5,x6,x7;//КЗУ
    unsigned int x_array[32];//массив в котором КЗУ выстроены в
порядке нужном для зашифрования
    unsigned int N1=0,N2=0;//накопители
    unsigned int CM1=0,CM2=0;
    unsigned int CM1_array[8]={0,0,0,0,0,0,0,0};//массив для
разбиения CM1 на части по 4 бита
    unsigned int K=0;//значение CM1 после подстановки
    unsigned int R=0,r=0;
    int tab[8][16] = {
        {
0x4,0xA,0x9,0x2,0xD,0x8,0x0,0xE,0x6,0xB,0x1,0xC,0x7,0xF,0x5,0x3
},//k1
        {
0xE,0xB,0x4,0xC,0x6,0xD,0xF,0xA,0x2,0x3,0x8,0x1,0x0,0x7,0x5,0x9
},//k2
        {
0x5,0x8,0x1,0xD,0xA,0x3,0x4,0x2,0xE,0xF,0xC,0x7,0x6,0x0,0x9,0xB
},//k3
        {
0x7,0xD,0xA,0x1,0x0,0x8,0x9,0xF,0xE,0x4,0x6,0xC,0xB,0x2,0x5,0x3
},//k4
        {
0x6,0xC,0x7,0x1,0x5,0xF,0xD,0x8,0x4,0xA,0x9,0xE,0x0,0x3,0xB,0x2
},//k5
        {
0x4,0xB,0xA,0x0,0x7,0x2,0x1,0xD,0x3,0x6,0x8,0x5,0x9,0xC,0xF,0xE
},//k6
        {
0xD,0xB,0x4,0x1,0x3,0xF,0x5,0x9,0x0,0xA,0xE,0x7,0x6,0x8,0x2,0xC
},//k7
        {
0x1,0xF,0xD,0x0,0x5,0x7,0xA,0x4,0x9,0x2,0x3,0xE,0x6,0xB,0x8,0xC
},//k8
    };//таблица замен Идентификатор: id-GostR3411-94-TestParamSet

//1.РАБОТА С КЛЮЧОМ

```

```

//1.1.открытие файла для определения длины (V)
//открытие файла ключа
FILE * ptrFile = fopen("key.txt", "rb");
if (ptrFile == NULL)
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    {
        system ("pause");
        return -1;
    }
}
else
{
    fseek(ptrFile, 0, SEEK_END); // переместить внутренний
указатель в конец файла
    key_length = ftell(ptrFile); // вернуть текущее положение
внутреннего указателя
    fclose (ptrFile); // закрыть файл
    if (key_length!=32)
    {
        cout<<"ОШИБКА! Файл ключа key.txt не равен 256
битам."<<endl;
        system ("pause");
        return -1;
    }
}

//1.2.открытие файла для чтения (V)
unsigned int *key_array = new unsigned int
[key_length]; //динамический массив под хранение кодов символов
ключа
ifstream keystream("key.txt", ios::binary);
if (!(keystream.is_open()))
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    system ("pause");
    return -1;
}
//чтение ключа
for (i=0; i<key_length; i++)
{
    key_array[i]=0;
    keystream.read((char*)&key_array[i], 1);
}
//закрытие
keystream.close();

```

```
//1.3 Заполнение КЗУ (V)
x0=(key_array[0]<<24)+(key_array[1]<<16)+(key_array[2]<<8)+key_array[3];
x1=(key_array[4]<<24)+(key_array[5]<<16)+(key_array[6]<<8)+key_array[7];
x2=(key_array[8]<<24)+(key_array[9]<<16)+(key_array[10]<<8)+key_array[11];
x3=(key_array[12]<<24)+(key_array[13]<<16)+(key_array[14]<<8)+key_array[15];
x4=(key_array[16]<<24)+(key_array[17]<<16)+(key_array[18]<<8)+key_array[19];
x5=(key_array[20]<<24)+(key_array[21]<<16)+(key_array[22]<<8)+key_array[23];
x6=(key_array[24]<<24)+(key_array[25]<<16)+(key_array[26]<<8)+key_array[27];
x7=(key_array[28]<<24)+(key_array[29]<<16)+(key_array[30]<<8)+key_array[31];
```

```
//1.4 Заполнение Массива x_array последовательности подачи КЗУ (V)
x_array[0]=x_array[8]=x_array[16]=x0;
x_array[1]=x_array[9]=x_array[17]=x1;
x_array[2]=x_array[10]=x_array[18]=x2;
x_array[3]=x_array[11]=x_array[19]=x3;
x_array[4]=x_array[12]=x_array[20]=x4;
x_array[5]=x_array[13]=x_array[21]=x5;
x_array[6]=x_array[14]=x_array[22]=x6;
x_array[7]=x_array[15]=x_array[23]=x7;
x_array[24]=x7;
x_array[25]=x6;
x_array[26]=x5;
x_array[27]=x4;
x_array[28]=x3;
x_array[29]=x2;
x_array[30]=x1;
x_array[31]=x0;
```

```
//2. РАБОТА С О.Т
```

```
//2.1.открытие файла для определения длины (V)
```

```
//открытие файла о.т.
```

```
FILE * ptrFile2 = fopen("text.txt", "rb");
```

```
if (ptrFile2 == NULL)
```

```
{
```

```
    cout<<"ОШИБКА! Файл открытого текста text.txt не был  
открыт."<<endl;
```

```
    system ("pause");
```

```

        return -1;
    }
    else
    {
        fseek(ptrFile2, 0, SEEK_END); // переместить внутренний
указатель в конец файла
        text_length = ftell(ptrFile2); // вернуть текущее положение
внутреннего указателя
        fclose (ptrFile2); // закрыть файл
    }

//2.2 проверка длины о.т (V)
if (text_length <4)//если в о.т меньше 4 символов
{
    cout<<"ОШИБКА! Файл открытого текста text.txt должен соержать
не менее 4 символов."<<endl;
    system ("pause");
    return -1;
}
//подготовка о.т. к шифрованию

//2.3 подсчет числа недостающих в о.т символов до кратности 64
битам (V)
if ((text_length%8)!=0)
    dop_length=8-text_length%8;

//2.4 определение числа блоков (по 64 бита) (V)
blocks=(text_length+dop_length)/8;

//2.5 открытие файла о.т для чтения (V)
unsigned int *text_array = new unsigned int
[text_length+dop_length];//динамический массив под хранение кодов
символов о.т + дополнения
unsigned int *dop_array = new unsigned int
[dop_length];//динамический массив под хранение кодов символов
дополнения
unsigned int *crypt_array = new unsigned int
[text_length+dop_length];//динамический массив под хранение кодов
символов ш.т
//обнуление crypt_array
for (i=0;i<text_length+dop_length;i++)
{
    crypt_array[i]=0;
}
ifstream textstream("text.txt",ios::binary);
if (!(textstream.is_open()))

```

```

    {
        cout<<"ОШИБКА! Файл открытого текста text.txt не был
открыт."<<endl;
        system ("pause");
        return -1;
    }
//чтение о.т
for (i=0;i<text_length+dop_length;i++)
{
    text_array[i]=0;
    if (i<text_length)
        textstream.read((char*)&text_array[i],1);
}
//закрытие
textstream.close();

//2.6 определение дополняющих символов
for (i=0;i<dop_length;i++)
{
    // тут выбирается какими символами дополнять о.т
    if (i%2==0)//если символ о.т четный
        dop_array[i]=text_array[i]-1;
    else //иначе
        dop_array[i]=text_array[i]+1;
}

//2.7 внедрение дополняющих символов
j=0;
for (i=text_length;i<text_length+dop_length;i++)//в незаполненные
ячейки массива text_array внести ячейки dop_array
{
    text_array[i]=dop_array[j];
    j++;
}

//3.ШИФРОВАНИЕ
//Для каждого блока..
for (block=0;block<blocks;block++)//начало поблочного цикла
{
    //первый цикл 32-3
    //определение начального заполнения N1 и N2 (перед первым
циклом)
    N1=(text_array[0+8*block]<<24)+(text_array[1+8*block]<<16
)+(text_array[2+8*block]<<8)+text_array[3+8*block];//(V)
    N2=(text_array[4+8*block]<<24)+(text_array[5+8*block]<<16
)+(text_array[6+8*block]<<8)+text_array[7+8*block];//(V)

```

```

//циклы 2...32 32-3
for (round=0;round<32/*32*/;round++)//со 2-го цикла 32-3
{

    //подсчет CM1 - сложение по модулю 2^32
    CM1=N1+x_array[round];// (V)
    //заполнение CM1_array для подстановки(V)
    CM1_array[0]=(CM1&0xf0000000)>>28;
    CM1_array[1]=(CM1&0xf000000)>>24;
    CM1_array[2]=(CM1&0xf00000)>>20;
    CM1_array[3]=(CM1&0xf0000)>>16;
    CM1_array[4]=(CM1&0xf000)>>12;
    CM1_array[5]=(CM1&0xf00)>>8;
    CM1_array[6]=(CM1&0xf0)>>4;
    CM1_array[7]=CM1&0xf;
    //произведение подстановки(V)
    for (i=0;i<8;i++)//для каждой четверки бит CM1
    {
        CM1_array[i]=tab[7-i][CM1_array[i]];
        //подстановка
    }
    //сборка массива CM1_array после подстановки в переменную
    K(V)

    K=(CM1_array[0]<<28)+(CM1_array[1]<<24)+(CM1_array[2]<<20)+(CM1_ar
    ray[3]<<16)+(CM1_array[4]<<12)+(CM1_array[5]<<8)+(CM1_array[6]<<4)
    +(CM1_array[7]);
    //циклический сдвиг на 11 влево (V)
    r=(K&0xffe00000)>>21;//старшая часть помещается в r и
    сдвигается в конец младший
    R=(K<<11)+r;//сдвиг числа на 11 влево + r
    //сложение по модулю 2 (XOR) (V)
    CM2=R^N2;
    // N1 -> N2, CM2 -> N1 (V)
    if (round!=31)//для всех циклов кроме последнего 32-го
    {
        N2=N1;
        N1=CM2;
    }
    else //для последнего цикла
    {
        N2=CM2;
    }
}
} //конец цикла 32-3
//запись N1,N2 в массив crypt_array
crypt_array[0+block*8]=(N1&0xff000000)>>24;

```

```

    crypt_array[1+block*8]=(N1&0x00ff0000)>>16;
    crypt_array[2+block*8]=(N1&0x0000ff00)>>8;
    crypt_array[3+block*8]=(N1&0x000000ff);
    crypt_array[4+block*8]=(N2&0xff000000)>>24;
    crypt_array[5+block*8]=(N2&0x00ff0000)>>16;
    crypt_array[6+block*8]=(N2&0x0000ff00)>>8;
    crypt_array[7+block*8]=(N2&0x000000ff);
} //конец поблочного цикла

//СКРЫТИЕ В СТЕГАНОГРАФИЧЕСКИЙ КОНТЕЙНЕР
//Инициализация переменных
unsigned int full_length=text_length+dop_length+5;//число символов
ш.т + служебные 5 байт
unsigned int pole1[32]; //массив для хранения 4 байт общей длины
массива array_to_hide в символах(байтах), включая служебную инф.
unsigned int *pole2 = new unsigned int
[(text_length+dop_length)*8]; //байты ш.т
unsigned int pole3[8]; //последний байт отводится под число длины
дополнения (число от 0 до 7)
unsigned int *array_to_hide = new unsigned int
[full_length*8]; //динамический массив под хранение бит символов
ш.т + служебной информации
//в данном массиве: первые 4 байта отводятся под хранение общей
длины массива в символах(байтах), включая служебную инф.
//последний байт отводится под число длины дополнения (число от 0
до 7)
//а байты между ними - сам ш.т
unsigned int tail_length=0; //длина хвостовой части
unsigned int tail_position=0; //позиция начала хвостовой части
unsigned int samples=0; //число сэмплов в файле
unsigned int subsamples=0; //число сэмплов * число каналов
unsigned int left=0, right=0; //значения для левого и правого
каналов
unsigned int can_hide=0; //число символов/байт для скрытия в данном
контейнере

//5. подготовка ш.т. к скрытию
//5.1 заполнение массива pole1 (V)
for (i=0; i<32; i++)
{
    pole1[i]=0; //обнуление
    pole1[i]=(full_length>>(31-i))&0x1;
}

//5.2 заполнение массива pole2 (V)
for (i=0; i<text_length+dop_length; i++)

```

```

{
    pole2[0+i*8]=(crypt_array[i]>>7)&0x1;
    pole2[1+i*8]=(crypt_array[i]>>6)&0x1;
    pole2[2+i*8]=(crypt_array[i]>>5)&0x1;
    pole2[3+i*8]=(crypt_array[i]>>4)&0x1;
    pole2[4+i*8]=(crypt_array[i]>>3)&0x1;
    pole2[5+i*8]=(crypt_array[i]>>2)&0x1;
    pole2[6+i*8]=(crypt_array[i]>>1)&0x1;
    pole2[7+i*8]=crypt_array[i]&0x1;
}

//5.3 заполнение массива pole3 (V)
for (i=0;i<8;i++)
{
    pole3[i]=(dop_length>>(7-i))&0x1;
}

//5.4 заполнение массива array_to_hide (V)
//первые 4 байта
for (i=0;i<32;i++)
{
    array_to_hide[i]=pole1[i];
}
//байты [5 ... до предпоследнего]
for (i=32;i<(full_length*8)-8;i++)
{
    array_to_hide[i]=pole2[i-32];
}
//последний байт
for (i=(full_length*8)-8;i<(full_length*8);i++)
{
    array_to_hide[i]=pole3[i-((full_length*8)-8)];
}

//6.ЧТЕНИЕ ДАННЫХ О КОНТЕЙНЕРЕ
unsigned int wavheader_bigger[1000]; //массив для хранения
заголовка Wav
unsigned int data_position=0; //позиция слова data
unsigned int wavheader_length=0; //длина заголовка в байтах
int trash=0; // мусор

//6.1 открытие файла контейнера для поиска позиции слова data
ifstream soundstream0("sound.wav",ios_base::binary);
if (!(soundstream0.is_open()))
{
    cout<<"ОШИБКА! Файл sound.wav не был открыт."<<endl;

```

```

        system ("pause");
        return -1;
    }
    //побайтное чтение в массив wavheader_bigger
    for (i=0;i<1000;i++)
    {
        wavheader_bigger[i]=0;
        soundstream0.read((char*)&wavheader_bigger[i],1);
    }
    //поиск в массиве wavheader_bigger байт слова data
    for (i=0;i<1000;i++)
    {
        data_position=i+1;
        if
        ((wavheader_bigger[i]==0x64)&&(wavheader_bigger[i+1]==0x61)&&(wavh
eader_bigger[i+2]==0x74)&&(wavheader_bigger[i+3]==0x61))
            break;
    }
    unsigned int *wavheader_real = new unsigned int
[data_position+7]; // дин.массив длиной = длине заголовка (всё без
сэмплов), содержащий байты заголовка
    for (i=0;i<data_position+7;i++)
    {
        wavheader_real[i]=wavheader_bigger[i];
    }
    //определение длины даты (сэмплов)
    AUDIO.subChunk2Size=wavheader_real[data_position+3]|wavheader_real
[data_position+4]<<8|wavheader_real[data_position+5]<<16|wavheader
_real[data_position+6]<<24;
    //определение длины заголовка
    wavheader_length=data_position+7;
    //сброс каретки/возврат к началу файла
    soundstream0.clear();
    soundstream0.seekg(0);

    //6.2 чтение информации о контейнере в переменные структуры audio
    soundstream0.read((char*)&AUDIO.chunkID,4); //Содержит символы
"RIFF" в ASCII кодировке (0x52494646 в big-endian представлении).
Является началом RIFF-цепочки.
    soundstream0.read((char*)&AUDIO.chunkSize,4); //Это оставшийся
размер цепочки, начиная с этой позиции. Иначе говоря, это размер
файла - 8, то есть, исключены поля chunkId и chunkSize.
    soundstream0.read((char*)&AUDIO.format,4); //Содержит символы
"WAVE" (0x57415645 в big-endian представлении)
    soundstream0.read((char*)&AUDIO.subChunk1ID,4); //Содержит
символы "fmt " (0x666d7420 в big-endian представлении)

```

```

    soundstream0.read((char*)&AUDIO.subChunk1Size,4); //16 для
формата PCM. Это оставшийся размер подцепочки, начиная с этой
позиции.
    soundstream0.read((char*)&AUDIO.audioFormat,2); //Аудио
формат, полный список можно получить здесь. Для PCM = 1 (то есть,
Линейное квантование). Значения, отличающиеся от 1, обозначают
некоторый формат сжатия.
    soundstream0.read((char*)&AUDIO.numChannels,2); //Количество
каналов. Моно = 1, Стерео = 2 и т.д.
    soundstream0.read((char*)&AUDIO.sampleRate,4); //Частота
дискретизации. 8000 Гц, 44100 Гц и т.д.
    soundstream0.read((char*)&AUDIO.byteRate,4); //Количество байт,
переданных за секунду воспроизведения.
    soundstream0.read((char*)&AUDIO.blockAlign,2); //Количество
байт для одного сэмпла, включая все каналы.
    soundstream0.read((char*)&AUDIO.bitsPerSamle,2); //Количество
бит в сэмпле. Так называемая "глубина" или точность звучания. 8
бит, 16 бит и т.д.
    soundstream0.read((char*)&AUDIO.subChunk2ID,4); //Содержит
символы "data" (0x64617461 в big-endian представлении)
    soundstream0.read((char*)&trash,4); //Количество байт в области
данных.
    //определение длины хвостовой части
tail_length=AUDIO.chunkSize+8-wavheader_length-
AUDIO.subChunk2Size;
    //определение позиции хвостовой части
tail_position=AUDIO.chunkSize+8-tail_length;

//6.3 проверка на ошибки
//проверка формата RIFF
if(AUDIO.chunkID!=0x46464952)
{
    cout<<"ОШИБКА! Файл sound.wav не содержит RIFF
контейнер."<<endl;
    system ("pause");
    return -1;
}
//проверка формата WAVE
if(AUDIO.format!=0x45564157)
{
    cout<<"ОШИБКА! Файл sound.wav не является WAVE файлом."<<endl;
    system ("pause");
    return -1;
}
//проверка на наличие формата PCM в audioFormat
if(AUDIO.audioFormat!=1)

```

```

{
    cout<<"ОШИБКА! Файл sound.wav не является форматом PCM WAVE
файла."<<endl;
    system ("pause");
    return -1;
}

//7.ПОДСЧЕТ НЕОБХОДИМОЙ ИНФОРМАЦИИ ПО КОНТЕЙНЕРУ (V)
//определение числа звуковых отсчетов (V)
samples = (AUDIO.subChunk2Size)/AUDIO.blockAlign;
//определение числа байт для хранения звуковых отсчетов (V)
subsamples = AUDIO.subChunk2Size;
//определение вместимости контейнера(в символах) при замене 1
младшего бита (V)
can_hide=subsamples/8/(AUDIO.blockAlign/AUDIO.numChannels);
    //уместится ли array_to_hide в контейнере?
if (full_length > can_hide)
{
    cout<<"ОШИБКА! Шифротекст ("<<full_length-5<<" байт) не
помещается в контейнер."<<endl;
    cout<<"Контейнер может вместить только "<<can_hide-5<<" байт
шифротекста."<<endl;
    system ("pause");
    return -1;
}
//переход каретки к данным
soundstream0.clear();
soundstream0.seekg(wavheader_length);

//8.ЧТЕНИЕ БАЙТ ЗВУКОВЫХ ОТСЧЕТОВ
unsigned int *subsamples_array = new unsigned int
[subsamples];//создание динамического массива для хранения байт
звуковых отсчетов
for (i=0;i<subsamples;i++)
{
    subsamples_array[i]=0;//обнуление
    soundstream0.read((char*)&subsamples_array[i],1);
}

//8.1 чтение хвоста
unsigned int *tail_array = new unsigned int
[tail_length];//создание динамического массива для хранения байт
хвоста
for (i=0;i<tail_length;i++)
{
    tail_array[i]=0;//обнуление

```

```

        soundstream0.read((char*)&tail_array[i],1);
    }
    soundstream0.close();

//9.ГЕНЕРАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ НОМЕРОВ БАЙТ ЗВУКОВЫХ ОТСЧЕТОВ,
ПОДЛЕЖАЩИХ ЗАМЕНЕ
    //unsigned long int *subsamples_nums = new unsigned long int
    [full_length*8];//создание динамического массива для хранения
    номеров субсэмплров, где будут заменены младшие биты
    unsigned long int *subsamples_nums = new unsigned long int
    [subsamples];//создание динамического массива для хранения номеров
    субсэмплров для 8 битных wav
    unsigned long int *subsamples_nums2 = new unsigned long int
    [subsamples];//создание динамического массива для хранения номеров
    субсэмплров, где будут заменены младшие биты . для не 8 битных wav
    unsigned int *subsamples_status = new unsigned int
    [subsamples];//0 - субсэмпл свободен, 1 - занят
    if (subsamples>1000000)
        cout<<"Подождите..."<<endl;
        //обнуление subsamples_status, subsamples_nums2
    for (i=0;i<subsamples;i++)
    {
        subsamples_status[i]=0;
        subsamples_nums2[i]=0;
    }
    //генератор
        //начальное заполнение
        subsamples_nums[0]=subsamples/2;
        subsamples_status[subsamples_nums[0]]++;
    //далее заполнение массива
    for (i=1;i<subsamples;i++)
    {
        subsamples_nums[i]=subsamples_nums[i-1]+(0+rand()%subsamples);
        //если переполнение
        if (subsamples_nums[i]>subsamples)
            subsamples_nums[i]=subsamples_nums[i]%subsamples;
            again:
            if (subsamples_status[subsamples_nums[i]]!=0)
            {
                subsamples_nums[i]=subsamples_nums[i]+(0+rand()%subsamples);
                if (subsamples_nums[i]>subsamples)
                    subsamples_nums[i]=subsamples_nums[i]%subsamples;
                    subsamples_nums[i]--;
                goto again;
            }
            subsamples_status[subsamples_nums[i]]++;

```

```

    }
//заполнение subsamples_nums2
j=0;
for (i=0;i<subsamples;i++)
{
    if
((subsamples_nums[i])%(AUDIO.blockAlign/AUDIO.numChannels)==0)
    {
        subsamples_nums2[j]=subsamples_nums[i];
        j++;
    }
}

//10.ЗАМЕНА МЛАДШИХ БИТ
for (i=0;i<full_length*8;i++)
{
    //отброс старого младшего бита и внесение нового из
array_to_hide
    if (AUDIO.blockAlign==1)//для 8 бит
        subsamples_array[subsamples_nums[i]]=
(subsamples_array[subsamples_nums[i]]&0xfe)|array_to_hide[i];
    else //для остальных
        subsamples_array[subsamples_nums2[i]]=
(subsamples_array[subsamples_nums2[i]]&0xfe)|array_to_hide[i];
}

//11.СОЗДАНИЕ ВЫХОДНОГО ФАЙЛА
ofstream ofstr("s_sound.wav", ios_base::out | ios::binary
|ios_base::trunc);//создание выходного файла
//запись заголовка
for (i=0;i<wavheader_length;i++)
{
    ofstr.write((char*)&wavheader_real[i],1);
}
//запись байт звуковых отсчетов
for (i=0;i<subsamples;i++)
{
    ofstr.write((char*)&subsamples_array[i],1);
}
//запись хвоста если он есть
if (tail_length!=0)
    for (i=0;i<tail_length;i++)
    {
        ofstr.write((char*)&tail_array[i],1);
    }
}
//закрытие файла

```

```
ofstr.close();  
//КОНЕЦ  
cout<<endl<<"Сообщение зашифровано и скрыто!"<<endl;  
system("pause");  
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Код программы, реализующей извлечение и расшифрование сообщений

```
//Seek_message.exe
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct audio//структура для WAV-контейнера
{
    unsigned int chunkID;          //Содержит символы "RIFF" в
ASCII кодировке (0x52494646 в big-endian представлении). Является
началом RIFF-цепочки.
    unsigned int chunkSize; //Это оставшийся размер цепочки,
начиная с этой позиции. Иначе говоря, это размер файла - 8, то
есть, исключены поля chunkId и chunkSize.
    unsigned int format;          //Содержит символы "WAVE"
(0x57415645 в big-endian представлении)
    char subChunk1ID[4];         //Содержит символы "fmt " (0x666d7420
в big-endian представлении)
    unsigned int subChunk1Size; //16 для формата PCM. Это
оставшийся размер подцепочки, начиная с этой позиции.
    unsigned int audioFormat;     //Аудио формат, полный список
можно получить здесь. Для PCM = 1 (то есть, Линейное квантование).
Значения, отличающиеся от 1, обозначают некоторый формат сжатия.
    unsigned int numChannels;     //Количество каналов. Моно = 1,
Стерео = 2 и т.д.
    unsigned int sampleRate; //Частота дискретизации. 8000 Гц,
44100 Гц и т.д.
    unsigned int byteRate;       //Количество байт, переданных за
секунду воспроизведения.
    unsigned int blockAlign; //Количество байт для одного сэмпла,
включая все каналы.
    unsigned int bitsPerSamle;   //Количество бит в сэмпле. Так
называемая "глубина" или точность звучания. 8 бит, 16 бит и т.д.
    char subChunk2ID[4];         //Содержит символы "data"
(0x64617461 в big-endian представлении)
    unsigned int subChunk2Size; //Количество байт в области
данных.
    unsigned int data;
} AUDIO;
```

```

int main ()
{
    std::locale::global(std::locale(""));
    //Инициализация начальных переменных
    int i,j,round,block;//счетчики
    unsigned int samples=0;
    unsigned int subsamples=0;
    unsigned int full_length=0;
    unsigned int dop_length=0;
    unsigned int text_length=0;

    //1.ЧТЕНИЕ ДАННЫХ О КОНТЕЙНЕРЕ
    unsigned int wavheader_bigger[1000]; //массив для хранения
    заголовка Wav
    unsigned int data_position=0; //позиция слова data
    unsigned int wavheader_length=0; //длина заголовка в байтах
    int trash=0; // мусор

    //1.1 открытие файла контейнера для поиска позиции слова data
    ifstream soundstream0("s_sound.wav",ios_base::binary);
    if (!(soundstream0.is_open()))
    {
        cout<<"ОШИБКА! Файл s_sound.wav не был открыт."<<endl;
        system ("pause");
        return -1;
    }
    //побайтное чтение в массив wavheader_bigger
    for (i=0;i<1000;i++)
    {
        wavheader_bigger[i]=0;
        soundstream0.read((char*)&wavheader_bigger[i],1);
    }

    //поиск в массиве wavheader_bigger байт слова data
    for (i=0;i<1000;i++)
    {
        data_position=i+1;
        if
        ((wavheader_bigger[i]==0x64)&&(wavheader_bigger[i+1]==0x61)&&(wavh
        eader_bigger[i+2]==0x74)&&(wavheader_bigger[i+3]==0x61))
            break;
    }

    unsigned int *wavheader_real = new unsigned int
    [data_position+7]; // дин.массив длиной = длине заголовка (всё без
    звуковых отсчетов), содержащий байты заголовка
    for (i=0;i<data_position+7;i++)
    {

```

```

        wavheader_real[i]=wavheader_bigger[i];
    }
    //определение длины даты (сэмплов)
    AUDIO.subChunk2Size=wavheader_real[data_position+3]|wavheader_
real[data_position+4]<<8|wavheader_real[data_position+5]<<16|wavhe
ader_real[data_position+6]<<24;
    //определение длины заголовка
    wavheader_length=data_position+7;
    //сброс каретки/возврат к началу файла
    soundstream0.clear();
    soundstream0.seekg(0);

    //1.2 чтение информации о контейнере в переменные структуры
audio
    soundstream0.read((char*)&AUDIO.chunkID,4); //Содержит символы
"RIFF" в ASCII кодировке (0x52494646 в big-endian представлении).
Является началом RIFF-цепочки.
    soundstream0.read((char*)&AUDIO.chunkSize,4); //Это оставшийся
размер цепочки, начиная с этой позиции. Иначе говоря, это размер
файла - 8, то есть, исключены поля chunkId и chunkSize.
    soundstream0.read((char*)&AUDIO.format,4); //Содержит символы
"WAVE" (0x57415645 в big-endian представлении)
    soundstream0.read((char*)&AUDIO.subChunk1ID,4); //Содержит
символы "fmt " (0x666d7420 в big-endian представлении)
    soundstream0.read((char*)&AUDIO.subChunk1Size,4); //16 для
формата PCM. Это оставшийся размер подцепочки, начиная с этой
позиции.
    soundstream0.read((char*)&AUDIO.audioFormat,2); //Аудио
формат, полный список можно получить здесь. Для PCM = 1 (то есть,
Линейное квантование). Значения, отличающиеся от 1, обозначают
некоторый формат сжатия.
    soundstream0.read((char*)&AUDIO.numChannels,2); //Количество
каналов. Моно = 1, Стерео = 2 и т.д.
    soundstream0.read((char*)&AUDIO.sampleRate,4); //Частота
дискретизации. 8000 Гц, 44100 Гц и т.д.
    soundstream0.read((char*)&AUDIO.byteRate,4); //Количество байт,
переданных за секунду воспроизведения.
    soundstream0.read((char*)&AUDIO.blockAlign,2); //Количество
байт для одного сэмпла, включая все каналы.
    soundstream0.read((char*)&AUDIO.bitsPerSamle,2); //Количество
бит в сэмпле. Так называемая "глубина" или точность звучания. 8
бит, 16 бит и т.д.
    soundstream0.read((char*)&AUDIO.subChunk2ID,4); //Содержит
символы "data" (0x64617461 в big-endian представлении)
    soundstream0.read((char*)&trash,4); //Количество байт в области
данных.

```

```

//1.3 проверка на ошибки
//проверка формата RIFF
if (AUDIO.chunkID!=0x46464952)
{
    cout<<"ОШИБКА! Файл s_sound.wav не содержит RIFF
контейнер."<<endl;
    system ("pause");
    return -1;
}
//проверка формата WAVE
if (AUDIO.format!=0x45564157)
{
    cout<<"ОШИБКА! Файл s_sound.wav не является WAVE
файлом."<<endl;
    system ("pause");
    return -1;
}
//проверка на наличие формата PCM в audioFormat
if (AUDIO.audioFormat!=1)
{
    cout<<"ОШИБКА! Файл s_sound.wav не является форматом PCM WAVE
файла."<<endl;
    system ("pause");
    return -1;
}

//2.ПОДСЧЕТ НЕОБХОДИМОЙ ИНФОРМАЦИИ ПО КОНТЕЙНЕРУ (V)
//определение числа звуковых отсчетов (V)
samples = ((AUDIO.subChunk2Size)/AUDIO.blockAlign);
//определение числа сабсэмплов (V)
subsamples = AUDIO.subChunk2Size;
//переход каретки к данным
soundstream0.clear();
soundstream0.seekg(wavheader_length);

//3.ЧТЕНИЕ ЗВУКОВЫХ ОТСЧЕТОВ
unsigned int *subsamples_array = new unsigned int
[subsamples]; //создание динамического массива для хранения байт
звуковых отсчетов
for (i=0;i<subsamples;i++)
{
    subsamples_array[i]=0; //обнуление
    soundstream0.read((char*)&subsamples_array[i],1);
}

```

```

//4.ГЕНЕРАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ НОМЕРОВ ЗВУКВЫХ ОТСЧЕТОВ,
ПОДЛЕЖАЩИХ ЗАМЕНЕ
    unsigned long int *subsamples_nums = new unsigned long int
[subsamples]; //создание динамического массива для хранения номеров
звуковых отсчетов, где будут заменены младшие биты (для 8 битных
файлов)
    unsigned long int *subsamples_nums2 = new unsigned long int
[subsamples]; //создание динамического массива для хранения номеров
звуковых отсчетов, где будут заменены младшие биты
    unsigned int *subsamples_status = new unsigned int
[subsamples]; //0 - байт звукового отсчета свободен, 1 - занят
if (subsamples>1000000)
    cout<<"Подождите..."<<endl;
//обнуление subsamples_status, subsamples_nums2
for (i=0;i<subsamples;i++)
{
    subsamples_status[i]=0;
    subsamples_nums2[i]=0;
}
//генератор
//начальное заполнение
subsamples_nums[0]=subsamples/2;
subsamples_status[subsamples_nums[0]]++;
//далее заполнение массива
for (i=1;i<subsamples;i++)
{
    subsamples_nums[i]=subsamples_nums[i-1]+(0+rand()%subsamples);
    //если переполнение
    if (subsamples_nums[i]>subsamples)
        subsamples_nums[i]=subsamples_nums[i]%subsamples;
    again:
    if (subsamples_status[subsamples_nums[i]]!=0)
    {
        subsamples_nums[i]=subsamples_nums[i]+(0+rand()%subsamples);
        if (subsamples_nums[i]>subsamples)
            subsamples_nums[i]=subsamples_nums[i]%subsamples;
        subsamples_nums[i]--;
        goto again;
    }
    subsamples_status[subsamples_nums[i]]++;
}
//заполнение subsamples_nums2
j=0;
for (i=0;i<subsamples;i++)
{
    if ((subsamples_nums[i])%(AUDIO.blockAlign/AUDIO.numChannels)==0)

```

```

        {
            subsamples_nums2[j]=subsamples_nums[i];
            j++;
        }
    }

//5.ПОЛУЧЕНИЕ МАССИВА Ш.Т
unsigned int pole1[32];
//заполнение массива pole1
for (i=0;i<32;i++)
{
    if (AUDIO.blockAlign==1)//для 8 бит
        pole1[i]=subsamples_array[subsamples_nums[i]]&0x01;
    else //if (AUDIO.blockAlign==2)//для 16 бит
        pole1[i]=subsamples_array[subsamples_nums2[i]]&0x01;
}
//ПОЛУЧЕНИЕ full_length из pole1;
full_length=((pole1[31]&0x1)|((pole1[30]<<1)&0x2)|((pole1[29]<<2)&0x4)|((pole1[28]<<3)&0x8)|((pole1[27]<<4)&0x10)|((pole1[26]<<5)&0x20)|((pole1[25]<<6)&0x40)|((pole1[24]<<7)&0x80)|((pole1[23]<<8)&0x100)|((pole1[22]<<9)&0x200)|((pole1[21]<<10)&0x400)|((pole1[20]<<11)&0x800)|((pole1[19]<<12)&0x1000)|((pole1[18]<<13)&0x2000)|((pole1[17]<<14)&0x4000)|((pole1[16]<<15)&0x8000)|((pole1[15]<<16)&0x10000)|((pole1[14]<<17)&0x20000)|((pole1[13]<<18)&0x40000)|((pole1[12]<<19)&0x80000)|((pole1[11]<<20)&0x100000)|((pole1[10]<<21)&0x200000)|((pole1[9]<<22)&0x400000)|((pole1[8]<<23)&0x800000)|((pole1[7]<<24)&0x1000000)|((pole1[6]<<25)&0x2000000)|((pole1[5]<<26)&0x4000000)|((pole1[4]<<27)&0x8000000)|((pole1[3]<<28)&0x10000000)|((pole1[2]<<29)&0x20000000)|((pole1[1]<<30)&0x40000000)|((pole1[0]<<31)&0x80000000);

unsigned int *array_to_hide = new unsigned int [full_length*8-32];//динамический массив под хранение бит символов ш.т + последний бит служебной информации
//ПОЛУЧЕНИЕ массива pole2+pole3
for (i=0;i<full_length*8-32;i++)
{
    if (AUDIO.blockAlign==1)//для 8 бит

        array_to_hide[i]=subsamples_array[subsamples_nums[i+32]]&0x1;
    else //if (AUDIO.blockAlign==2)//для 16 бит
        array_to_hide[i]=subsamples_array[subsamples_nums2[i+32]]&0x1;
}

```

```
//получение dop_length из pole3
dop_length=      (array_to_hide[full_length*8-33]&0x1)
                  |((array_to_hide[full_length*8-34]<<1)&0x2)
                  |((array_to_hide[full_length*8-35]<<2)&0x4)
                  |((array_to_hide[full_length*8-36]<<3)&0x8)
                  |((array_to_hide[full_length*8-37]<<4)&0x10)
                  |((array_to_hide[full_length*8-38]<<5)&0x20)
                  |((array_to_hide[full_length*8-39]<<6)&0x40)
                  |((array_to_hide[full_length*8-40]<<7)&0x80);
```

3. РАСШИФРОВАНИЕ

```
long key_length=0;//длина ключа в символах/байтах
int crypt_length=0;//длина ш.т в символах/байтах
int blocks=0;//число блоков о.т по 64 бита
unsigned int x0,x1,x2,x3,x4,x5,x6,x7;//КЗУ
unsigned int x_array[32];//массив в котором КЗУ выстроены в
порядке нужном для зашифрования
unsigned int N1=0,N2=0;//накопители
unsigned int CM1=0,CM2=0;
unsigned int CM1_array[8]={0,0,0,0,0,0,0,0};//массив для
разбиения CM1 на части по 4 бита
unsigned int K=0;//значение CM1 после подстановки
unsigned int R=0,r=0;
int tab[8][16] = {
{
0x4,0xA,0x9,0x2,0xD,0x8,0x0,0xE,0x6,0xB,0x1,0xC,0x7,0xF,0x5,0x3
},//k1
{
0xE,0xB,0x4,0xC,0x6,0xD,0xF,0xA,0x2,0x3,0x8,0x1,0x0,0x7,0x5,0x9
},//k2
{
0x5,0x8,0x1,0xD,0xA,0x3,0x4,0x2,0xE,0xF,0xC,0x7,0x6,0x0,0x9,0xB
},//k3
{
0x7,0xD,0xA,0x1,0x0,0x8,0x9,0xF,0xE,0x4,0x6,0xC,0xB,0x2,0x5,0x3
},//k4
{
0x6,0xC,0x7,0x1,0x5,0xF,0xD,0x8,0x4,0xA,0x9,0xE,0x0,0x3,0xB,0x2
},//k5
{
0x4,0xB,0xA,0x0,0x7,0x2,0x1,0xD,0x3,0x6,0x8,0x5,0x9,0xC,0xF,0xE
},//k6
{
0xD,0xB,0x4,0x1,0x3,0xF,0x5,0x9,0x0,0xA,0xE,0x7,0x6,0x8,0x2,0xC
},//k7
```

```

        {
0x1,0xF,0xD,0x0,0x5,0x7,0xA,0x4,0x9,0x2,0x3,0xE,0x6,0xB,0x8,0xC
} //k8
}; //таблица замен Идентификатор: id-GostR3411-94-TestParamSet

//1. РАБОТА С КЛЮЧОМ
//1.1. открытие файла для определения длины (V)
FILE * ptrFile = fopen("key.txt", "rb");
if (ptrFile == NULL)
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    system ("pause");
    return -1;
}
Else
{
    fseek(ptrFile, 0, SEEK_END); // переместить внутренний
указатель в конец файла
    key_length = ftell(ptrFile); // вернуть текущее положение
внутреннего указателя
    fclose (ptrFile); // закрыть файл
    if (key_length!=32)
    {
        cout<<"ОШИБКА! Файл ключа key.txt не равен 256
битам."<<endl;
        system ("pause");
        return -1;
    }
}

//1.2. открытие файла для чтения (V)
unsigned int *key_array = new unsigned int
[key_length]; //динамический массив под хранение кодов символов
ключа
ifstream keystream("key.txt", ios::binary);
if (!(keystream.is_open()))
{
    cout<<"ОШИБКА! Файл ключа key.txt не был открыт."<<endl;
    system ("pause");
    return -1;
}
//чтение ключа
for (i=0; i<key_length; i++)
{
    key_array[i]=0;
    keystream.read((char*)&key_array[i], 1);
}

```

```

}
//закрытие
keystream.close();

//1.3 Заполнение КЗУ (V)
    x0=(key_array[0]<<24)+(key_array[1]<<16)+(key_array[2]<<8)+key
_array[3];
    x1=(key_array[4]<<24)+(key_array[5]<<16)+(key_array[6]<<8)+key
_array[7];
    x2=(key_array[8]<<24)+(key_array[9]<<16)+(key_array[10]<<8)+ke
y_array[11];
    x3=(key_array[12]<<24)+(key_array[13]<<16)+(key_array[14]<<8)+
key_array[15];
    x4=(key_array[16]<<24)+(key_array[17]<<16)+(key_array[18]<<8)+
key_array[19];
    x5=(key_array[20]<<24)+(key_array[21]<<16)+(key_array[22]<<8)+
key_array[23];
    x6=(key_array[24]<<24)+(key_array[25]<<16)+(key_array[26]<<8)+
key_array[27];
    x7=(key_array[28]<<24)+(key_array[29]<<16)+(key_array[30]<<8)+
key_array[31];

    //1.4 Заполнение Массива x_array последовательности подачи КЗУ
(V)
    x_array[24]=x_array[8]=x_array[16]=x7;
    x_array[25]=x_array[9]=x_array[17]=x6;
    x_array[26]=x_array[10]=x_array[18]=x5;
    x_array[27]=x_array[11]=x_array[19]=x4;
    x_array[28]=x_array[12]=x_array[20]=x3;
    x_array[29]=x_array[13]=x_array[21]=x2;
    x_array[30]=x_array[14]=x_array[22]=x1;
    x_array[31]=x_array[15]=x_array[23]=x0;
    x_array[0]=x0;
    x_array[1]=x1;
    x_array[2]=x2;
    x_array[3]=x3;
    x_array[4]=x4;
    x_array[5]=x5;
    x_array[6]=x6;
    x_array[7]=x7;

//2. РАБОТА С Ш.Т
//определения длины (V)
crypt_length=full_length-5;//full_length*8-32-8;//от full_length
отнять 4 первых байта и последний

```

```

//определение числа блоков (по 64 бита) (V)
blocks=(crypt_length)/8;

//создание массивов (V)
    unsigned int *crypt_array = new unsigned int
[crypt_length]; //динамический массив под хранение кодов символов
ш.т+дополнения
    unsigned int *text_array = new unsigned int
[crypt_length]; //динамический массив под хранение кодов символов
о.т+дополнения

//заполнение массивов (V)
j=0;
for (i=0;i<crypt_length;i++)
{
    crypt_array[i]=((array_to_hide[j+0]<<7)&0x80)+
        ((array_to_hide[j+1]<<6)&0x40)+
        ((array_to_hide[j+2]<<5)&0x20)+
        ((array_to_hide[j+3]<<4)&0x10)+
        ((array_to_hide[j+4]<<3)&0x8)+
        ((array_to_hide[j+5]<<2)&0x4)+
        ((array_to_hide[j+6]<<1)&0x2)+
        ((array_to_hide[j+7])&0x1);
        j=j+8;
}
//3.РАСШИФРОВАНИЕ
//Для каждого блока..
for (block=0;block<blocks;block++) //начало поблочного цикла
{
    //определение начального заполнения N1 и N2 (перед первым
циклом)
    N1=(crypt_array[0+8*block]<<24)+(crypt_array[1+8*block]<<16)+(
crypt_array[2+8*block]<<8)+crypt_array[3+8*block]; // (V)
    N2=(crypt_array[4+8*block]<<24)+(crypt_array[5+8*block]<<16)+(
crypt_array[6+8*block]<<8)+crypt_array[7+8*block]; // (V)
    //циклы 1...32 32-Р
    for (round=0;round<32;round++)
    {
        //подсчет CM1 - сложение по модулю 2^32
        CM1=N1+x_array[round]; // (V)
        //заполнение CM1_array для подстановки (V)
        CM1_array[0]=(CM1&0xf0000000)>>28;
        CM1_array[1]=(CM1&0xf000000)>>24;
        CM1_array[2]=(CM1&0xf00000)>>20;
        CM1_array[3]=(CM1&0xf0000)>>16;
        CM1_array[4]=(CM1&0xf000)>>12;
    }
}

```

```

CM1_array[5]=(CM1&0xf00)>>8;
CM1_array[6]=(CM1&0xf0)>>4;
CM1_array[7]=CM1&0xf;
//произведение подстановки(V)
for (i=0;i<8;i++)//для каждой четверки бит CM1
{
    CM1_array[i]=tab[7-i][CM1_array[i]];//подстановка
}
//сборка массива CM1_array после подстановки в переменную
K(V)
    K=(CM1_array[0]<<28)+(CM1_array[1]<<24)+(CM1_array[2]<<20)+(CM1_array[3]<<16)+(CM1_array[4]<<12)+(CM1_array[5]<<8)+(CM1_array[6]<<4)+(CM1_array[7]);
    //циклический сдвиг на 11 влево (V)
    r=(K&0xffe00000)>>21;//старшая часть помещается в r и сдвигается в конец младший
    R=(K<<11)+r;//сдвиг числа на 11 влево + r
    //сложение по модулю 2 (XOR) (V)
    CM2=R^N2;
    // N1 -> N2, CM2 -> N1 (V)
    if (round!=31)//для всех циклов кроме последнего 32-го
    {
        N2=N1;
        N1=CM2;
    }
    else //для последнего цикла
    {
        N2=CM2;
    }

} //конец цикла 32-3
//запись N1,N2 в массив text_array
text_array[0+block*8]=(N1&0xff000000)>>24;
text_array[1+block*8]=(N1&0x00ff0000)>>16;
text_array[2+block*8]=(N1&0x0000ff00)>>8;
text_array[3+block*8]=(N1&0x000000ff);
text_array[4+block*8]=(N2&0xff000000)>>24;
text_array[5+block*8]=(N2&0x00ff0000)>>16;
text_array[6+block*8]=(N2&0x0000ff00)>>8;
text_array[7+block*8]=(N2&0x000000ff);
} //конец поблочного цикла

//4.ЗАПИСЬ В ФАЙЛ О.Т
//создание файла о.т
ofstream textstream("text.txt", ios_base::out | ios::binary
|ios_base::trunc);//создание выходного файла

```

```
for (i=0;i<crypt_length-dop_length;i++)
{
    char symbol= text_array[i];
    textstream.write((char*)&symbol,1);
}
textstream.close();
//КОНЕЦ
cout<<endl<<"Сообщение извлечено и расшифровано."<<endl;
system ("pause");
}
```

ПРИЛОЖЕНИЕ В

(обязательное)

ПРОТОКОЛ

проверки ВКР на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Программное обеспечение, реализующее шифрование и скрытие сообщений с помощью стеганографического метода» студента 6 курса Р. И. Свинкина направления подготовки 10.05.02 «Информационная безопасность телекоммуникационных систем». Специализация – разработка защищённых телекоммуникационных систем.

Скриншот отчёта о проверке ВКР в системе «Антиплагиат.ВУЗ» представлен на рисунке В1.

Руководитель ВКР: к.т.н., инженер I категории АО «ПНИЭИ» О.В. Кулагин.

Информация о документе:

Имя исходного файла: Свинкин_Р.И._11ПТ1_Диплом_отчет (тз текстом).docx
Имя компании: Пензенский гос. Университет
Тип документа: Прочее
Имя документа: Группа 11ПТ1 ВКР Свинкин РИ
Дата проверки: 29.12.2016 22:52
Модули поиска: Интернет (Антиплагиат), Диссертации и авторефераты РГБ, Модуль поиска ЭБС "Лань", Пензенский гос. Университет, Кольцо вузов, Университетская библиотека онлайн, Модуль поиска ЭБС БиблиоРоссика, Цитирования

Текстовые

статистики:

Индекс читаемости: обычный
Неизвестные слова: в пределах нормы
Макс. длина слова: в пределах нормы
Большие слова: в пределах нормы

Оригинальные блоки: 91,04%

Заимствованные блоки: 8,92%

Заимствование из "белых" источников: 0,05%

Итоговая оценка оригинальности: **91,08%**

Рисунок В1 – Скриншот отчёта о проверке ВКР в системе «Антиплагиат.ВУЗ»

«___» _____ 20__ г.

_____ О. В. Кулагин