Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 03 «Информационные системы и технологии»

Специализация «Издательско-полиграфический комплекс»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА**

по дисциплине «Защита информации и надежность информационных систем»

Тема «Программное средство для шифрования текста с размещением преобразованной информации в видеофайл»

Исполнитель

|  |  |  |
| --- | --- | --- |
| cтудент 3 курса группы 2    Руководитель |  | \_\_\_\_\_\_\_\_\_ Яськуть П.О.  подпись, дата |
| ассистент |  | \_\_\_\_\_\_\_\_\_\_\_\_ Сущеня А.А.  подпись, дата |
| Курсовой проект защищен с оценкой |  | \_\_\_\_\_\_\_\_\_ |
| Руководитель \_\_\_\_\_\_\_ |  | Сущеня А.А. |

подпись

Минск 2019

Содержание

[Введение 3](#_Toc8480145)

[1. Теоретическое введение 4](#_Toc8480146)

[1.1 Шифрование 4](#_Toc8480147)

[1.2 Стеганография 6](#_Toc8480148)

[2. Постановка задачи 7](#_Toc8480149)

[3. Описание метода 8](#_Toc8480150)

[3.1 Алгоритм шифрования DES 8](#_Toc8480151)

[3.2 Библиотека AForge.NET 11](#_Toc8480152)

[4. Описание программного средства 13](#_Toc8480153)

[5. Тестирование 18](#_Toc8480154)

[6. Руководство пользователя 21](#_Toc8480155)

[Заключение 24](#_Toc8480156)

[Список используемых источников 25](#_Toc8480157)

[Приложение 26](#_Toc8480158)

# Введение

Информация выступает сегодня как один из первостепенных ресурсов, значение которого не меньше, чем значение материальных, сырьевых и других ресурсов. Использование последних, в значительной степени, зависит именно от состояния и использования информации. В отличие от большинства ресурсов, которые способны истощаться, информационный потенциал может использоваться многократно как коллективами, так и отдельными работниками. При этом он постоянно увеличивается и обогащается.

Информационные технологии активно входят в современную жизнь, в том числе в организацию производственного процесса, деятельность которого невозможна без соответствующей системы управления. Динамичность современной деятельности требует обдуманного подхода к организации системы управления, повышение эффективности которого можно достигнуть при уместном и рациональном использовании системы информационных технологий. Правильный набор этих технологий, а также их логичное взаимодействие друг с другом, помогут предприятию грамотно выстроить свое функционирование.

С той же частотой, с которой информация появляется в повседневной жизни, осуществляются попытки её хищения, модификации и подобных действий, несущих за собой злые намерения.

Это всё, в частности, касается данных, передаваемых от отправителя к получателю. Всегда может найтись третье лицо, которое своими действиями может увидеть секретную информацию.

Задача защиты информации от несанкционированного доступа решалась во все времена на протяжении истории человечества. Уже в древнем мире выделилось два основных направления решения этой задачи, существующие и по сегодняшний день: криптография и стеганография. Целью криптографии является скрытие содержимого сообщений за счет их шифрования. В отличие от этого, при стеганографии скрывается сам факт существования тайного сообщения.

Целью курсового проекта является создание программного средства, позволяющего произвести защиту передаваемых данных, а также скрыть сам факт существования данных.

# Теоретическое введение

## **Шифрование**

Шифрование – это преобразование информации, делающее ее нечитаемой для посторонних. При этом  доверенные лица могут провести дешифрование и прочитать исходную информацию.

В наше время используются два вида шифрования:

* Симметричное шифрование использует один и тот же ключ и для зашифрования, и для расшифрования.
* Асимметричное шифрование использует два разных ключа: один для зашифрования (который также называется открытым), другой для расшифрования (называется закрытым).

Главная проблема использования одноключевых (симметричных) криптосистем заключается в распределении ключей. Для того, чтобы был возможен обмен информацией между двумя сторонами, ключ должен быть сгенерирован одной из них, а затем в конфиденциальном порядке передан другой. Особую остроту данная проблема приобрела в наши дни, когда криптография стала общедоступной, вследствие чего количество пользователей больших криптосистем может исчисляться сотнями и тысячами.

Начало асимметричным шифрам было положено в работе «Новые направления в современной криптографии» Уитфилда Диффи и Мартина Хеллмана, опубликованной в 1976 г. Находясь под влиянием работы Ральфа Меркла (Ralph Merkle) о распространении открытого ключа, они предложили метод получения секретных ключей для симметричного шифрования, используя открытый канал. В 2002 г. Хеллман предложил называть данный алгоритм «Диффи - Хеллмана - Меркла», признавая вклад Меркла в изобретение криптографии с открытым ключом.

Хотя работа Диффи-Хеллмана создала большой теоретический задел для открытой криптографии, первой реальной криптосистемой с открытым ключом считают алгоритм RSA (названный по имени авторов - Рон Ривест (Ronald Linn Rivest), Ади Шамир (Adi Shamir) и Леонард Адлеман (Leonard Adleman)).

Справедливости ради следует отметить, что в декабре 1997 г. была обнародована информация, согласно которой британский математик Клиффорд Кокс (Clifford Cocks), работавший в центре правительственной связи (GCHQ) Великобритании, описал систему, аналогичную RSA, в 1973 г., а несколькими месяцами позже в 1974 г. Малькольм Вильямсон изобрел математический алгоритм, аналогичный алгоритму Диффи – Хеллмана - Меркла.

Суть шифрования с открытым ключом заключается в том, что для шифрования данных используется один ключ, а для расшифрования другой (поэтому такие системы часто называют ассиметричными).

Основная предпосылка, которая привела к появлению шифрования с открытым ключом, заключалось в том, что отправитель сообщения (тот, кто зашифровывает сообщение), не обязательно должен быть способен его расшифровывать. Т.е. даже имея исходное сообщение, ключ, с помощью которого оно шифровалось, и зная алгоритм шифрования, он не может расшифровать закрытое сообщение без знания ключа расшифрования.

Первый ключ, которым шифруется исходное сообщение, называется открытым и может быть опубликован для использования всеми пользователями системы. Расшифрование с помощью этого ключа невозможно. Второй ключ, с помощью которого дешифруется сообщение, называется секретным (закрытым) и должен быть известен только законному получателю закрытого сообщения.

У криптографии с открытыми ключами есть ряд преимуществ перед симметричной криптографией. Наиболее полезное из них касается управления ключами (в частности, их выбором и рассылкой). В симметричных алгоритмах шифрования ключ зашифрования является также ключом расшифрования, следовательно, первый не может быть раскрыт. Это приводит к тому, что две легальные стороны (отправитель и получатель) договариваются заранее об алгоритме зашифрования и ключах либо при личной встрече, либо при передаче по абсолютно секретному каналу.

При использовании же криптосистем с открытым ключом стороны не обязаны встречаться, знать друг друга и иметь секретные каналы связи. Это преимущество становится еще более актуальным в случае большого количества пользователей системы. Тогда, к примеру, один пользователь может «закрыто» связаться с другим, взяв некоторую информацию (открытый ключ) из общедоступной базы данных (банка ключей).

Различие ключей (открытого и личного) в криптографии с открытыми ключами позволило создать следующие технологии: электронные цифровые подписи, распределенная проверка подлинности, согласование общего секретного ключа сессии, шифрование больших объемов данных без предварительного обмена общим секретным ключом.

В настоящее время хорошо известен целый ряд алгоритмов шифрования с открытым ключом. Алгоритмы криптосистемы с открытым ключом можно использовать в трёх назначениях:

* Как самостоятельные средства защиты передаваемых и хранимых данных.
* Как средства для распределения ключей. Алгоритмы - более трудоёмки, чем традиционные криптосистемы. Поэтому часто на практике рационально с помощью систем с открытым ключом распределять ключи, объем которых как информации незначителен. А потом с помощью обычных алгоритмов осуществлять обмен большими информационными потоками.
* Средства аутентификации пользователей (электронная цифровая подпись).

## **1.2 Стеганография**

Стеганография – это наука о скрытой передаче информации путем сохранения в тайне самого факта передачи. Этот термин ввел в 1499 году Иоганн Тритемий в своем трактате «Стеганография», зашифрованном под магическую книгу.

В отличие от криптографии, которая скрывает содержимое секретного сообщения, стеганография скрывает сам факт его существования. Как правило, сообщение будет выглядеть как что-либо иное, например, как изображение, статья, список покупок, письмо или судоку. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Преимущество стеганографии над чистой криптографией состоит в том, что сообщения не привлекают к себе внимания. Сообщения, факт шифрования которых не скрыт, вызывают подозрение и могут быть сами по себе уличающими в тех странах, в которых запрещена криптография. Таким образом, криптография защищает содержание сообщения, а стеганография защищает сам факт наличия каких-либо скрытых посланий.

# Постановка задачи

Функционально должны быть выполнены следующие задачи:

* Разработать алгоритм с использованием криптографического метода «DES» для создания зашифрованных документов со вставкой зашифрованной информации в видеофайл
* На основе разработанного алгоритма создать программное средство
* Составить руководство пользователя

# Описание метода

## **Алгоритм шифрования DES**

Одной из наиболее известных криптографических систем с закрытым ключом является DES – Data Encryption Standard. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы IBM и вступила в действие в США 1977 году. Алгоритм DES широко использовался при хранении и передаче данных между различными вычислительными системами; в почтовых системах, в электронных системах чертежей и при электронном обмене коммерческой информацией. Стандарт DES реализовывался как программно, так и аппаратно. Предприятиями разных стран был налажен массовый выпуск цифровых устройств, использующих DES для шифрования данных. Все устройства проходили обязательную сертификацию на соответствие стандарту.

Несмотря на то, что уже некоторое время эта система не имеет статуса государственного стандарта, она по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

Длина ключа в алгоритме DES составляет 56 бит. Именно с этим фактом связана основная полемика относительно способности DES противостоять различным атакам. Как известно, любой блочный шифр с закрытым ключом можно взломать, перебрав все возможные комбинации ключей. При длине ключа 56 бит возможны  разных ключей. Если компьютер перебирает за одну секунду 1 000 000 ключей (что примерно равно ), то на перебор всех   ключей потребуется  секунд или чуть более двух тысяч лет, что, конечно, является неприемлемым для злоумышленников.

Однако возможны более дорогие и быстрые вычислительные системы, чем персональный компьютер. Например, если иметь возможность объединить для проведения параллельных вычислений миллион процессоров, то максимальное время подбора ключа сокращается примерно до 18 часов. Это время не слишком велико, и крипто аналитик, оснащенный подобной дорогой техникой, вполне может выполнить вскрытие данных, зашифрованных DES за приемлемое для себя время.

Вместе с этим можно отметить, что систему DES вполне можно использовать в небольших и средних приложениях для шифрования данных, имеющих небольшую ценность. Для шифрования данных государственной важности или имеющих значительную коммерческую стоимость система DES в настоящее время не должна использоваться.

Основные параметры DES: размер блока 64 бита, длина ключа 56 бит, количество раундов – 16. DES является классической сетью Фейштеля с двумя ветвями. Алгоритм преобразует за несколько раундов 64-битный входной блок данных в 64-битный выходной блок. Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

Общая структура DES представлена на [рис. 1.1](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.1). Процесс шифрования каждого 64-битового блока исходных данных можно разделить на три этапа:

* начальная подготовка блока данных;
* 16 раундов «основного цикла»;
* конечная обработка блока данных.

На первом этапе 64-разрядный блок исходных данных подвергается начальной перестановке. При начальной перестановке биты блока данных определенным образом переупорядочиваются. Эта операция придает некоторую «хаотичность» исходному сообщению, снижая возможность использования крипто анализа статистическими методами.

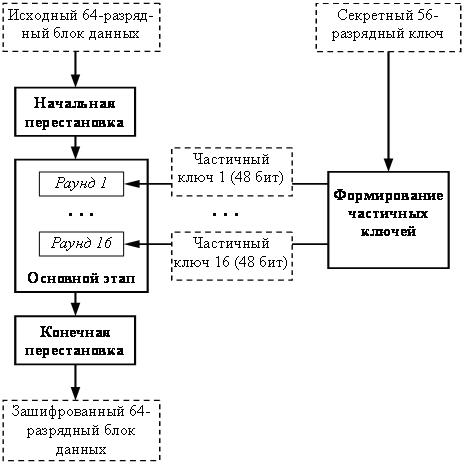


Рисунок 1.1 – Общая схема DES

Одновременно с начальной перестановкой блока данных выполняется начальная перестановка 56 бит ключа. Из [рисунка 1.1](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.1). видно, что в каждом из раундов используется соответствующий 48-битный частичный ключ . Ключи  получаются по определенному алгоритму, используя каждый из битов начального ключа по нескольку раз. В каждом раунде 56-битный ключ делится на две 28-битовые половинки. Затем половинки сдвигаются влево на один или два бита в зависимости от номера раунда. После сдвига определенным образом выбирается 48 из 56 битов. Так как при этом не только выбирается подмножество битов, но и изменяется их порядок, то эта операция называется "перестановка со сжатием". Ее результатом является набор из 48 битов. В среднем каждый бит исходного 56-битного ключа используется в 14 из 16 подключей, хотя не все биты используются одинаковое количество раз.

Далее выполняется основной цикл преобразования, организованный по сети Фейштеля и состоящий из 16 одинаковых раундов. При этом в каждом раунде ([рис. 1.2](https://www.intuit.ru/studies/courses/691/547/lecture/12377?page=1#image.4.2)) получается промежуточное 64-битное значение, которое затем обрабатывается в следующем раунде.

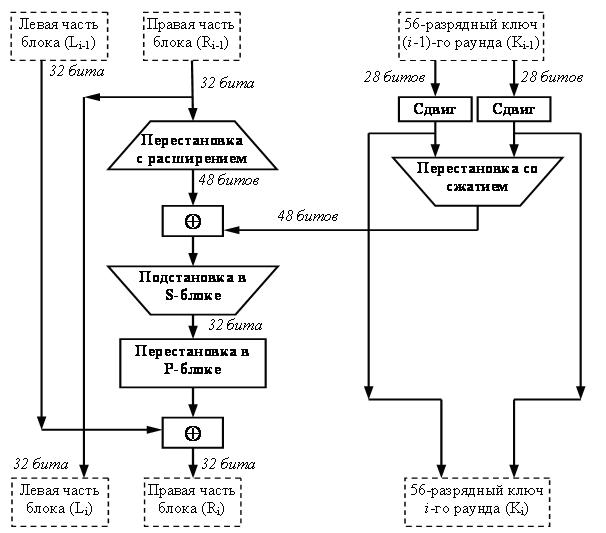


Рисунок 1.2 – Структура одного раунда DES

Левая и правая ветви каждого промежуточного значения обрабатываются как отдельные 32-битные значения, обозначенные L и R.

Вначале правая часть блока  расширяется до 48 битов, используя таблицу, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой половины в соответствие с размером ключа для выполнения операции XOR. Кроме того, за счет выполнения этой операции быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа (это называется «лавинным эффектом»). Чем сильнее проявляется лавинный эффект при использовании того или иного алгоритма шифрования, тем лучше.

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом . Затем полученное 48-битное значение подается на вход блока подстановки S, результатом которой является 32-битное значение. Подстановка выполняется в восьми блоках подстановки или восьми S-блоках. При выполнении этой операции 48 битов данных делятся на восемь 6-битовых подблоков, каждый из которых по своей таблице замен заменяется четырьмя битами. Подстановка с помощью S-блоков является одним из важнейших этапов DES. Таблицы замен для этой операции специально спроектированы специалистами так, чтобы обеспечивать максимальную безопасность. В результате выполнения этого этапа получаются восемь 4-битовых блоков, которые вновь объединяются в единое 32-битовое значение.

Полученное 32-битовое значение обрабатывается с помощью перестановки Р, которая не зависит от используемого ключа. Целью перестановки является максимальное переупорядочивание битов такое, чтобы в следующем раунде шифрования каждый бит с большой вероятностью обрабатывался другим S-блоком.

Результат перестановки объединяется с помощью операции XOR с левой половиной первоначального 64-битового блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд.

После шестнадцати раундов шифрования выполняется конечная перестановка результата. Эта перестановка инверсна (обратна) начальной перестановке.

После выполнения всех указанных шагов блок данных считается полностью зашифрованным и можно переходить к шифрованию следующего блока исходного сообщения.

Для зашифрования и расшифрования используется один и тот же алгоритм. При расшифровании на вход алгоритма подается зашифрованный текст. Единственное отличие состоит в обратном порядке использования частичных ключей .  используется на первом раунде,  — на последнем раунде.

После последнего раунда процесса расшифрования две половины выхода меняются местами так, чтобы вход заключительной перестановки был составлен из   и   . Выходом этой стадии является незашифрованный текст.

## **Библиотека AForge.NET**

AForge.NET — это библиотека с открытым исходным кодом, разработанная на языке C#, которая используется разработчиками и исследователями при решении задач, связанных с компьютерным изображением. А также в библиотеке AForge.NET есть возможности для решения задач в области искусственного интеллекта. Диапазон средств, применяемых библиотекой достаточно разнообразен: обработка изображений, нейронные сети, генетические алгоритмы, нечеткая логика, машинное обучение, робототехника и многое другое.

Основные компоненты библиотеки AForge.NET представлены ниже:

* AForge.Imaging — библиотека, предназначенная для работы с изображениями и фильтрами;
* AForge.Vision — библиотека, применяющая методы компьютерного зрения;
* AForge.Video — пакет библиотек для выполнения работ, связанных с видеоданными;
* AForge.Neuro — библиотека, в которой используются возможности нейронных сетей;
* AForge.Genetic — библиотека, предназначенная для решения разнообразных задач с применением генетических алгоритмов;
* AForge.Fuzzy — библиотека работает с нечеткой логикой;
* AForge.Robotics — библиотека поддерживает методы, используемые в области робототехники;
* AForge.MachineLearning — библиотека, в которой применяются элементы машинного обучения.

AForge.Video.VFW включает в себя классы AviWriter и AviReader. Класс AviWriter позволяет создавать новые AVI-файлы, класс AviReader позволяет работать с существующими AVI-файлами.

Библиотека AForge.NET находится в процессе постоянного улучшения и развития. По данной библиотеке существует много примеров, показывающих ее работу, а также есть html-документация, которая может понадобиться начинающим разработчикам, которые захотят использовать данную библиотеку в своих проектах.

# Описание программного средства

Программное средство «AVIHide» разработано в программе Visual Studio 2017 на языке C# с применением технологии Windows Presentation Foundation и библиотеки AForge.NET.

Программа «AVIHide» выполняет поставленные задачи, а именно:

* шифрование сообщения – файл формата txt, с помощью алгоритма DES и последующее помещение зашифрованного сообщения в контейнер – файл формата avi;
* чтение зашифрованных сообщений из файлов формата avi.

Для шифрования данных с помощью программы «AVIHide» необходимо указать 2 файла: текстовый документ формата txt и видеофайл формата avi. Данные из текстового документа шифруются при помощи алгоритма DES и создается новый видеофайл с использованием библиотек AForge.Video на основе выбранного видео. В зависимости от размера сообщения, байты сообщения заносятся в один или несколько кадров нового видео при помощи метода LSB, а также создается header файл, в котором указывается длинна вводимого сообщения, расширение и LSB метод, после чего данный header файл так же прячется в кадрах нового видео. Новое видео визуально не отличается от указанного вначале, но уже содержит зашифрованное сообщение. Алгоритм программы указан на рисунке 4.1.

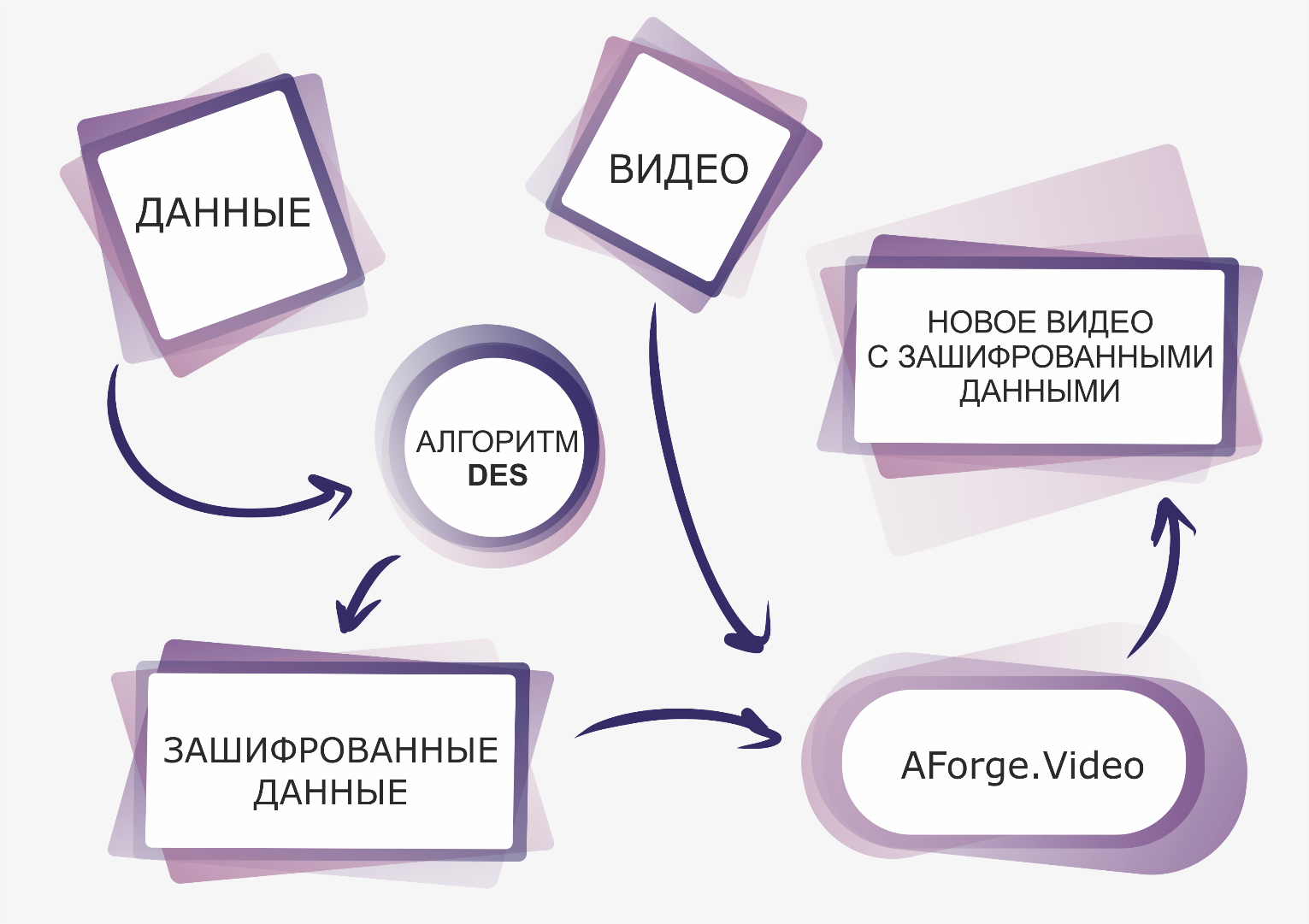


Рисунок 4.1 – Алгоритм программы при шифровании

На рисунке 4.2. представлена функция «Encrypt» из файла DES.сs. Данная функция принимает 3 аргумента – сообщение, ключ шифрования для алгоритма DES и переменную для нового ключа. Параметр по ссылке необходим для получения ключа необходимого формата, в случае если пользователь ввел ключ меньшей/большей длины, чем необходимо для алгоритма DES. Результат функции является зашифрованное сообщение.

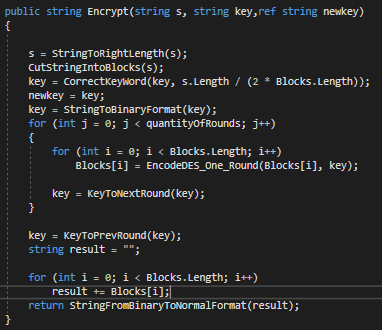


Рисунок 4.2 – Функция шифрования «Encrypt»

На рисунке 4.3. предоставлен листинг программы из файла Engine.cs. Здесь, с помощью библиотеки AForge.Video создается новое видео, где указывается 5 параметров – исходный видеофайл, путь для нового видеофайла, длина зашифрованного сообщения, расширение, метод LSB, а также пароль, используемый в алгоритме функции RandIndex, которая в свою очередь отвечает за номер кадра, в котором будет произведена замена байтов. Далее в цикле идет проверка длины зашифрованного сообщения, если размер сообщения в байтах больше, чем количество байтов в кадре, то зашифрованное сообщение будет размещено в нескольких кадрах, если же меньше, то только в одном. Вставка зашифрованных данных в новый видеофайл осуществляется с помощью функции InsertToFrame – она принимает один аргумент – байты кадра.

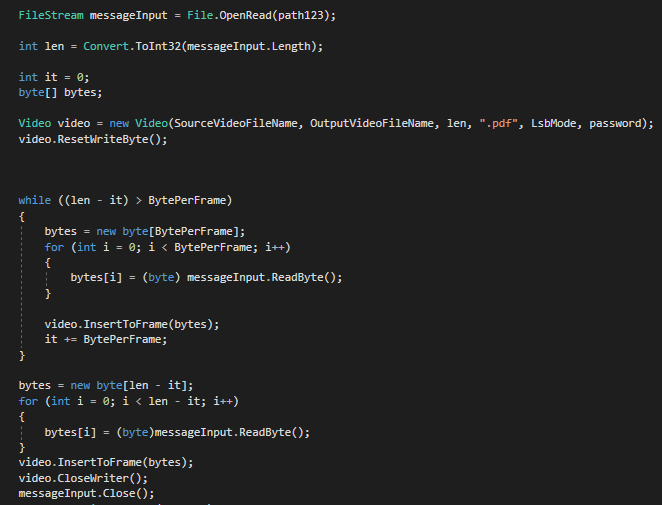


Рисунок 4.3 – Вставка байтов зашифрованного сообщения в кадр

На рисунке 4.4 изображена функция HideHeader файла Video.cs. Данная функция создает header файл, куда записывается длина зашифрованного сообщения, расширение и метод LSB. После чего данный файл представляется в виде байтов и заносится в кадр нового видео.

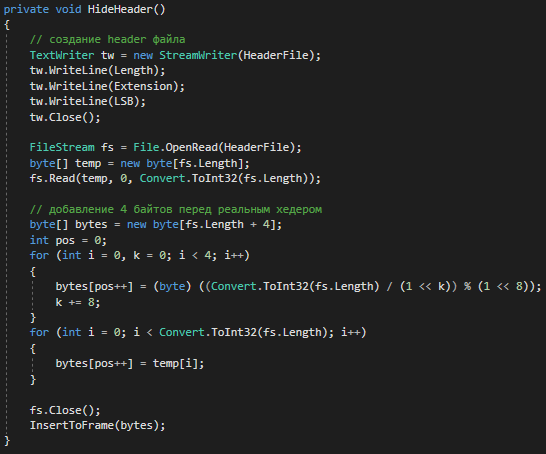


Рисунок 4.4 – Функция сокрытия заголовочного файла «HideHeader»

Извлечение данных из видео осуществляется по обратному алгоритму (рис. 4.5). Из видео, где содержится зашифрованное сообщение извлекается заголовочный файл. С помощью заголовочного файла определяется длина зашифрованного сообщения и извлекаются байты из определенных кадров выбранного видео и записываются в текстовый файл. Последней операцией является расшифрование извлеченного сообщения.

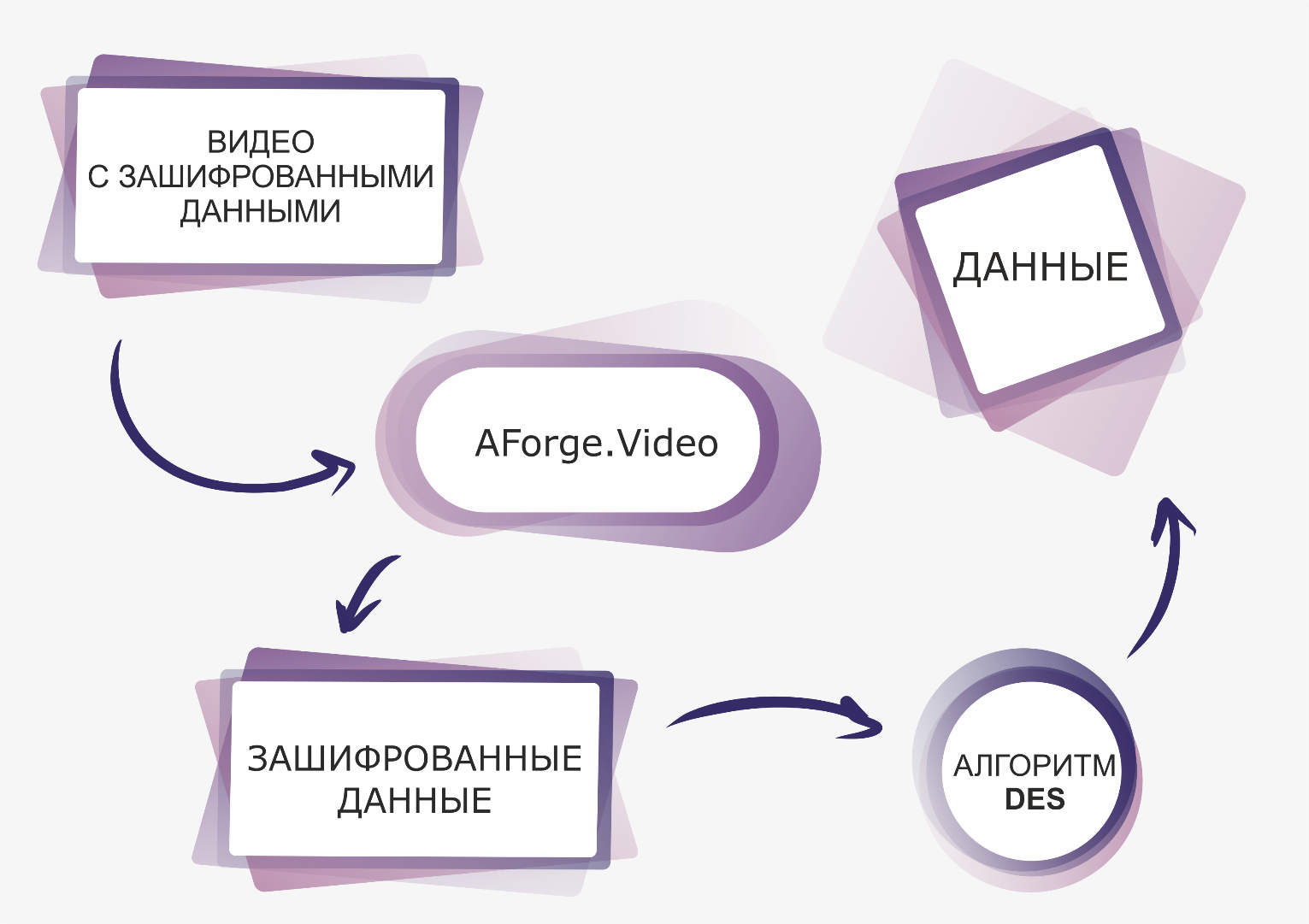


Рисунок 4.5 – Алгоритм программы при шифровании

На рисунке 4.6 изображена часть функции ExtractHeader файла Video.cs. Данная функция считывает из определенного кадра заголовочный файл, с помощью которого определяет длину зашифрованного сообщения.

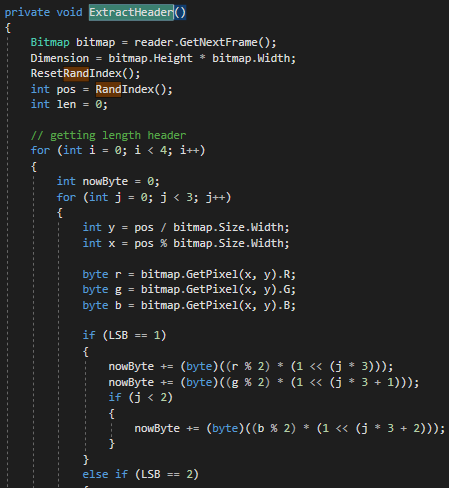


Рисунок 4.6 – Функция извлечения заголовочного файла «ExtractHeader»

На рисунке 4.7 изображена функция DecryptAndSave файла Engine.cs. Данная функция создает временный файл и записывает в него считанные байты из кадров видео. После чего данный файл расшифровывается при помощи функции Decrypt и в текстовый файл записывается сообщение, которое было спрятано в указанном видеофайле.

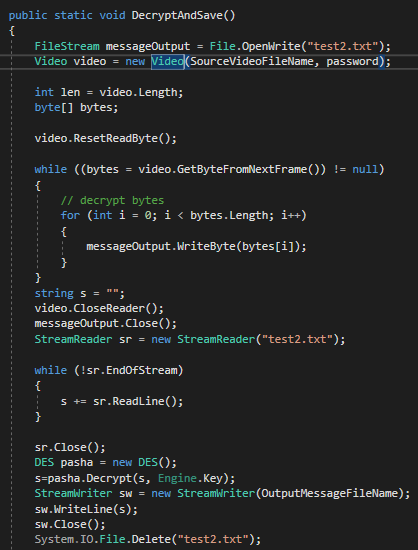


Рисунок 4.7 – Функция расшифрования и сохранения сообщения в файл «DecryptAndSave»

## **Тестирование**

Тестирование программного обеспечения — процесс исследования, испытания [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом.

Испытания проводятся с целью проверки правильности функционирования программы и выявления имеющихся недостатков.

Для проведения испытания программы необходимо запустить программу и проверить работу всех вариантов использования приложения. Разработанное приложение может использоваться на персональном компьютере с операционной системой Windows и установленным .NET Framework 4.5.

Для того, чтобы приложение корректно работало, в ходе разработки были обработаны исключения, чтобы пользователь был уведомлен о том, что он сделал не так.

Если все поля требуемые заполнения будут пусты, то приложение оповестит вас об этом окном с сообщением об ошибке, а также звуком, соответствующим ошибке на вашем ПК (рис. 5.1).

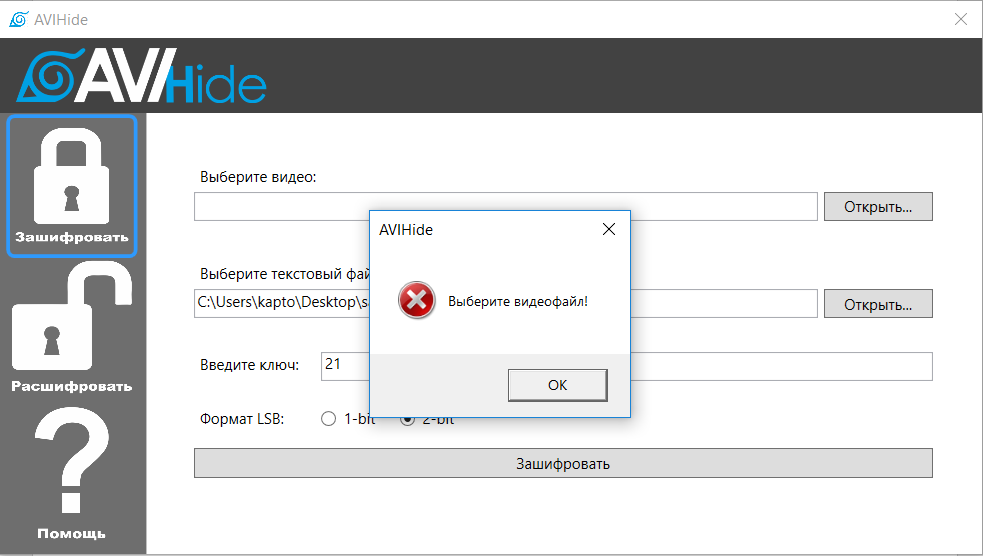


Рисунок 5.1 – Ошибка при пустых TextBox

Приложение также выдаст сообщение об ошибке, если путь текстового файла или видеофайла ссылается на несуществующий документ (рис. 5.2).

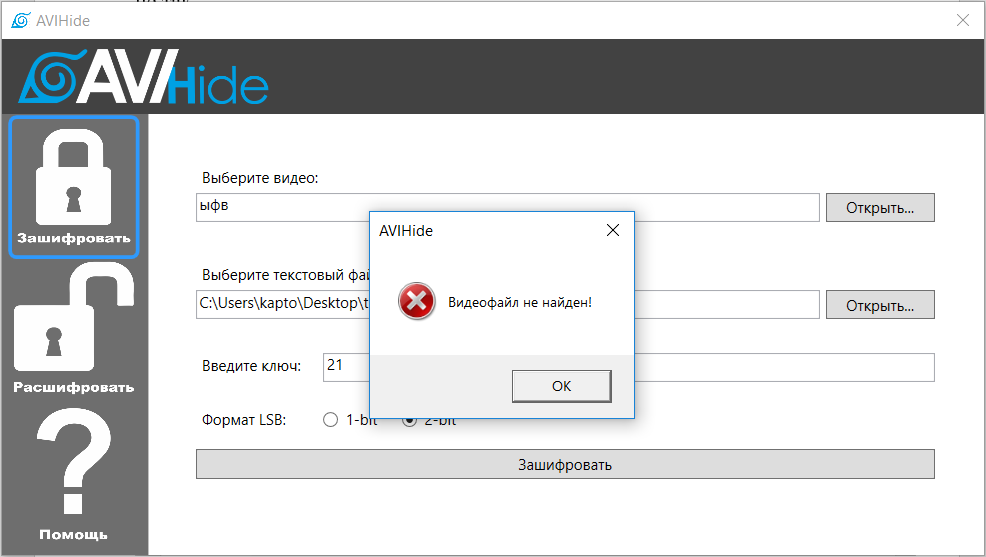


Рисунок 5.2 – Ошибка при вводе ошибочных данных

Алгоритм программы предусматривает автоматическую коррекцию ключа для шифрования алгоритмом DES. Например, если ключ введен в неверном формате, программа после шифрования сообщения автоматически выдаст нужный ключ, в нужном формате, для расшифровки сообщения (рис 5.3.).

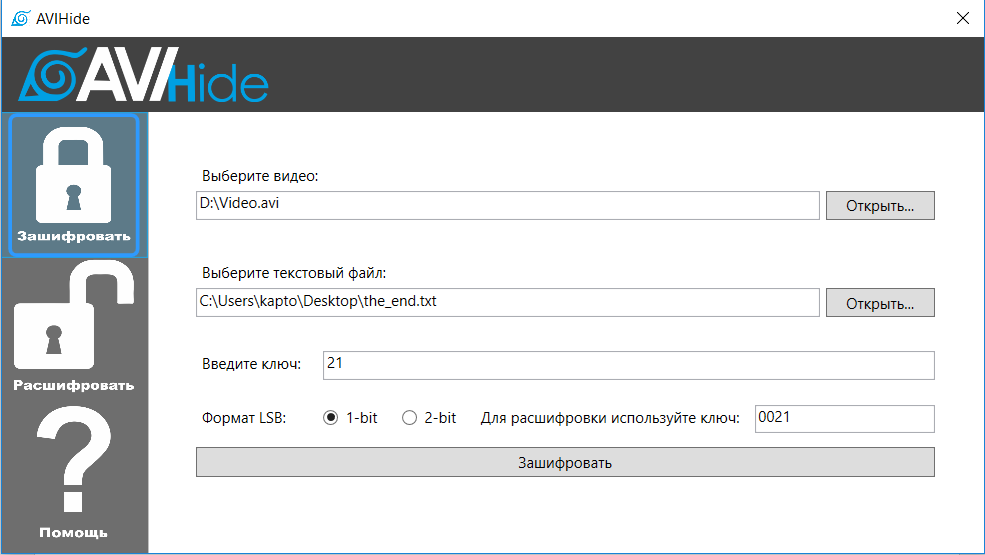


Рисунок 5.3 – Автоматическая коррекция ключа для алгоритма DES

Все описанные выше возможные исключения были обработаны для правильной работы приложения, а также для того, чтобы пользователь смог понять, что пошло не так, если произошла какая-то ошибка.

## **Руководство пользователя**

Для того чтобы использовать разработанное в рамках данного курсового проекта приложение «AVIHide» необходимо иметь на компьютере .NET Framework 4.5. Внешний вид приложения после запуска предоставлен на рисунке 6.1.

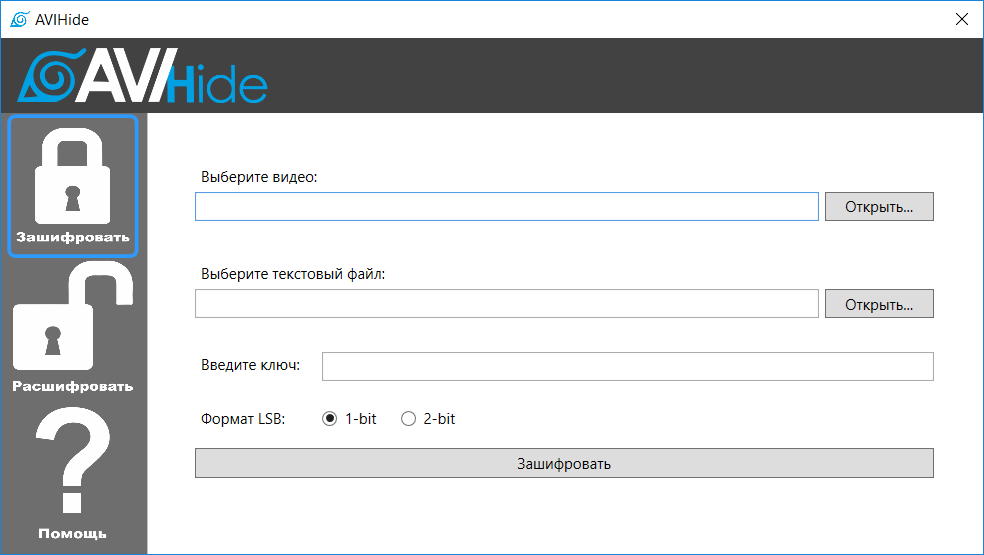


Рисунок 6.1 – Окно программы при запуске приложения

Если пользователь впервые работает с программой «AVIHide» или у него возникли какие-либо вопросы по работе с приложением, он может в любой момент нажать на кнопку «Помощь» и получить полную информацию по работе с «AVIHide» ( рис. 6.2. ).

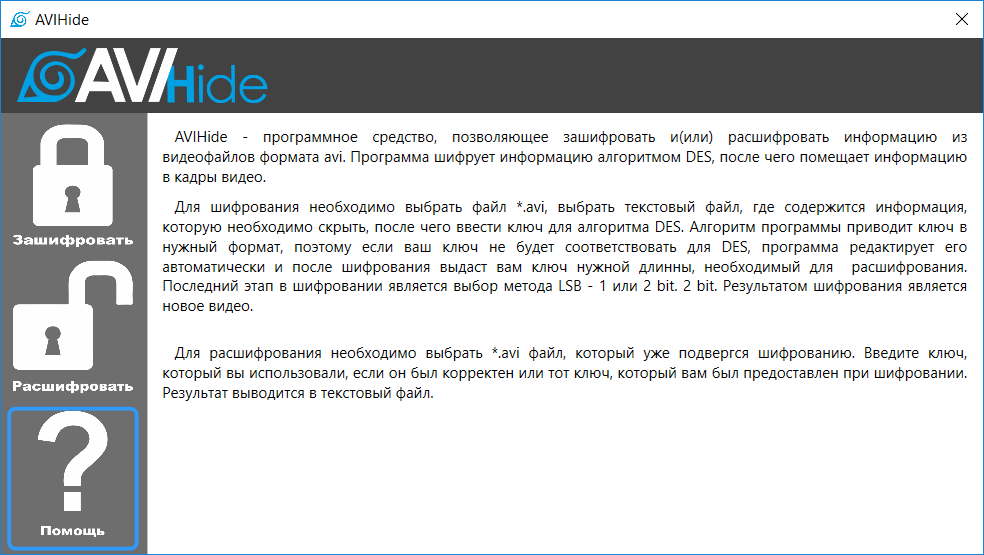


Рисунок 6.2 – Окно программы на вкладке «Помощь»

Для шифрования и скрытия сообщения в видео, необходимо выбрать: текстовый файл, где будет содержаться само сообщение; видеофайл, в который необходимо спрятать зашифрованное сообщение; ввести ключ в десятичном формате для алгоритма DES; выбрать метод LSB: 1 или 2 bit. После чего необходимо нажать на кнопку «Зашифровать» (рис. 6.3).

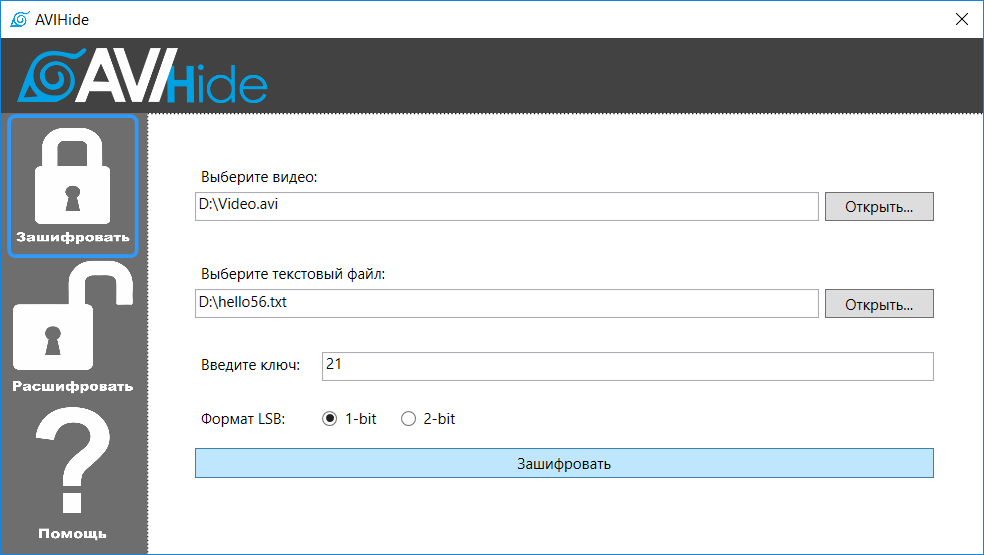


Рисунок 6.3 – Окно программы на вкладке «Зашифровать»

После нажатия на кнопку «Зашифровать», пользователю открывается окно, где он должен выбрать путь для видеофайла, в котором будет зашифрованное хранится сообщение (рис 6.4).

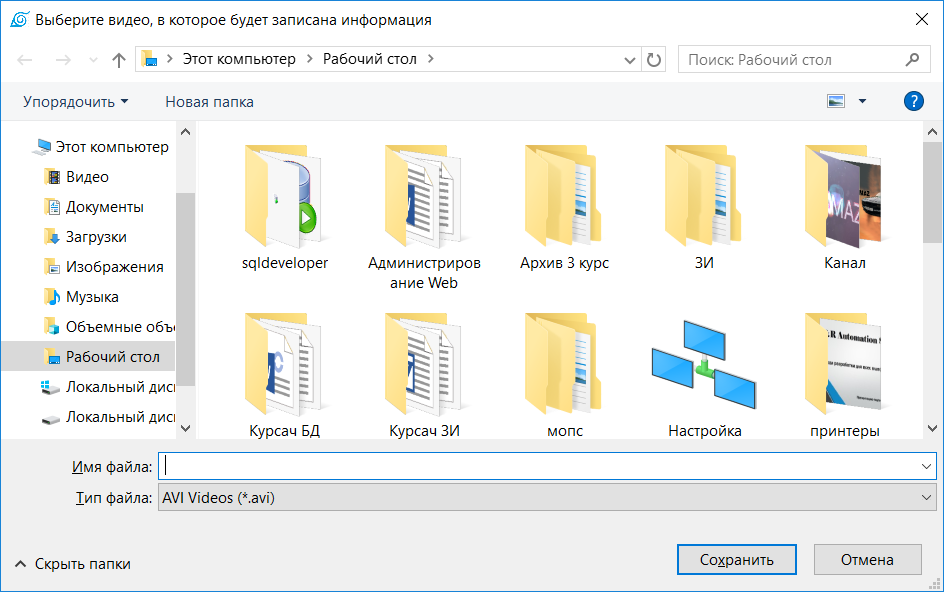


Рисунок 6.4 – Выбор пути для нового видеофайла

Для расшифрования необходимо указать видеофайл, в котором хранится зашифрованное сообщение и ключ для алгоритма DES. После чего нажать на кнопку «Расшифровать» (рис 6.5).

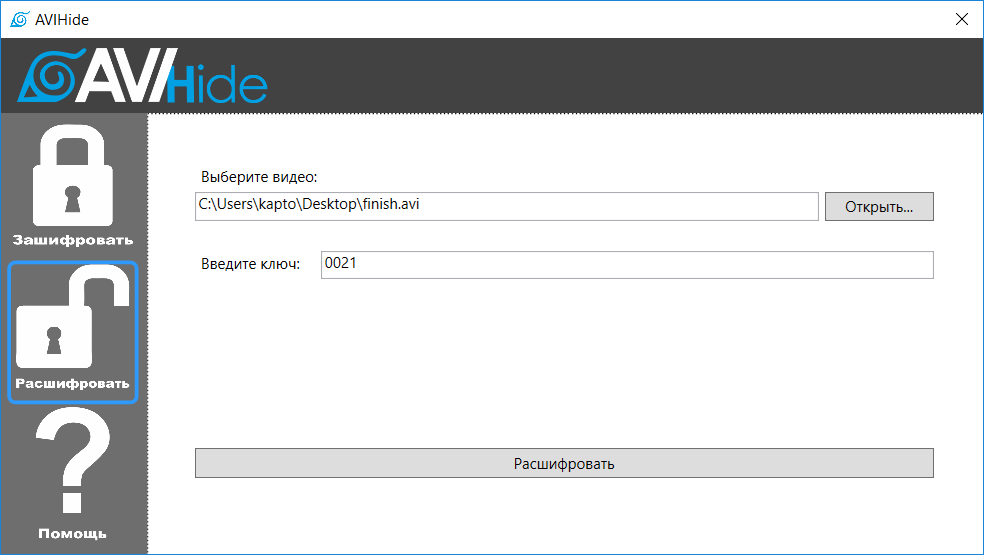


Рисунок 6.5 – Окно программы на вкладке расшифровать

После нажатия на кнопку «Расшифровать», пользователю открывается окно, где он должен выбрать путь для нового текстового файла, в который будет помещена скрытая информация (рис 6.6).

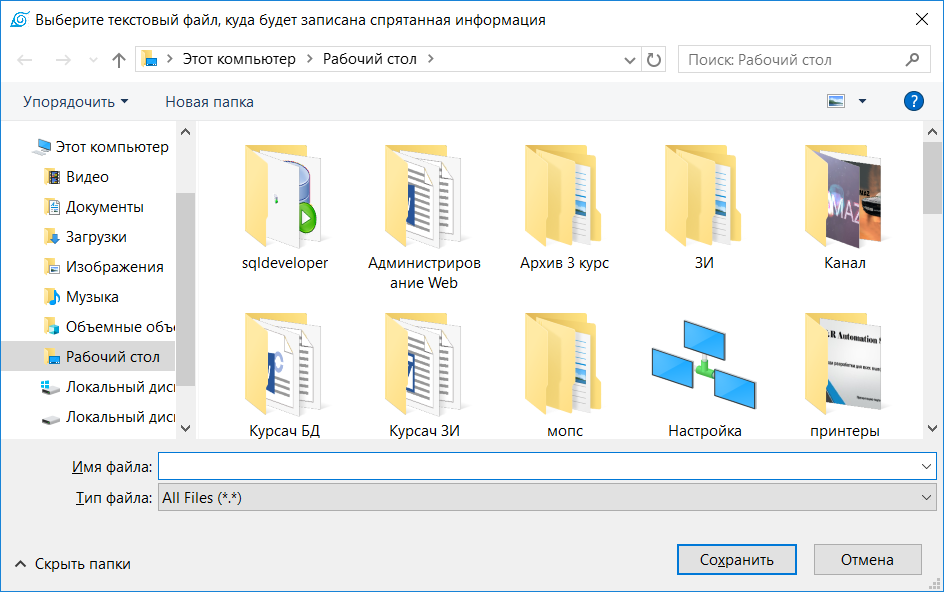


Рисунок 6.4 – Выбор пути для текстового документа с данными

# Заключение

В качестве результата данного курсового проекта можно представить приложение защиты данных «AVIHide». Данная программа позволяет зашифровать и скрыть наличие зашифрованного сообщения, а также получить зашифрованное сообщение из видеофайла. Приложение полностью удовлетворяет требованиям курсового проекта.

Для создания данного проекта были проанализированы различные алгоритмы шифрования, к числу которых относится и используемый в курсовом проекте алгоритм DES, было проведено сравнение существующих алгоритмов с алгоритмом, используемым в разрабатываемом приложении.

И хотя старый американский стандарт DES в настоящее время утратил свой официальный статус, этот алгоритм все же заслуживает внимания при изучении криптографии.

Алгоритм DES может уступать в некоторых характеристиках другим алгоритмам, но в общем целом является вполне надежным алгоритмом для шифрования данных. Чтобы обеспечить еще большую криптостойкость алгоритму программы «AVIHide», можно добавить пароль при шифровании и расшифровании сообщения – данный пароль будет участвовать в алгоритме выборки кадров для шифрования и расшифрования заголовочного файла и файла сообщения. В текущей версии программы данная переменная является константой.

Приложение является десктопным и разработано на языке C# с использованием технологии WPF и библиотек AForge.NET.

Технология WPF была выбрана неслучайно, так как пользователь в первую очередь оценивает внешний вид приложения, поэтому необходимо было разработать привлекательный графически интерфейса, для создания которого на программной платформе .NET идеально подходит WPF.

Программа «AVIHide» может быть доработана добавлением новых контейнеров для сообщений – картинок, аудиофайлов, zip-архивов и т.д., а так же добавлением различных алгоритмов шифрования, таких как 3DES, AES.

Список используемых источников

1. Рихтер Д. Программирование на платформе Microsoft .NET Framework, СПБ.: Русская редакция, 2005. – 512 с.
2. Шилдт Г. Полный справочник по C# Вильямс, 2004. – 752 с.
3. Урбанович П. П., Шиман Д. В. Защита информации и надежность информационных систем: учебно-метод. Пособие, Минск, БГТУ, 2014. – 90 с.
4. Урбанович П. П. Защита информации методами криптографии, стеганографии и обфускации, Минск, БГТУ, 2016 – 188 с.
5. Википедия [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org> – Дата доступа: 21.03.2019.
6. Руководство по WPF [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/> – Дата доступа: 20.03.2019.
7. Библиотека AForge.NET [Электронный ресурс] – Режим доступа: <http://www.aforgenet.com/framework/> – Дата доступа: 15.03.2019.

# Приложение

Avi.cs

using System;

using System.Drawing;

using System.Runtime.InteropServices;

namespace AviHide

{

public class Avi

{

public const int StreamtypeVIDEO = 1935960438; //mmioStringToFOURCC("vids", 0)

public const int OF\_SHARE\_DENY\_WRITE = 32;

public const int BMP\_MAGIC\_COOKIE = 19778; //ascii string "BM"

#region structure declarations

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct RECT

{

public UInt32 left;

public UInt32 top;

public UInt32 right;

public UInt32 bottom;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct BITMAPINFOHEADER

{

public UInt32 biSize;

public Int32 biWidth;

public Int32 biHeight;

public Int16 biPlanes;

public Int16 biBitCount;

public UInt32 biCompression;

public UInt32 biSizeImage;

public Int32 biXPelsPerMeter;

public Int32 biYPelsPerMeter;

public UInt32 biClrUsed;

public UInt32 biClrImportant;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct AVISTREAMINFO

{

public UInt32 fccType;

public UInt32 fccHandler;

public UInt32 dwFlags;

public UInt32 dwCaps;

public UInt16 wPriority;

public UInt16 wLanguage;

public UInt32 dwScale;

public UInt32 dwRate;

public UInt32 dwStart;

public UInt32 dwLength;

public UInt32 dwInitialFrames;

public UInt32 dwSuggestedBufferSize;

public UInt32 dwQuality;

public UInt32 dwSampleSize;

public RECT rcFrame;

public UInt32 dwEditCount;

public UInt32 dwFormatChangeCount;

[MarshalAs(UnmanagedType.ByValArray, SizeConst = 64)]

public UInt16[] szName;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct BITMAPFILEHEADER

{

public Int16 bfType; //"magic cookie" - must be "BM"

public Int32 bfSize;

public Int16 bfReserved1;

public Int16 bfReserved2;

public Int32 bfOffBits;

}

#endregion structure declarations

#region method declarations

//Initialize the AVI library

[DllImport("avifil32.dll")]

public static extern void AVIFileInit();

//Open an AVI file

[DllImport("avifil32.dll", PreserveSig = true)]

public static extern int AVIFileOpen(

ref int ppfile,

String szFile,

int uMode,

int pclsidHandler);

//Get a stream from an open AVI file

[DllImport("avifil32.dll")]

public static extern int AVIFileGetStream(

int pfile,

out IntPtr ppavi,

int fccType,

int lParam);

//Get the start position of a stream

[DllImport("avifil32.dll", PreserveSig = true)]

public static extern int AVIStreamStart(int pavi);

//Get the length of a stream in frames

[DllImport("avifil32.dll", PreserveSig = true)]

public static extern int AVIStreamLength(int pavi);

//Get information about an open stream

[DllImport("avifil32.dll")]

public static extern int AVIStreamInfo(

int pAVIStream,

ref AVISTREAMINFO psi,

int lSize);

//Get a pointer to a GETFRAME object (returns 0 on error)

[DllImport("avifil32.dll")]

public static extern int AVIStreamGetFrameOpen(

IntPtr pAVIStream,

ref BITMAPINFOHEADER bih);

/\*[DllImport("avifil32.dll")]

public static extern int AVIStreamGetFrameOpen(

IntPtr pAVIStream,

int dummy);\*/

//Get a pointer to a packed DIB (returns 0 on error)

[DllImport("avifil32.dll")]

public static extern int AVIStreamGetFrame(

int pGetFrameObj,

int lPos);

//Create a new stream in an open AVI file

[DllImport("avifil32.dll")]

public static extern int AVIFileCreateStream(

int pfile,

out IntPtr ppavi,

ref AVISTREAMINFO ptr\_streaminfo);

//Set the format for a new stream

[DllImport("avifil32.dll")]

public static extern int AVIStreamSetFormat(

IntPtr aviStream, Int32 lPos,

ref BITMAPINFOHEADER lpFormat, Int32 cbFormat);

//Write a sample to a stream

[DllImport("avifil32.dll")]

public static extern int AVIStreamWrite(

IntPtr aviStream, Int32 lStart, Int32 lSamples,

IntPtr lpBuffer, Int32 cbBuffer, Int32 dwFlags,

Int32 dummy1, Int32 dummy2);

//Release the GETFRAME object

[DllImport("avifil32.dll")]

public static extern int AVIStreamGetFrameClose(

int pGetFrameObj);

//Release an open AVI stream

[DllImport("avifil32.dll")]

public static extern int AVIStreamRelease(IntPtr aviStream);

//Release an open AVI file

[DllImport("avifil32.dll")]

public static extern int AVIFileRelease(int pfile);

//Close the AVI library

[DllImport("avifil32.dll")]

public static extern void AVIFileExit();

#endregion methos declarations

}

}

DES.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AviHide

{

public class DES

{

private const int sizeOfBlock = 128; //в DES размер блока 64 бит, но поскольку в unicode символ в два раза длинее, то увеличим блок тоже в два раза

private const int sizeOfChar = 16; //размер одного символа (in Unicode 16 bit)

private const int shiftKey = 2; //сдвиг ключа

private const int quantityOfRounds = 16; //количество раундов

string[] Blocks; //сами блоки в двоичном формате

string[] check;

/\*Метод, доводящий строку до такого размера, чтобы она делилась на sizeOfBlock.

\* Размер увеличивается с помощью добавления к исходной строке символа “решетка”.\*/

private string StringToRightLength(string input)

{

while (((input.Length \* sizeOfChar) % sizeOfBlock) != 0)

input += "#";

return input;

}

//Метод, разбивающий строку в обычном (символьном) формате на блоки.

private void CutStringIntoBlocks(string input)

{

Blocks = new string[(input.Length \* sizeOfChar) / sizeOfBlock];

int lengthOfBlock = input.Length / Blocks.Length;

for (int i = 0; i < Blocks.Length; i++)

{

Blocks[i] = input.Substring(i \* lengthOfBlock, lengthOfBlock);

Blocks[i] = StringToBinaryFormat(Blocks[i]);

}

}

//Метод, разбивающий строку в двоичном формате на блоки.

private void CutBinaryStringIntoBlocks(string input)

{

Blocks = new string[input.Length / sizeOfBlock];

int lengthOfBlock = input.Length / Blocks.Length;

for (int i = 0; i < Blocks.Length; i++)

Blocks[i] = input.Substring(i \* lengthOfBlock, lengthOfBlock);

}

//Метод, переводящий строку в двоичный формат.

private string StringToBinaryFormat(string input)

{

string output = "";

for (int i = 0; i < input.Length; i++)

{

string char\_binary = Convert.ToString(input[i], 2);

while (char\_binary.Length < sizeOfChar)

char\_binary = "0" + char\_binary;

output += char\_binary;

}

return output;

}

//Метод, доводящий длину ключа до нужной длины.

private string CorrectKeyWord(string input, int lengthKey)

{

if (input.Length > lengthKey)

input = input.Substring(0, lengthKey);

else

while (input.Length < lengthKey)

input = "0" + input;

return input;

}

//Один раунд шифрования алгоритмом DES.

private string EncodeDES\_One\_Round(string input, string key)

{

string L = input.Substring(0, input.Length / 2);

string R = input.Substring(input.Length / 2, input.Length / 2);

return (R + XOR(L, f(R, key)));

}

//Один раунд дешифрования алгоритмом DES.

private string DecodeDES\_One\_Round(string input, string key)

{

string L = input.Substring(0, input.Length / 2);

string R = input.Substring(input.Length / 2, input.Length / 2);

return (XOR(f(L, key), R) + L);

}

//XOR двух строк с двоичными данными.

private string XOR(string s1, string s2)

{

string result = "";

for (int i = 0; i < s1.Length; i++)

{

bool a = Convert.ToBoolean(Convert.ToInt32(s1[i].ToString()));

bool b = Convert.ToBoolean(Convert.ToInt32(s2[i].ToString()));

if (a ^ b)

result += "1";

else

result += "0";

}

return result;

}

//Шифрующая функция f.

private string f(string s1, string s2)

{

return XOR(s1, s2);

}

//Вычисление ключа для следующего раунда шифрования DES. Циклический сдвиг >> shiftKey.

private string KeyToNextRound(string key)

{

for (int i = 0; i < shiftKey; i++)

{

key = key[key.Length - 1] + key;

key = key.Remove(key.Length - 1);

}

return key;

}

//Вычисление ключа для следующего раунда расшифровки DES. циклический сдвиг << shiftKey.

private string KeyToPrevRound(string key)

{

for (int i = 0; i < shiftKey; i++)

{

key = key + key[0];

key = key.Remove(0, 1);

}

return key;

}

//Метод, переводящий строку с двоичными данными в символьный формат.

private string StringFromBinaryToNormalFormat(string input)

{

string output = "";

while (input.Length > 0)

{

string char\_binary = input.Substring(0, sizeOfChar);

input = input.Remove(0, sizeOfChar);

int a = 0;

int degree = char\_binary.Length - 1;

foreach (char c in char\_binary)

a += Convert.ToInt32(c.ToString()) \* (int)Math.Pow(2, degree--);

output += ((char)a).ToString();

}

return output;

}

public string Encrypt(string s, string key,ref string newkey)

{

s = StringToRightLength(s);

CutStringIntoBlocks(s);

key = CorrectKeyWord(key, s.Length / (2 \* Blocks.Length));

newkey = key;

key = StringToBinaryFormat(key);

for (int j = 0; j < quantityOfRounds; j++)

{

for (int i = 0; i < Blocks.Length; i++)

Blocks[i] = EncodeDES\_One\_Round(Blocks[i], key);

key = KeyToNextRound(key);

}

key = KeyToPrevRound(key);

string result = "";

for (int i = 0; i < Blocks.Length; i++)

result += Blocks[i];

return StringFromBinaryToNormalFormat(result);

}

public string Decrypt(string s, string key)

{

key = StringToBinaryFormat(key);

for (int j = 0; j < quantityOfRounds; j++)

{

key = KeyToNextRound(key);

}

key = KeyToPrevRound(key);

s = StringToBinaryFormat(s);

CutBinaryStringIntoBlocks(s);

for (int j = 0; j < quantityOfRounds; j++)

{

for (int i = 0; i < Blocks.Length; i++)

Blocks[i] = DecodeDES\_One\_Round(Blocks[i], key);

key = KeyToPrevRound(key);

}

key = KeyToNextRound(key);

string result = "";

for (int i = 0; i < Blocks.Length; i++)

result += Blocks[i];

result = StringFromBinaryToNormalFormat(result);

result = result.TrimEnd('#');

return result;

}

}

}

Engine.cs

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Diagnostics;

namespace AviHide

{

public class Engine

{

public static string SourceMessageFileName { get; set; }

public static string SourceVideoFileName { get; set; }

public static string OutputMessageFileName { get; set; }

public static string OutputVideoFileName { get; set; }

public static string Key { get; set; }

public static double PNSR { get; set; }

public static int LsbMode { get; set; }

public static string newkey;

public static string password = "21";

// TODO

public static int BytePerFrame = 1024;

public static void EncryptAndSave()

{

string path123 = "text1.txt"; DES pasha = new DES();

FileStream stream = new FileStream(SourceMessageFileName, FileMode.Open);

StreamReader reader = new StreamReader(stream);

string str = reader.ReadToEnd();

stream.Close();

str=pasha.Encrypt(str, Engine.Key,ref newkey);

using (StreamWriter sw = File.CreateText(path123))

{

sw.WriteLine(str);

}

FileStream messageInput = File.OpenRead(path123);

int len = Convert.ToInt32(messageInput.Length);

int it = 0;

byte[] bytes;

Video video = new Video(SourceVideoFileName, OutputVideoFileName, len, ".pdf", LsbMode, password);

video.ResetWriteByte();

while ((len - it) > BytePerFrame)

{

bytes = new byte[BytePerFrame];

for (int i = 0; i < BytePerFrame; i++)

{

bytes[i] = (byte) messageInput.ReadByte();

}

video.InsertToFrame(bytes);

it += BytePerFrame;

}

bytes = new byte[len - it];

for (int i = 0; i < len - it; i++)

{

bytes[i] = (byte)messageInput.ReadByte();

}

video.InsertToFrame(bytes);

video.CloseWriter();

messageInput.Close();

System.IO.File.Delete(path123);

}

public static void DecryptAndSave()

{

FileStream messageOutput = File.OpenWrite("test2.txt");

Video video = new Video(SourceVideoFileName, password);

int len = video.Length;

byte[] bytes;

video.ResetReadByte();

while ((bytes = video.GetByteFromNextFrame()) != null)

{

// decrypt bytes

for (int i = 0; i < bytes.Length; i++)

{

messageOutput.WriteByte(bytes[i]);

}

}

string s = "";

video.CloseReader();

messageOutput.Close();

StreamReader sr = new StreamReader("test2.txt");

while (!sr.EndOfStream)

{

s += sr.ReadLine();

}

sr.Close();

DES pasha = new DES();

s=pasha.Decrypt(s, Engine.Key);

StreamWriter sw = new StreamWriter(OutputMessageFileName);

sw.WriteLine(s);

sw.Close();

System.IO.File.Delete("test2.txt");

}

}

}

Video.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.IO;

using AForge.Video;

using AForge.Video.VFW;

namespace AviHide

{

class Video

{

public int Length { get; private set; }

public String Extension { get; private set; }

public int LSB { get; private set; }

public int Password { get; private set; }

public double PNSR { get; private set; }

private AVIReader reader;

private AVIWriter writer;

private bool[] RandUsed;

private int CurrentLength;

private int RandCounter;

private int Dimension;

private int SumPNSR;

private int SumPNSRFrame;

private static int BytePerFrame = 1024;

private static String HeaderFile = "header.txt";

public Video(String input, String password)

{

Password = ToPasswordSeed(password);

reader = new AVIReader();

reader.Open(input);

ResetReadByte();

ExtractHeader();

}

public Video(String input, String output, int length, String extension, int LSB, String password)

{

Length = length;

Extension = extension;

this.LSB = LSB;

Password = ToPasswordSeed(password);

SumPNSRFrame = 0;

SumPNSR = 0;

PNSR = 0;

reader = new AVIReader();

writer = new AVIWriter();

reader.Open(input);

writer.Open(output, reader.Width, reader.Height);

ResetWriteByte();

HideHeader();

}

public void InsertToFrame(byte[] insertedByte)

{

Bitmap bitmap = reader.GetNextFrame();

Dimension = bitmap.Height \* bitmap.Width;

ResetRandIndex();

int pos = RandIndex();

for (int i = 0; i < (insertedByte.Length); i++)

{

for (int j = 0; j < 3; j++)

{

int y = pos / bitmap.Size.Width;

int x = pos % bitmap.Size.Width;

byte r = bitmap.GetPixel(x, y).R;

byte g = bitmap.GetPixel(x, y).G;

byte b = bitmap.GetPixel(x, y).B;

if (LSB == 1)

{

int byteAtPos = (insertedByte[i] / (1 << (j \* 3))) % 2;

r = (byte)ToByte(r, byteAtPos, 1);

byteAtPos = (insertedByte[i] / (1 << (j \* 3 + 1))) % 2;

g = (byte)ToByte(g, byteAtPos, 1);

if (j < 2)

{

byteAtPos = (insertedByte[i] / (1 << (j \* 3 + 2))) % 2;

b = (byte)ToByte(b, byteAtPos, 1);

}

}

else if (LSB == 2)

{

int byteAtPos;

if (j < 2)

{

byteAtPos = ((insertedByte[i] / (1 << (j \* 6 + 1))) % 2) \* 2 +

((insertedByte[i] / (1 << (j \* 6))) % 2);

r = (byte)ToByte(r, byteAtPos, 2);

}

if (j == 0)

{

byteAtPos = ((insertedByte[i] / (1 << (j \* 6 + 3))) % 2) \* 2 +

((insertedByte[i] / (1 << (j \* 6 + 2))) % 2);

g = (byte)ToByte(g, byteAtPos, 2);

byteAtPos = ((insertedByte[i] / (1 << (j \* 6 + 5))) % 2) \* 2 +

((insertedByte[i] / (1 << (j \* 6 + 4))) % 2);

b = (byte)ToByte(b, byteAtPos, 2);

}

}

SumPNSR += ((bitmap.GetPixel(x,y).R - r) \*(bitmap.GetPixel(x,y).R - r) ) +

((bitmap.GetPixel(x,y).G - g) \* (bitmap.GetPixel(x,y).G - g)) +

((bitmap.GetPixel(x, y).B - b) \* (bitmap.GetPixel(x, y).B - b));

SumPNSRFrame++;

bitmap.SetPixel(x, y, Color.FromArgb(r, g, b));

pos = RandIndex();

}

}

PNSR = (double) SumPNSR / SumPNSRFrame;

writer.AddFrame(bitmap);

}

public byte[] GetByteFromNextFrame()

{

Bitmap bitmap = reader.GetNextFrame();

Dimension = bitmap.Height \* bitmap.Width;

int remaining = ((Length - CurrentLength) > BytePerFrame) ? BytePerFrame : (Length - CurrentLength);

if (remaining < 0)

{

return null;

}

byte[] ret = new byte[remaining];

ResetRandIndex();

int pos = RandIndex();

for (int i = 0; i < remaining; i++)

{

byte nowByte = 0;

for (int j = 0; j < 3; j++)

{

int y = pos / bitmap.Size.Width;

int x = pos % bitmap.Size.Width;

byte r = bitmap.GetPixel(x, y).R;

byte g = bitmap.GetPixel(x, y).G;

byte b = bitmap.GetPixel(x, y).B;

if (LSB == 1)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 3)));

nowByte += (byte)((g % 2) \* (1 << (j \* 3 + 1)));

if (j < 2)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 3 + 2)));

}

}

else if (LSB == 2)

{

if (j < 2)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 6)));

nowByte += (byte)(((r / 2) % 2) \* (1 << (j \* 6 + 1)));

}

if (j < 1)

{

nowByte += (byte)((g % 2) \* (1 << (j \* 6 + 2)));

nowByte += (byte)(((g / 2) % 2) \* (1 << (j \* 6 + 3)));

}

if (j < 1)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 6 + 4)));

nowByte += (byte)(((b / 2) % 2) \* (1 << (j \* 6 + 5)));

}

}

pos = RandIndex();

}

ret[i] = nowByte;

}

CurrentLength += BytePerFrame;

return ret;

}

private void ExtractHeader()

{

Bitmap bitmap = reader.GetNextFrame();

Dimension = bitmap.Height \* bitmap.Width;

ResetRandIndex();

int pos = RandIndex();

int len = 0;

// getting length header

for (int i = 0; i < 4; i++)

{

int nowByte = 0;

for (int j = 0; j < 3; j++)

{

int y = pos / bitmap.Size.Width;

int x = pos % bitmap.Size.Width;

byte r = bitmap.GetPixel(x, y).R;

byte g = bitmap.GetPixel(x, y).G;

byte b = bitmap.GetPixel(x, y).B;

if (LSB == 1)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 3)));

nowByte += (byte)((g % 2) \* (1 << (j \* 3 + 1)));

if (j < 2)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 3 + 2)));

}

}

else if (LSB == 2)

{

if (j < 2)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 6)));

nowByte += (byte)(((r / 2) % 2) \* (1 << (j \* 6 + 1)));

}

if (j < 1)

{

nowByte += (byte)((g % 2) \* (1 << (j \* 6 + 2)));

nowByte += (byte)(((g / 2) % 2) \* (1 << (j \* 6 + 3)));

}

if (j < 1)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 6 + 4)));

nowByte += (byte)(((b / 2) % 2) \* (1 << (j \* 6 + 5)));

}

}

pos = RandIndex();

}

len += nowByte;

}

//getting header

byte[] ret = new byte[len];

for (int i = 0; i < len; i++)

{

byte nowByte = 0;

for (int j = 0; j < 3; j++)

{

int y = pos / bitmap.Size.Width;

int x = pos % bitmap.Size.Width;

byte r = bitmap.GetPixel(x, y).R;

byte g = bitmap.GetPixel(x, y).G;

byte b = bitmap.GetPixel(x, y).B;

if (LSB == 1)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 3)));

nowByte += (byte)((g % 2) \* (1 << (j \* 3 + 1)));

if (j < 2)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 3 + 2)));

}

}

else if (LSB == 2)

{

if (j < 2)

{

nowByte += (byte)((r % 2) \* (1 << (j \* 6)));

nowByte += (byte)(((r / 2) % 2) \* (1 << (j \* 6 + 1)));

}

if (j < 1)

{

nowByte += (byte)((g % 2) \* (1 << (j \* 6 + 2)));

nowByte += (byte)(((g / 2) % 2) \* (1 << (j \* 6 + 3)));

}

if (j < 1)

{

nowByte += (byte)((b % 2) \* (1 << (j \* 6 + 4)));

nowByte += (byte)(((b / 2) % 2) \* (1 << (j \* 6 + 5)));

}

}

pos = RandIndex();

}

ret[i] = nowByte;

}

// output to header file

FileStream fs = File.OpenWrite(HeaderFile);

fs.Write(ret, 0, len);

fs.Close();

// read from Header

TextReader tr = new StreamReader(HeaderFile);

Length = Convert.ToInt32(tr.ReadLine());

Extension = tr.ReadLine();

LSB = Convert.ToInt32(tr.ReadLine());

tr.Close();

}

private void HideHeader()

{

// создание header файла

TextWriter tw = new StreamWriter(HeaderFile);

tw.WriteLine(Length);

tw.WriteLine(Extension);

tw.WriteLine(LSB);

tw.Close();

FileStream fs = File.OpenRead(HeaderFile);

byte[] temp = new byte[fs.Length];

fs.Read(temp, 0, Convert.ToInt32(fs.Length));

// добавление 4 байтов перед реальным хедером

byte[] bytes = new byte[fs.Length + 4];

int pos = 0;

for (int i = 0, k = 0; i < 4; i++)

{

bytes[pos++] = (byte) ((Convert.ToInt32(fs.Length) / (1 << k)) % (1 << 8));

k += 8;

}

for (int i = 0; i < Convert.ToInt32(fs.Length); i++)

{

bytes[pos++] = temp[i];

}

fs.Close();

InsertToFrame(bytes);

}

public void ResetReadByte()

{

CurrentLength = 0;

}

public void ResetWriteByte()

{

reader.Position = reader.Start;

}

public void CloseReader()

{

reader.Close();

}

public void CloseWriter()

{

InsertRemaining();

writer.Close();

}

private int ToPasswordSeed(String password)

{

int ret = 0;

for (int i = 0; i < password.Length; i++)

{

ret += (int) password[i];

}

return ret;

}

private void ResetRandIndex()

{

RandCounter = 0;

RandUsed = new bool[Dimension];

for (int i = 0; i < Dimension; i++)

{

RandUsed[i] = false;

}

}

private int RandIndex()

{

int r = (++RandCounter \* Password) % Dimension;

while (RandUsed[r])

{

r = (r + 1) % Dimension;

}

RandUsed[r] = true;

return r;

}

private void InsertRemaining()

{

while (true)

{

Bitmap bitmap = null;

try

{

bitmap = reader.GetNextFrame();

writer.AddFrame(bitmap);

}

catch (Exception e)

{

return;

}

}

}

private int ToByte(byte x, int byteAtPos, int len)

{

x -= (byte)(x % (1 << len));

x += (byte)byteAtPos;

return x;

}

}

}