**3Министерство образования Республики Беларусь**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра технологий программирования

**РАЗРАБОТКА ПРОГРАММНЫХ СРЕДСТВ ЗАЩИТЫ ДАННЫХ ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ**

Дипломная работа

Фаминчик Евгений Александрович

Минск, 2018

**ОГЛАВЛЕНИЕ**

[РЕФЕРАТ 4](#_Toc515565826)

[ВВЕДЕНИЕ 6](#_Toc515565827)

[ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc515565828)

[1.1 Основные определения 7](#_Toc515565829)

[1.2 Существующие каналы несанкционированного получения информации 7](#_Toc515565830)

[1.3 Существующие методы и средства защиты программ и данных 9](#_Toc515565831)

[1.3.1 Идентификация и аутентификация 11](#_Toc515565832)

[1.3.2 Пароли 11](#_Toc515565833)

[1.3.3 Токен 13](#_Toc515565834)

[1.3.4 Разграничение доступа к элементам защищаемой информации 14](#_Toc515565835)

[1.3.5 Криптографическое закрытие 14](#_Toc515565836)

[1.3.6 "Запутывание" 16](#_Toc515565837)

[1.3.7 Противостояние изучению 17](#_Toc515565838)

[1.4 Сжатие данных 19](#_Toc515565839)

[1.4.1 Влияние сжатия и шифрования друг на друга 19](#_Toc515565840)

[1.4.2 Обзор существующих алгоритмов сжатия данных 21](#_Toc515565841)

[1.5 Основные программы-архиваторы 31](#_Toc515565842)

[1.5.1 Исследование эффективности алгоритмов сжатия данных без потерь 33](#_Toc515565843)

[1.5.2 Статистика использования программ-архиваторов 34](#_Toc515565844)

[1.6. Исследование структуры файла 35](#_Toc515565845)

[ГЛАВА 2. ПРОГРАММНОЕ СРЕДСТВО 37](#_Toc515565846)

[2.1 Инструментарий 37](#_Toc515565847)

[2.2 Основные требования 37](#_Toc515565848)

[2.3 Поведение программного средства 37](#_Toc515565849)

[2.3.1 Описание «Encrypt Mode» 38](#_Toc515565850)

[2.3.2 Описание «Decrypt Mode» 46](#_Toc515565851)

[2.3.3 Описание «Menu Bar» 51](#_Toc515565852)

[2.4 Описание программного средства с технической стороны. 51](#_Toc515565853)

[2.4.1 Описание процесса защиты данных 51](#_Toc515565854)

[2.4.2 Структура выходного файла 52](#_Toc515565855)

[2.4.3 Описание процесса снятия защиты с данных и проверки пароля на достоверность 54](#_Toc515565856)

[2.5 Сравнение разработанного программного средства с уже существующими 55](#_Toc515565857)

[ЗАКЛЮЧЕНИЕ 57](#_Toc515565858)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 58](#_Toc515565859)

[ПРИЛОЖЕНИЯ 60](#_Toc515565860)

[ПРИЛОЖЕНИЕ А 60](#_Toc515565861)

[ПРИЛОЖЕНИЕ Б 62](#_Toc515565862)

[ПРИЛОЖЕНИЕ В 68](#_Toc515565863)

[ПРИЛОЖЕНИЕ Г 72](#_Toc515565864)

[ПРИЛОЖЕНИЕ Д 74](#_Toc515565865)

[ПРИЛОЖЕНИЕ Е 75](#_Toc515565866)

[ПРИЛОЖЕНИЕ Ж 79](#_Toc515565867)

## РЕФЕРАТ

Дипломная работа: 59 страниц, 30 рисунков, 8 таблиц, 18 источников.

Ключевые слова: защита, несанкционированное использование, пароль, криптографическое закрытие, сжатие данных.

Объект исследования - методы защиты данных от несанкционированного использования.

Цели работы - произвести обзор методов защиты данных от несанкционированного использования, произвести сравнительный анализ и выбрать наиболее эффективный алгоритм сжатия данных без потерь, разработать программное средство для защиты данных от несанкционированного использования.

Результат работы - произведен обзор методов защиты данных от несанкционированного использования, произведен анализ эффективности алгоритмов сжатия данных без потерь, разработано программное средство для защиты данных от несанкционированного использования.

**РЭФЕРАТ**

Дыпломная работа: 59 старонак, 30 малюнкаў, 8 табліц, 18 крыніц.

Ключавыя словы: абарона, несанкцыянаванае выкарыстанне, пароль, крыптаграфічныя закрыццё, сціск дадзеных.

Аб'ект даследавання - метады абароны дадзеных ад несанкцыянаванага выкарыстання.

Мэты работы - зрабіць агляд метадаў абароны дадзеных ад несанкцыянаванага выкарыстання, зрабіць параўнальны аналіз і выбраць найбольш эфектыўны алгарытм сціску дадзеных без страт, распрацаваць праграмны сродак для абароны дадзеных ад несанкцыянаванага выкарыстання.

Вынік работы - зроблен агляд метадаў абароны дадзеных ад несанкцыянаванага выкарыстання, зроблен аналіз эфектыўнасці алгарытмаў сціску дадзеных без страт, распрацаваны праграмны сродак для абароны дадзеных ад несанкцыянаванага выкарыстання.

**ABSTRACT**

Graduate work: 59 pages, 30 images, 8 tables, 18 sources.

Keywords: protection, unauthorized use, password, cryptographic processing, data compression.

Research object - methods of data protection against to unauthorized use.

Work goals - make a review of data protection methods against to unauthorized use, make a comparative analysis and opt the most effective algorithm of data compression without loss, develop a software tool for protecting data against to unauthorized use.

Work result - was made a review of data protection methods against to unauthorized use, was made an analysis of algorithms efficiency of data compression without loss, was developed a software tool for protecting data against to unauthorized use.

## ВВЕДЕНИЕ

Современное общество не может прожить и дня без информации. В общем смысле информация содержит сведения об окружающем мире и является объектом хранения, передачи, преобразования и использования её для определенных целей.

Информация может быть коммерческой, военной, политической, экономической, научно-технической или производственной. По степени секретности можно разделить информацию на конфиденциальную и неконфиденциальную. К конфиденциальной информации обычно относятся сведения, которые содержат коммерческую, служебную, врачебную, адвокатскую, следственную тайну, а также тайну переписки почтовых и телеграфных отправлений, телефонных переговоров и сведения о частной жизни и деятельности граждан.

С развитием технологий объем информации стремительно возрастает и человек уже не в состоянии хранить ее в собственной памяти. На помощь же приходят современные средства хранения информации - информационные системы. Но информация, которая хранится в информационных системах, зачастую подвержена опасности вероятного доступа и использования ее сторонними лицами. Поэтому информационная безопасность не только становится обязательной, но и выступает как одна из важнейших характеристик информационной системы. В большом количестве систем безопасности отведена первостепенная роль фактору безопасности информационных систем. Многие современные организации, которые занимаются бизнесом, не могут вести деятельность без обеспечения безопасности своей конфиденциальной информации.

С современным развитием информационного пространства огромное значение приобрело качество защиты конфиденциальной информации. Методы защиты такой информации рассматриваются в данной работе.

## ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Основные определения

Для понимания, о чем конкретно будет идти речь, представлены основные понятия в данной области:

*Программа* – описание алгоритма решения задач для работы вычислительной машины. [1, c. 540]

*Данные* – формы представления информации, с которыми имеют дело информационные системы и их пользователи. [2]

*Защита* – предохранение от чего-либо. [1, c. 203]

*Несанкционированное использование* – использование, предпринятое без соответствующих полномочий

## 1.2 Существующие каналы несанкционированного получения информации

1. Каналы, проявляющиеся безотносительно к обработкеинформации и без доступа злоумышленника к элементам электронно-вычислительной технике (ЭВТ): [3, с. 81-83]

* использование злоумышленником акустических средств;
* использование злоумышленником визуальных средств;
* использование злоумышленником оптических средств;
* провоцирование на разговоры лиц, имеющих отношение к СОИ;
* подслушивание разговоров лиц, имеющих отношение к системе обработки информации (СОИ);
* хищение носителей информации на заводах, где производится ремонт ЭВТ.

1. Каналы, проявляющиеся в процессе обработки информациибез доступа злоумышленника к элементам СОИ:

* подключение регистрирующей аппаратуры;
* подключение генераторов помех;
* электромагнитные излучения вспомогательной аппаратуры;
* электромагнитные излучения аппаратуры связи;
* электромагнитные излучения линий связи;
* электромагнитные излучения внешних запоминающих устройств;
* электромагнитные излучения устройств отображения информации;
* электромагнитные излучения процессоров.

1. Каналы, проявляющиеся безотносительно к обработке информациис доступом злоумышленника к элементам СОИ, но без изменения последних:

* хищение производственных отходов;
* копирование других документов;
* копирование с устройств отображения;
* копирование магнитных носителей;
* копирование выходных документов;
* копирование бланков с исходными данными.

1. Каналы, проявляющиеся в процессе обработки информации с доступом злоумышленника к элементам СОИ, но без изменения последних:

* использование зараженности программного обеспечения вирусом;
* использование недостатков операционных систем;
* маскировка под зарегистрированного пользователя; использование недостатков систем программирования;
* использование программных ловушек;
* копирование распечатки массивов;
* изготовление дубликатов массивов и выходных документов;
* копирование (фотографирование) информации в процессе обработки;
* запоминание служебных данных;
* запоминание информации на выходных документах;
* запоминание информации с устройств наглядного отображения;
* запоминание информации на бланках с исходными данными.

1. Каналы, проявляющиеся безотносительно к обработке информации с доступом злоумышленника к элементам СОИ с изменением последних:

* подмена/хищение бланков;
* подмена/хищение магнитных носителей;
* подмена/хищение выходных документов;
* подмена/хищение аппаратуры;
* подмена элементов программ;
* подмена элементов баз данных;
* хищение других документов;
* внедрение в программы блоков типа «троянский конь», «бомба».

1. Каналы, проявляющиеся в процессе обработки информации с доступом злоумышленника к объектам СОИ с их изменением:

* снятие информации на шинах питания внешних запоминающих устройств;
* снятие информации на шинах питания аппаратуры связи;
* снятие информации на шинах питания процессоров;
* снятие информации на шинах питания линий связи;
* снятие информации на шинах питания устройств отображения;
* незаконное подключение к аппаратуре;
* незаконное подключение к линиям связи.

## 1.3 Существующие методы и средства защиты программ и данных

Методы защиты программ и данных согласно [4, с 186]:

* управление доступом;
* принуждение;
* побуждение;
* регламентация;
* механизмы шифрования;
* создание препятствий;
* противодействие атакам вредоносных программ.

**Управление доступом** – методы защиты информации регулированием использования всех ресурсов информационных систем (ИС) и информационных технологий (ИТ). [4, c. 186]

**Принуждение** – метод защиты, при котором пользователи и персонал ИС вынуждены соблюдать правила обработки, передачи и использования защищаемой информации под угрозой материальной, административной или уголовной ответственности. [4, c 187]

**Побуждение** – метод защиты, побуждающий пользователей и персонал ИС не нарушать установленные порядки за счет соблюдения сложившихся моральных и этических норм. [4, c 187]

**Регламентация** – создание таких условий автоматизированной обработки, хранения и передачи защищаемой информации, при кото­рых нормы и стандарты по защите выполняются в наибольшей степени. [4, c 187]

**Механизмы шифрования** – криптографическое закрытие информации. Эти методы защиты все шире применяются как при обработке, так и при хранении информации на магнитных носителях. [4, c. 186]

**Создание препятствий** – метод физического преграждения пути злоумышленнику к защищаемой информации. [4, c. 186]

**Противодействие атакам вредоносных программ** предполагает комплекс разнообразных мер организационного характера и исполь­зование антивирусных программ. [4, c 187]

Вся совокупность технических средств подразделяется на физические и аппаратные. [4, c 187]

**Физические средства** включают различные инженерные устройства и сооружения, препятствующие физическому проникновению злоумышленников на объекты защиты и осуществляющие защиту персонала, материальных средств и финансов, информации от противоправных действий. [4, c 187]

**Аппаратные средства** – устройства, встраиваемые непосредственно в вычислительную технику, или устройства, которые сопрягаются с ней по стандартному интерфейсу. [4, c 187]

**Организационные средства** осуществляют своим комплексом регламентацию производственной деятельности в ИС и взаимоотношений исполнителей на нормативно-правовой основе таким образом, что разглашение, утечка и несанкционированный доступ к конфиденциальной информации становится невозможным или существенно затрудняется за счет проведения организационных мероприятий. [4, c 187]

**Программные средства** – это специальные программы и программные комплексы, предназначенные для защиты информации в ИС. [4, c 187]

**Морально-этические** средства защиты включают всевозможные нормы поведения, складываются по мере распространения ИС и ИТ в стране и в мире или специально разрабатываются. [4, c 188]

**Законодательные средства** защиты определяются законодательными актами страны, которыми регламентируются правила пользования, обработки и передачи информации ограниченного доступа и устанавливаются меры ответственности за нарушение этих правил. [4, c 188]

## 1.3.1 Идентификация и аутентификация

В качестве средств управления доступом к системе обработки информации стоит выделить **идентификацию** и **аутентификацию** при входе в систему и разграничение доступа при нахождении внутри системы.

Идентификация позволяет субъекту - процессу или пользователю - сообщить свое имя. Посредством аутентификации вторая сторона убеждается, что субъект действительно тот, за кого он себя выдает. [3, c. 125]

Субъект может подтвердить свою подлинность, если предъявит хотя бы одну из следующих идентификационных сущностей:

* пароль, личный идентификационный номер, криптографический ключ, то есть то, что он знает;
* личную карточку или другое устройство аналогичного назначения, то есть то, чем он владеет;
* биометрические характеристики (отпечатки пальцев, голос), то есть то, что есть атрибут его самого;

К сожалению, надежная идентификация и аутентификация затруднена из-за следующих причин:

* компьютерная система основывается на информации в том виде, в котором она была получена, то есть, если говорить строго, источник информации остается неизвестным;
* почти все аутентификационные сущности можно украсть, подделать или узнать;
* противоречие между надежностью аутентификации и удобствами пользователя и системного администратора;
* чем надежнее средство защиты, тем оно дороже; особенно дорогими являются устройства измерения биометрических характеристик.

## 1.3.2 Пароли

**Пароли**являются наиболее распространенным средством аутентификации. Система сравнивает введенный и ранее заданный для данного пользователя пароль. В случае совпадения паролей подлинность пользователя считается доказанной. Существуют следующие парольные методы проверки подлинности пользователей при входе в систему, которые можно разделить на две группы:

* методы проверки подлинности на основе динамически изменяющегося пароля;
* методы проверки подлинности на основе простого пароля.

Пароль подтверждения подлинности пользователя при использовании простого пароля не изменяется от сеанса к сеансу в течение установленного администратором службы безопасности времени его существования.

При использовании динамически изменяющегося пароля пароль пользователя для каждого нового сеанса работы или нового периода действия одного пароля изменяется по правилам, которые зависят от используемого метода.

Можно выделить следующие способы опознавания при использовании простого пароля в виде следующей последовательности действий:

* пользователь посылает запрос на доступ к компьютерной системе и вводит свой идентификатор;
* система запрашивает пароль;
* пользователь вводит пароль;
* система сравнивает полученный пароль с паролем пользователя, хранящимся в базе данных системы защиты, и разрешает доступ, если пароли совпадают; иначе же пользователь не допускается к ресурсам компьютерной системы.

Опишем основные способы повышения стойкости системы защиты на этапе аутентификации:

* повышение степени нетривиальности пароля;
* наложение технических ограничений;
* ограничение числа неудачных попыток входа в систему (это приведет к затруднению использования метода грубой силы);
* увеличение времени задержки между разрешенными попытками повторного ввода при неправильно введенном пароле;
* обучение пользователей правилам парольной защиты;
* повышение ограничений на минимальное и максимальное время действия пароля.
* использование генераторов паролей (данная программа, которая основывается на несложных правилах, может порождать только благозвучные и запоминающиеся пароли).
* наложить ограничение доступа к файлу паролей.

Для поддержания необходимого уровня парольной защиты следует придерживаться определенных правил генерации паролей:

* длина пароля должна быть не менее установленной (6-8 символов);
* пароль не должен включать в себя легко вычисляемые сочетания символов (номера телефонов, имена, фамилии), а также общеизвестные сокращения;
* обязательно должны присутствовать буквы в верхнем и нижнем регистрах, цифры специальные символы (@, #, $, &, \*, %);
* личный пароль никогда нельзя сообщать третьим лицам;
* при смене пароля новое значение должно отличаться от предыдущего на определенное количество символов.

Перечисленные меры необходимо применять всегда, даже когда вместе с паролями используются и другие методы аутентификации, которые основаны, например, на применении токенов.

## 1.3.3 Токен

**Токен**-устройство или предмет, которое способно подтвердить подлинность пользователя [3, c. 129]. Есть два вида токенов: активные (интеллектуальные) и пассивные, которые только хранят, но не обрабатывают информацию (с памятью).

Самый широкоизвестный вид токенов с памятью – это карточки с магнитной полосой. Для использования таких токенов нужно устройство чтения, которое снабженно также процессором и клавиатурой. Обычно пользователь вводит на данной клавиатуре личный идентификационный номер, и затем уже процессор проверяет его совпадение с тем, что было записано на карточке, а также подлинность самой карточки.

У токенов с памятью также есть и определенные недостатки. Они намного дороже паролей, их необходимо изготовлять, произвожить процесс раздачи пользователям, обслуживать случаи потери. К тому же они нуждаются в специальных устройствах чтения.

Отличительный признак интеллектуальных токенов - это присутствие собственной вычислительной мощности. Они делятся на интеллектуальные карты и прочие токены. Карты нуждаются в интерфейсном устройстве, а прочие токены обычно обладают клавиатурой или дисплеем и напоминают калькуляторы по внешнему виду. Для начала работы токена пользователю необходимо ввести свой личный идентификационный номер.

Главное достоинство интеллектуальных токенов - возможность применения их при аутентификации по открытой сети. Выдаваемые или генерируемые в ответ пароли постоянно изменяются, поэтому злоумышленник при перехвате текущего пароля не получит никаких преимуществ.

Основной же недостаток интеллектуальных токенов – это их высокая стоимость. Правда, отчасти данный момент можно обозначать как и достоинство, поскольку тогда затрудняется подделка.

## 1.3.4 Разграничение доступа к элементам защищаемой информации

Разграничение доступа заключается в том, чтобы каждому зарегистрированному пользователю предоставить возможности беспрепятственного доступа к информации в пределах его полномочий, и исключить возможности превышения своих полномочий. В данных целях разработаны и реализованы на практике методы и средства разграничения доступа к программам обработки информации, к устройствам ЭВМ, к полям и к массивам (базам) данных. Разграничение может осуществляться следующими двумя способами:

* принудительное управление доступом;
* произвольное управление доступом.

**Произвольное управление доступом** - ограничение доступа к объектам, которое основано на учете личности субъекта или группы, в которые входит субъект [3, c. 132]. Произвольность управления заключается в том, что некоторое лицо может по своему усмотрению давать или отбирать у других субъектов права доступа к объекту. Есть два подхода к построению дискреционного управления доступом:

* система имеет одного выделенного субъекта - суперпользователя, который имеет право устанавливать права владения для всех остальных субъектов системы;
* каждый объект системы имеет привязанного к нему субъекта, который именуется владельцем. Именно владелец устанавливает права доступа к объекту.

## 

## 1.3.5 Криптографическое закрытие

Криптографические методы защиты основаны на возможности осуществления некоторой операции преобразования информации, которая может выполняться одним или несколькими пользователями СОИ, обладающими некоторым секретом, без знания которого невозможно осуществить данную операцию.

К криптографическим методам относят шифрование и кодирование.

**Шифрование** - такой вид криптографического закрытия, при котором каждый символ защищаемого сообщения подвергается преобразованию. [3, c. 137]. Способы шифрования можно разбить на пять групп: гаммирование, аналитическое преобразование, комбинированное шифрование, подстановка (замена) и перестановка. Каждый из них может иметь несколько разновидностей.

**Кодирование** - такой вид криптографического закрытия, когда некоторые элементы защищаемых данных заменяются кодами, которые были выбраны заранее - цифровыми, буквенными, буквенно-цифровыми сочетаниями [3, с. 137]. У этого метода существует две разновидности: символьное и смысловое кодирование. При символьном кодировании кодируется каждый символ защищаемого сообщения. Символьное кодирование совпадает с шифрованием заменой. При смысловом кодировании кодируемые элементы имеют определенный смысл (слова, предложения).

**Гаммирование** - данный метод заключается в наложении на исходный текст некоторой псевдослучайной последовательности, которая генерируется на основе ключа. [3, с. 139]

**Перестановки** -несложный метод криптографического преобразования. Используется, как правило, в сочетании с другими методами. [3, с. 137]

**Многоалфавитная подстановка** -наиболее простой вид преобразований, который заключаюется в замене символов исходного текста на другие символы этого же алфавита по определенному правилу. Чтобы обеспечить высокий уровень криптостойкости требуется использование больших ключей. [3, с. 138]

**Блочные шифры** - последовательность с возможным повторением и чередованием основных методов преобразования, которые применяются к блоку шифруемого текста. [3, с. 139]

Основные методы криптографического закрытия информации:

* **Шифрование методом перестановки** - в данном методе символы шифртекста переставляются по определенным правилам внутри шифруемого блока символов.
* **Шифрование заменой (подстановка)** - в данном методе символы шифртекста заменяются другими символами, которые взяты из одного или нескольких алфавитов. Разновидностью этого метода является простая замена, когда буквы шифруемого сообщения заменяются другими буквами того же или другого алфавита.
* **Шифрование методом гаммирования** заключается в том, что символы шифртекста суммируются последовательно с символами некоторой служебной последовательности, называемой гаммой.

## 1.3.6 "Запутывание"

Программное средство, которое написал человек, может быть им разобрана, понята и проанализирована. Методы, которые были упомянуты выше, являются достойными средствами защиты, однако их подробное изучение взломщиком может свести на нет всю их эффективность. Поэтому данные методы следует применять вместе с защитой от анализа кода программного средства.

Обфускация ("запутывание") - превращение исходного кода программы к виду, в котором сохранена функциональность, но сильно затруднено понимание принципов функционирования. Для создания обфускаций существуют специальные программы, которые называются называются обфускаторами.

Определенные программы распространяются в виде исходного текста. Для затруднения читаемости кода могут применяться советы из [5]. Но код подобного рода писать достаточно сложно, а еще сложнее данный код поддерживать. Поэтому стоит вначале написать программу в понятном виде, а затем уже и произвести процесс обфускации.

Также стоит помнить, что неверные комментарии, ужасное форматирование и переименования переменных никак повлияют на итоговое поведение программного средства.

Для автоматизации процесса обфускации исходного кода существует множество программных средств, которые, однако, различаются своими возможностями и ценой. Недостатком некоторых из них является существование программ-деобфускаторов, которые способны "раздеть" программу, защищенную с помощью не совсем качественных обфускаторов.

Любопытных результатов можно достичь, если писать программное средство языком программирования, в котором существуют функции, позволяющие выполнить код, который был передан на вход в виде строки (eval в JavaScript, exec в Python). Таким образом, команды можно спрятать в строке, а при выполнении программы извлечь строку и передать её в функцию. К примеру, на сайте [6] программа шифрует код в набор пробелов и табуляций, после чего в текстовом редакторе или браузере без определенных манипуляций код становится полностью "невидимым", а вот программа на сайте [7] превращает код в набор японских смайлов. Однозначно, эффективность данных методов невысока, но неопытного взломщика сбить с толку можно.

## 1.3.7 Противостояние изучению

Для исследования программ обычно применяются различные программные средства:

* дизассемблеры
* отладчики
* вспомогательные утилиты, которые следят за обращением программы к жесткому диску и реестру, снимающие дамп памяти.

Дизассемблеры (статическое исследование) в основном используются для изучения кода программы без исходных текстов. Они преобразуют машинный код в текст на языке ассемблера. Отладчики (динамическое исследование) информируют о всех процессах, которые протекают в компьютере, после выполнения отдельного участка или каждой инструкции программы.

Отладчики используют два вида точек останова - программные и аппаратные.

Противостоять статическому исследованию можно следующими способами:

1. Включение в тело программы переходов по динамически изменяемым адресам
2. Перекрывающийся код
3. Шифрование программного кода
4. Обфускация
5. Генерация исполняемых команд впереди вычислительного процесса, производя операция над кодом или массивом данных.
6. Создание длинной команды и последующий переход в ее середину так, чтобы попасть на код "осмысленной" команды. Такое действие приводит к тому, что после данной команды дизассемблер начинает выдавать неверную последовательность команд.

Это основные методы "запутывания" дизассемблеров. Большинство из них основано на том, что на самом деле во время выполнения программы команды будут отличны от тех, которые были первоначально записаны в анализируемый дизассемблером исполняемый модуль.

Существует также отладочные механизмы. Противостоять исследованию под отладчиком можно двумя путями:

1. Обнаружить отладчик и передать управление на некоторую ветку реакции на отладчик.
2. Поместить в программу код, выполняющийся нормально без отладчика, но под отладчиком приводящий к аварийному завершению, зависанию компьютера или искажению хода выполнения программы.

Выявить отладчик можно следующим образом:

1. Проверять время выполнения отдельных участков программы.
2. Проверять результат подсчета контрольной суммы, занимаемой программой в оперативной памяти.
3. WinAPI функция IsDebuggerPresent.
4. Межпоточное взаимодействие. Есть два потока: один выполняет полезный код и периодически извещает второй поток о том, что является активным. Второй поток следит за частотой, с которой "отзывается" первый поток.

Первый способ защиты применим, но имеет недостаток: противник может вообще не разбираться, как программа определяет наличие отладчика. Ему достаточно найти команду, которая отвечает за переход на ветку с реакцией на отладчик и изменить ее. Поэтому способ номер один необходимо комбинировать со вторым, где явная проверка не производится.

Также стоит обратить внимание, что программы, которые защищены только от дизассемблирования, могут легко быть изучены с помощью отладчика и наоборот. Следовательно, для противодействия изучению алгоритмов систем защиты необходимо иметь средства, которые препятствуют как дизассемблированию, так и пошаговому выполнению.

## 1.4 Сжатие данных

Сжатие данных – это преобразование с применением определенных алгоритмов. Целью сжатия является сокращение объема данных. Как и шифрование, является способом кодирования информации.

Алгоритмы сжатия данных можно разделить на два больших класса - с потерями и без. Алгоритмы с потерями чаще всего применяются для сжатия аудио изображений. Данные алгоритмы позволяют достичь больших степеней сжатия посредством избирательной потере качества. Однако, процесс восстановления первоначальных данных из сжатого результата невозможен.

Алгоритмы сжатия данных без потерь используются для уменьшения размера данных и работают таким образом, что восстановить данные в точности, какими они были до сжатия действительно возможно. Данные алгоритмы применяются в архиваторах и некоторых алгоритмах сжатия графической информации и аудио.

Основной принцип алгоритмов сжатия основан на том, что в любом файле, который содержит неслучайные данные, информация частично повторяется. И если использовать статистические математические модели, то можно определить вероятность повторения определённой комбинации символов. После можно создать коды, которые обозначают выбранные фразы, и назначить самым часто повторяющимся фразам самые короткие коды. Для этого применяются различные техники, такие как: кодирование повторов, сжатие при помощи словаря и энтропийное кодирование. С помощью их можно заменить 8-битный символ или же целую строку несколькими битами, таким образом устраняя излишнюю информацию.

## 1.4.1 Влияние сжатия и шифрования друг на друга

Для уменьшения объема информация должна быть закодирована определенным образом, и для ее последующего использования или прочтения необходима обратная раскодировка. Поэтому, с одной стороны, сжатие может рассматриваться в некоторых случаях в качестве криптографического преобразования при сохранении в секрете алгоритмов сжатия. Тогда для восстановления информации из сжатых файлов требуются статистические методы обработки, что исключает несанкционированный доступ случайных лиц к данным.

Однако, доступность средств сжатия и, соответственно, обратного преобразования, не делает алгоритмы сжатия средствами криптографической защиты. Поэтому, если конфиденциальная информация должна быть надежно защищена от злоумышленников и криптоанализа, файлы с секретными данными должны быть дополнительно зашифрованы с помощью некоторого криптографического алгоритма.

Таким образом, если использовать и сжатие данных, и их шифрование, необходимо учитывать влияние данных методов друг на друга.

Есть два преимущества в шифровании предварительно сжатого файла:

* Во-первых, сжатие файла перед шифрованием уменьшает избыточность информации, которая обычно используется для взлома шифра при криптоанализе. Таким образом, сжатие способствует шифрованию, так как затрудняется поиск шифра доступным для взломщика статистическим методом;
* Во-вторых, ускоряется процесс шифрования, являющийся достаточно длительной процедурой, из-за уменьшения длины предварительно сжатой информации, которую требуется зашифровать.

Однако эти преимущества возникают только при предварительном сжатии и последующем шифровании файла. Важно сжимать данные именно перед шифрованием.

В противном же случае, предварительное шифрование оказывает отрицательное влияние на сжатие по следующей причине. Если шифрование выполняется правильно, то результатом является псевдослучайная последовательность, близкая по своим параметрам к случайным данным. Для сжатия же и уменьшения размера большинство схем ищут повторяющиеся последовательности в данных в целях их дальнейшего исключения. Но так как шифрование превращает структурированные данные в случайные, удаляя последовательности, алгоритм сжатия не сможет сильно уменьшить размер, если применить его к зашифрованным данным. Таким образом, предварительно зашифрованные данные становятся плохо сжимаемыми.

Посредством сжатия даже можно определять, насколько качественным является применяемый для шифрования данных алгоритм. Чем в меньшей степени поддаются сжатию зашифрованные данные, тем лучше криптографический алгоритм.

Итак, в качестве вывода из вышеизложенного следует, что при использовании и сжатия, и шифрования файлов с конфиденциальной информацией, требующей защиты от несанкционированного использования, необходимо сначала выполнять сжатие данных, а затем их шифрование (рисунок №1.1).



*Рисунок 1.1. Схема корректной последовательности при совместном сжатии и шифровании*

## 1.4.2 Обзор существующих алгоритмов сжатия данных

#### Алгоритм Хаффмана

Алгоритм Хаффмана (Huffman codes) – это широко распространенный и очень эффективный метод сжатия данных, который обычно позволяет сэкономить от 20% до 90% объема.

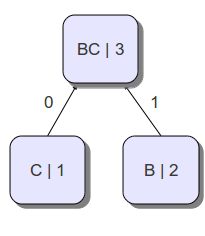
Рассматриваются данные, которые представляют собой последовательность символов. В данном алгоритме используется таблица, которая содержит частоты появления тех или иных символов. С помощью данной таблицы определяется оптимальное представление отдельного символа в виде бинарной строки.

***Построение кода:***

В основу алгоритма была положена следующая идея: кодировать более коротко именно те символы, которые встречаются чаще, а те, которые уже встречаются реже кодировать длиннее. Для построения кода Хаффмана нам необходима таблица частот символов.

Пример [8] построения кода на строке *«abacaba»* (рисунки №1.2 и №1.3)*:*

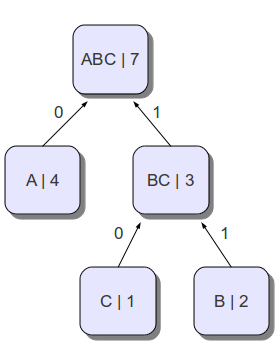
* *a b c*
* *4 2 1*



*Рисунок 1.2. Пример из Алгоритма Хаффмана: обработка «b» и «c»*

Следующим шаг – это построение дерева, где вершины - «символы», а пути соответствуют их префиксным кодам. Для этого на каждом шаге нужно брать два символа с минимальной частотой вхождения и объединять их в новые символы с частотой, которая будет равна сумме тех частот, которые объединяли, и также соединять их рёбрами, образуя тем самым дерево. Нужно выбирать минимальные два символа из всех, исключая уже те, которые выбирали. В примере объединяется *«b»* и *«с»* в символ *«bc»* с частотой 3. Затем объединяется *«a»* и *«bc»* в символ *«abc»*, таким образом получается дерево. Теперь пути от корня (*«abc»*) до листьев и есть Коды Хаффмана (каждому ребру соответствует либо 1 либо 0).

* *a b c*
* *0 11 10*



*Рисунок 1.3. Пример алгоритма Хаффмана: получившееся дерево*

#### Алгоритм LZW

Алгоритм Лемпеля - Зива - Велча (Lempel-Ziv-Welch, LZW) - алгоритм сжатия данных без потерь.

Процесс сжатия данных можно описать так: последовательно считываются символы входного потока, затем происходит проверка, есть ли в созданной таблице строк данная строка. Если данная строка присутствует, то считывается следующий символ, а если нет, то в поток заносится код найденной строки, которая была найдена на предыдущем шаге, сама же строка заносится в таблицу, а поиск начинается сначала.

К примеру, если сжимаются байтовые данные, то в таблице строк окажется 256 (от 0 до 255). При использовании 10-битного кода коды для строк будут значения в от 256 до 1023. Новые строки формируют таблицу последовательно, следовательно, ее можно считать кодом индекс строки.

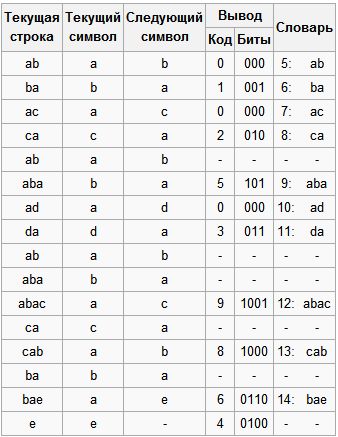
Алгоритму декодирования на входе требуется только закодированный текст, так как он может восстановить таблицу преобразования по тексту, который был закодирован. Алгоритм генерирует декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. Алгоритм постоянно проверяет, является ли строка уже известной, и если является, то выводит существующий код без генерации нового. Следовательно, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер. Таким образом, при дешифровании при получении нового кода генерируется новая строка, а при получении уже известного, строка извлекается из словаря.

Пример сжатия и декодирования изображения [9].

***Кодирование:***

Сжимается последовательность *«abacabadabacaba*e».

* добавляем к изначально пустой строке *«a»* и проверяем, есть ли строка *«a»* в таблице.
* читаем следующий символ *«b»* из входного потока и проверяем, присутствует ли строка *«ab»* в таблице. Данной строки нет.
* добавляем в таблицу *<5>* *«ab»*. В поток: *<0>*;
* *«ba»* - нет. В таблицу: *<6>* *«ba»*. В поток: *<1>*;
* *«ac»* - нет. В таблицу: *<7>* *«ac»*. В поток: *<0>*;
* *«ca»* - нет. В таблицу: *<8>* *«ca»*. В поток: *<2>*;
* *«ab»* - есть в таблице; «aba» - нет. В таблицу: *<9>* *«aba»*. В поток: *<5>*;
* *«ad»* - нет. В таблицу: *<10>* *«ad»*. В поток: *<0>*;
* *«da»* - нет. В таблицу: *<11>* *«da»*. В поток: *<3>*;
* «aba» - есть в таблице; *«abac»* - нет. В таблицу: *<12>* *«abac»*. В поток: *<9>*;
* *«ca»* - есть в таблице; *«cab»* - нет. В таблицу: *<13>* *«cab»*. В поток: *<8>*;
* *«ba»* - есть в таблице; *«bae»* - нет. В таблицу: *<14>* *«bae»*. В поток: *<6>*;
* последняя строка *«e»*, за ней конец сообщения, поэтому происходит вывод в поток *<4>* (рисунок №1.4).



*Рисунок 1.4. Пример алгоритма LZW: таблица на этапе кодирования*

Получаем закодированное сообщение *«0 1 0 2 5 0 3 9 8 6 4»*, что на 11 бит короче.

##### ***Декодирование:***

Особенность LZW заключается в том, что для декомпрессии необязательно хранить таблицу строк в файле для распаковки. Алгоритм построен так, что можно восстановить таблицу строк, используя только поток кодов.

Предположим, что было получено закодированное сообщение, которое приведено выше, и его нужно декодировать. Для начала необходимо знать начальный словарь, а последующие записи словаря можно реконструировать уже по ходу, так как они являются конкатенацией предыдущих записей.

Пример можно посмотреть на рисунке №1.5.



*Рисунок 1.5. Пример алгоритма LZW: таблица на этапе декодирования*

#### Алгоритмы LZ77 и LZ78

LZ77 и LZ78 - алгоритмы сжатия без потерь. Данные алгоритмы широко известны в семействе LZ\*, которое включает в себя и LZW, LZSS, LZMA и другие алгоритмы. Оба алгоритма относятся к алгоритмам со словарным подходом.

##### ***Принцип работы LZ77:***

Основная идея - замена повторного вхождения строки ссылкой на одну из предыдущих позиций вхождения. Для этого используют метод скользящего окна. Скользящее окно можно представить в виде динамической структуры данных, организованной так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Следовательно, сам процесс сжатия напоминает написание программы, команды которой позволяют обращаться к элементам скользящего окна, и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в скользящем окне. В стандартном алгоритме LZ77 совпадения строки кодируются парой:

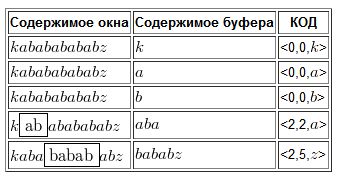
* длина совпадения (match length)
* смещение (offset) или дистанция (distance)

Кодируемая пара трактуется как команда копирования символов из скользящего окна с конкретной позиции, или как: «Вернуться в словаре на значение смещения символов и скопировать значение длины символов, начиная с текущей позиции». Особенностью данного алгоритма являюется то, что использование кодируемой пары длина-смещение является как приемлемым, так и эффективным в моментах, когда значение длины превышает значение смещения.

##### ***Описание алгоритма LZ77:***

LZ77 использует скользящее по сообщению окно. Предположим, на текущей итерации окно зафиксировано. С правой стороны окна наращиваем подстроку, пока она есть в строке < скользящее окно + наращиваемая строка > и начинается в скользящем окне. Назовем наращиваемую строку буфером. После наращивания алгоритм выдает код состоящий из трех элементов:

* смещение в окне;
* длина буфера;
* последний символ буфера (рисунок №1.6).



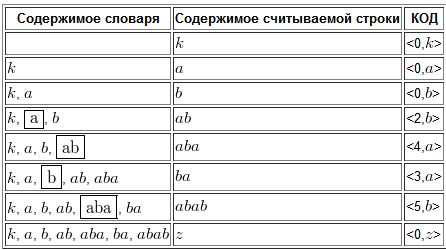
*Рисунок 1.6. Пример алгоритма LZ77: содержимое «kabababababz****»*** *[9]*

В конце итерации алгоритм сдвигает окно на длину, равную длине буфера+1.

***Описание алгоритма LZ78:***

LZ78 ориентируется на данные, которые только будут получены (скользящее окно не используется, а хранится словарь из уже просмотренных фраз). Алгоритм считывает символы сообщения, пока накапливаемая подстрока целиком входит в одну из фраз словаря. Когда данная строка перестанет соответствовать как минимум одной фразе словаря, алгоритм генерирует код, который состоит из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Далее в словарь добавляется введенная подстрока. Если же словарь заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда выдаётся код в виде: < индекс строки в словаре без последнего символа, последний символ >.

Пример алгоритма можно посмотреть на рисунке №1.7.



*Рисунок 1.7. Пример алгоритма LZ78: содержимое «kabababababz» [9]*

#### Алгоритм LZMA

LZMA (Lempel-Ziv-Markov chain Algorithm) - это алгоритм сжатия без потерь.

Основная идея - использование словарного сжатия, выходные данные которого закодированы интервальным кодированием, которые используют сложную модель вычисления вероятности появления каждого бита. Система сжатия находит соответствия, используя словарную структуру данных, и создает поток символов и ссылок фраз, находящихся в словаре, который закодирован одним битом интервальным кодировщиком.

Главной инновацией алгоритма LZMA является то, что вместо общей байтовой модели модель LZMA использует зависящие от контекста битовые поля в каждом представлении букв или фраз. Данная модель почти такая же, как и битовая, но, правда, дает лучший коэффициент сжатия, так как избегает смешивания несвязных битов.

По сравнению с алгоритмом LZ77 алгоритм LZMA имеет следующие преимущества: более высокий коэффициент сжатия, изменяемый размер словаря, небольшие требования по памяти для «распаковки» данных. Однако алгоритм LZMA не на всех типах входных данных работает одинаково эффективно.

***Дельта-кодирование и декодирование:***

Дельта фильтр перестраивает входные данные для эффективного сжатия скользящим окном. Первый байт на выходе совпадает с первым байтом на входе, последующие байты представлены в качестве разности между текущим и предыдущим байтом. Для постоянно изменяющихся данных дельта-кодирование делает работу скользящего окна более эффективной.

#### Кодер:

1. Функция принимает массив и длину массива как аргументы. Если же длина массива передана не была, то массив не обрабатывается.
2. Инициализируются переменные *tmp* для сохранения последнего элемента и*last* для хранения предыдущего числа. Инициализируется цикл.
3. В цикле:

3.1. Сохраняется элемент с индексом i.

3.2. Вычисляется разница между элементом под номером *i* и *i-1* и эта разница перезаписывается в элемент массива с индексом *i*.

***Декодер:***

1. Инициализируется переменная для хранения последнего символа.
2. Инициализируется цикл.
3. В цикле:

3.1. Добавляется к этому элементу значение предыдущего элемента.

3.2. Сохраняется значение текущего элемента.

***Интервальное кодирование:***

При интервальном кодировании все символы сообщения кодируются как одно число, для достижения наилучшего коэффициента сжатия. Это работает эффективно с вероятностями появления символа, которые не являются степенями двойки. Интервальное кодирование работает следующим образом:

1. Выделяется достаточно большой диапазон целых чисел и производится оценка вероятности вхождения для символов.
2. Исходный диапазон чисел делится на поддиапазоны, размер которых пропорционален вероятности вхождения символа, за который они отвечают.
3. Каждый символ сообщения кодируется, затем диапазон сокращается до размера диапазона только что закодированного символа и опять делится по вероятностям.

В декодере должны быть такие же распределения вероятностей, как и при кодировании.

Рассмотрим пример: [10]

Кодируется строка «*abehhilopsu»*. Для начала данная строка пропускается через дельта фильтр. Тогда исходная строка  примет вид: *«a 1 3 3 0 1 3 3 1 3 2».*

Теперь в строке вместо десяти различных символов только пять различных символов. Далее получившейся строке применяется метод «скользящего» окна (рисунок №1.7):



*Рисунок 1.7. Пример алгоритма LZMA: применение скользящего окна*

Получается код: *«00a001003110431212$»*, где *$* - символ конца сообщения.

Для строки получается диапазон  и распределение вероятностей:

Таким образом, произведя интервальное кодирование строки, получается диапазон, который отвечает закодированному сообщению.

## 1.5 Основные программы-архиваторы

Наиболее популярными программами-архиваторами на данный момент для операционной системы Windows являются:

* WinRar
* WinZip
* 7-zip

1. WinRar поддерживает форматы архивации файлов и сжатия данных без потерь Zip и Rar.

В формате Zip используется алгоритм сжатия Deflate, который использует комбинацию алгоритмов LZ77 и Хаффмана (Huffman codes). LZ77 – алгоритм из семейства алгоритмов Лемпеля-Зива (Lempel-Ziv).

В формате Rar используется алгоритм сжатия LZSS – алгоритм Лемпеля-Зива-Сторрера-Сжимански (Lempel-Ziv-Storer-Szymanski). Данный алгоритм является версией алгоритма LZ77.

1. WinZip поддерживает форматы архивации файлов и сжатия данных без потерь Zip и ZipX.

Формат ZipX поддерживается только в относительно новых версиях программы. В данном формате помимо алгоритма сжатия Deflate также могут применяться методы BZip, LZMA, PPMd, JPEG, WavPack. По этой причине файлы могут не распознаваться классическими программами распаковки Zip, в которых реализован лишь метод [Deflate](https://ru.wikipedia.org/wiki/Deflate).

1. 7-Zip поддерживает форматы архивации файлов и сжатия данных без потерь 7z и Zip.

В формате 7z могут применяться алгоритмы сжатия без потерь LZMA, LZMA2, PPMd, BZip2. В основном же используется алгоритм LZMA/LZMA2 (Lempel-Ziv-Markov chain Algorithm).

Программа-архиватор tar - наиболее распространенный архиватор, используемый в Linux-системах. tar не является архиватором в привычном понимании этого слова, т.к. он самостоятельно не использует сжатие. В то же время, некоторые программы-архиваторы не умеют сжимать несколько файлов. Поэтому чаще всего данные программы используются вместе. tar создает несжатый архив, в который помещаются выбранные файлы. После этого полученный файл \*.tar сжимается архиватором, например, gzip. Программа-архиватор gzip для сжатия данных использует алгоритм Deflate.

Используя вышеуказанную информацию, приведена таблица №1.1 для наглядного представления каждого из исследованных программных средств:

|  |  |  |
| --- | --- | --- |
| Программа-архиватор | Форматы архивации | Алгоритмы сжатия данных без потерь |
| WinRar | Rar | LZSS (LZ77) |
| Zip | Deflate (LZ77 + Huffman codes) |
| WinZip | Zip | Deflate (LZ77 + Huffman codes) |
| ZipX | Deflate, LZMA, BZip, PPMd, JPEG, WavPack |
| 7-Zip | 7z | LZMA, LZMA2, PPMd, BZip2 |
| tar + gzip | TGZ | Deflate (LZ77 + Huffman codes) |

*Таблица 1.1 Описание*

## 

## 1.5.1 Исследование эффективности алгоритмов сжатия данных без потерь

Для исследования эффективности по времени работы и по размеру результирующих данных относительно размера оригинальных данных были проделаны следующие действия: была взята папка с большим количеством файлов разного формата - 4988 файлов общим размером **106** мегабайт, к данной папке были применены различные программы-архиваторы, используя максимально возможный уровень сжатия. Результат исследования описан в таблице №1.2:

|  |  |  |  |
| --- | --- | --- | --- |
| Программа-архиватор | Форматы архивации и алгоритм сжатия данных без потерь | Время выполнения, с | Результирующий размер файла, мб |
| WinRar | Rar (LZSS) | ≈ 31 | 52  (49% от оригинального размера) |
| Zip (Deflate) | ≈ 16,5 | 57,1  (53,9% от оригинального размера) |
| WinZip | Zip (Deflate) | ≈ 37 | 56,6  (53,4% от оригинального размера) |
| 7-Zip | 7z (LZMA) | ≈ 46 | 23,3  (22% от оригинального размера) |
| 7z (LZMA2) | ≈ 55 | 23,3  (22% от оригинального размера) |
| 7z (BZip2) | ≈ 120 | 51,4  (48,5% от оригинального размера) |
| 7z (PPMd) | ≈ 61 | 48,3  (45,6% от оригинального размера) |

*Таблица 1.2 Таблица эффективности*

По результатам исследования можно сделать следующие выводы:

1. Меньше всего времени на проделанные действия было затрачено программой WinRar при использовании алгоритма сжатия Deflate.
2. Наилучший показатель процентного отношения результирующего размера данных от оригинального размера данных показала программа-архиватор 7-Zip, используя алгоритмы LZMA/LZMA2. При использовании алгоритма LZMA время на выполнение действий было затрачено меньше.

## 1.5.2 Статистика использования программ-архиваторов

Проведя опрос среди студентов ФПМИ на тему предпочтения по использованию программ-архиваторов на ОС Windows, сложилась следующая ситуация (таблица №1.3):

|  |  |
| --- | --- |
| Программа-архиватор | Количество голосов, % |
| WinRar | ≈ 76,5 |
| 7-Zip | ≈ 23,5 |
| WinZip | 0 |

*Таблица 1.3 Результаты опроса*

WinRar действительно считается самой популярной программой для сжатия данных среди русскоязычных пользователей, но данная программа является платной.

7-Zip же - полностью бесплатное программное средство и поэтому считается самым лучшим бесплатным архиватором.

WinZip, как и WinRar, тоже является платным программным средством. К слову, стоимость его больше (относительно WinRar), а период бесплатного использования почти в 2 раза меньше. Неудивительно, что данное программное средство не получило поддержку в опросе среди студентов БГУ

Данные слова подкрепляются оценками с интернет-источника [11] (таблица №1.4):

|  |  |
| --- | --- |
| Программа-архиватор | Оценка пользователей на сайте [12] |
| WinRar | 9 |
| 7-Zip | 8.3 |
| WinZip | 8 |

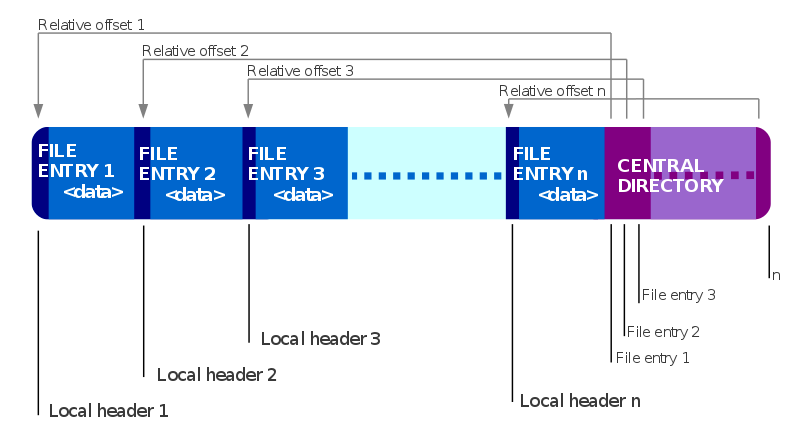
*Таблица 1.4 Рейтинг*

## 

## 1.6. Исследование структуры файла

Для исследования был взят формат файла Zip [12].

Внутренний макет Zip-файла выглядит следующим образом (рисунок №1.8):



*Рисунок 1.8. Внутренний макет Zip-файла*

Как видно из рисунка, весь каталог Zip-файла находится в конце файла, а каждый внутренний файл имеет собственный заголовок, который описан в таблице №1.5:

|  |  |  |
| --- | --- | --- |
| Offset | Байты | Описание |
| 0 | 4 | Локальная подпись заголовка = 0x04034b50 |
| 4 | 2 | Версия, необходимая для извлечения (минимум) |
| 6 | 2 | Битовый флаг общего назначения |
| 8 | 2 | Метод сжатия |
| 10 | 2 | Время последнего изменения файла |
| 12 | 2 | Дата последнего изменения файла |
| 14 | 4 | CRC-32 |
| 18 | 4 | Сжатый размер |
| 22 | 4 | Несжатый размер |
| 26 | 2 | Длина имени файла (*n*) |
| 28 | 2 | Длина дополнительного поля (*m*) |
| 30 | *n* | Имя файла |
| 30+*n* | *m* | Дополнительное поле |

*Таблица 1.5 Структура заголовка Zip-файла*

По результатам данного исследования в дальнейшем будет описана собственная структура файла, которая используется в программном средстве.

## ГЛАВА 2. ПРОГРАММНОЕ СРЕДСТВО

## 2.1 Инструментарий

Для разработки программного средства использовался язык программирования C# (C Sharp).

Для создания исполняемых файлов использовалась среда разработки Microsoft Visual Studio 2015. [13]

Для создания пользовательского интерфейса использовалась система Windows Presentation Foundation (WPF). [14]

Встроенный отладчик в Microsoft Visual Studio использовался для наблюдения за пошаговым выполнением программы при поиске ошибок на этапе разработки.

## 2.2 Основные требования

Для использования программного средства необходим компьютер с операционной системой Windows (64-х разрядной) не ранее версии «Windows 7» и со следующими минимальными характеристиками:

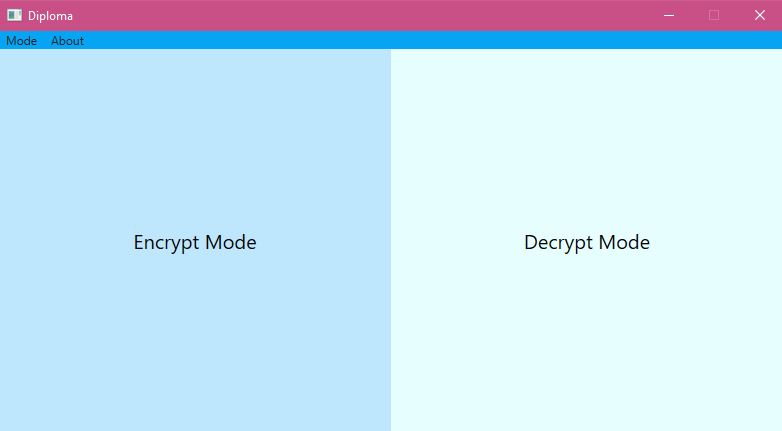
* оперативна память 2 ГБ
* 64-разрядный (x64) процессор с тактовой частотой 1 ГГц

Пользователь для работы с программным средством должен обладать, по крайней мере, средним уровнем знаний пользования компьютером.

Программное средство должно располагаться на жестком диске компьютера.

## 2.3 Поведение программного средства

После запуска программного средства открывается главное окно (рисунок №2.1):



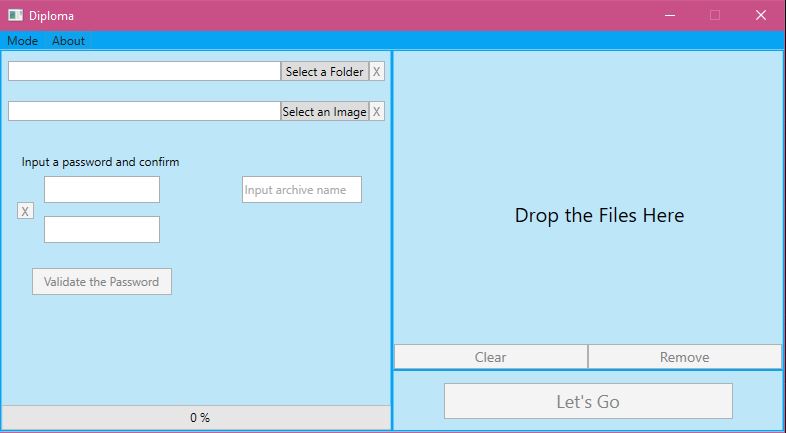
*Рисунок 2.1. Главное окно программного средства*

Главное окно программного средства позволяет пользователю выбрать режим, в котором он желает продолжить процесс, а именно:

* Encrypt Mode – режим, который позволит пользователю защитить и замаскировать свои данные;
* Decrypt Mode – режим, который позволит пользователю снять защиту с защищенных и замаскированных данных.

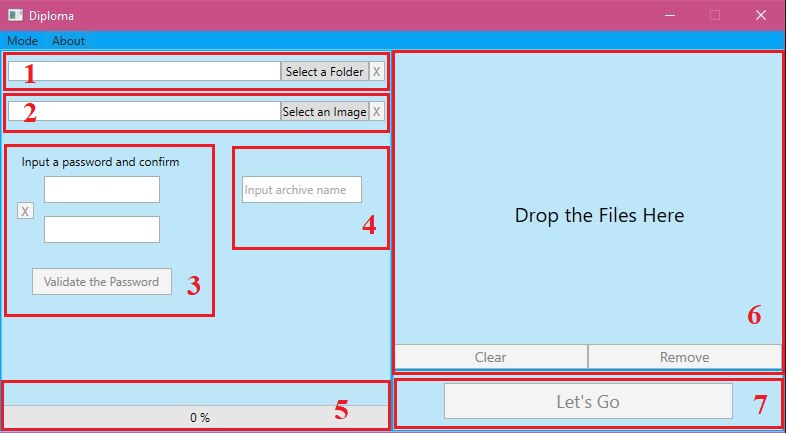
## 2.3.1 Описание «Encrypt Mode»

Пойдем по порядку, поэтому рассмотрим сначала «Encrypt Mode». При выборе данного режима пользователю будет отображен следующий интерфейс (рисунок №2.2):



*Рисунок 2.2. Окно «Encrypt Mode» программного средства*

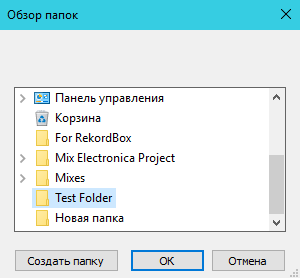
Стоит обратить внимание, что окно данного режима состоит из определенного числа составляющих, которые отвечают определенную логику в процессе выполнения программного средства. На рисунке №2.3 можно увидеть эти важные составляющие:



*Рисунок 2.3. Окно «Encrypt Mode»: главные составляющие*

Ниже приведено подробное описание каждой составляющей окна «Encrypt Mode» программного средства и поведение программы в различных ситуациях:

1. В данной области при нажатии на кнопку «Select a Folder» открывается окно (рисунок №2.4), в котором пользователю предоставляется возможность выбрать папку, по пути которой будет создан замаскированный файл с защищенными данными. Выбранный путь папки отобразится в соответствующем поле (рисунок №2.5).



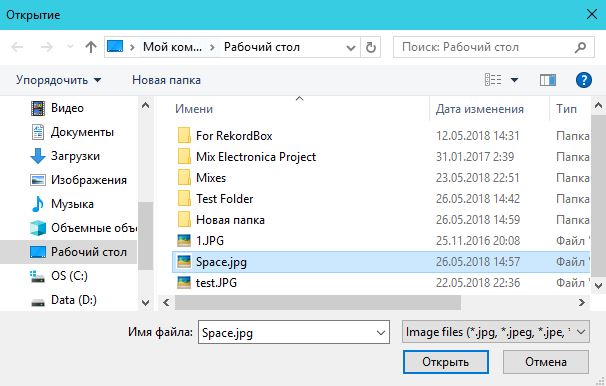
*Рисунок 2.4. Окно для выбора папки*

*C:\Users\User\Desktop\Новая папка\5.JPG*

*Рисунок 2.5. Отображение пути выбранной папки*

Кнопка «X» позволяет пользователю удалить выбранное решение, а кнопка «Select a Folder» в любой момент позволит пользователю изменить его (решение).

1. В данной области при нажатии на кнопку «Select an Image» открывается окно (рисунок №2.6), в котором пользователю предоставляется возможность выбрать изображение, с помощью которого будут замаскированы защищенный данные. Выбранный путь изображения отобразится в соответствующем поле (рисунок №2.7).



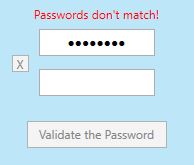
*Рисунок 2.6. Окно для выбора изображения*

*C:\Users\User\Desktop\Новая папка\7.JPG*

*Рисунок 2.7. Отображение пути выбранного изображения*

Кнопка «X», как и в предыдущем случае, позволяет пользователю удалить выбранное решение, а кнопка «Select an Image» в любой момент позволит пользователю выбрать другое изображение.

1. Данная область предоставляет пользователю возможность ввести и подтвердить пароль, с помощью которого в дальнейшем будут защищены его данные. Пользователю необходимо ввести пароль два раза в соответствующие поля, а затем нажать на кнопку «Validate the Password» (которая становится доступной только после заполнения полей для паролей), после чего программное средство определит, совпадают ли пароли и соответствует ли пароль все известным правилам. В случае, если пароли не совпадают, пользователь увидит соответствующее сообщение, а поле для подтверждения пароля будет очищено (рисунок №2.8).



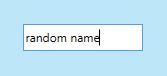
*Рисунок 2.8. Пароли не совпадают*

При успешном же вводе пароля пользователь также увидит соответствующее сообщение, а поля для ввода паролей и кнопка «Validate the Password» будут заблокированы. Но будет разблокирована кнопка «X», с помощью которой пользователь сможет запустить заново процесс ввода пароля для защиты данных от несанкционированного использования (рисунок №2.9).

**

*Рисунок 2.9. Пароль принят*

1. В данной области пользователь вводит имя, которое примет замаскированный файл с защищенными данными (рисунок №2.10).



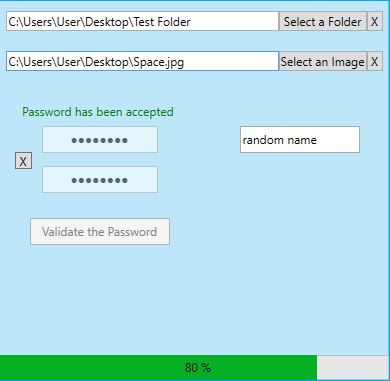
*Рисунок 2.10. Ввод имени для* замаскированного файла с защищенными данными

1. В данной области представлен так называемый «Status Bar», который отображает значение в процентном соотношении (от 0% до 100%), сколько шагов пользователей выполнил и сколько шагов осталось для того, чтобы он мог начать процесс защиты и маскировки.

В «Encrypt Mode» основными для заполнения являются пять пунктов (правда, некоторые из них уже были описаны выше):

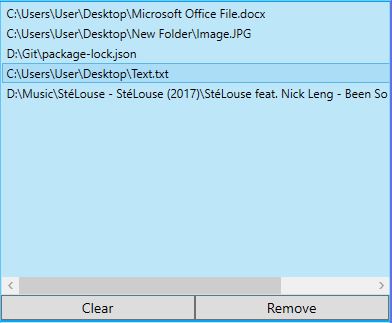
* выбор папки, по пути которой будет создан замаскированный файл с защищенными данными;
* выбор изображения, с помощью которого будут замаскированы защищенные данные;
* ввод пароля для защиты данных;
* ввод имени файла, которое примет по итогу замаскированный файл с защищенными данными;
* выбор данных для защиты от несанкционированного использования.

В итоге, по мере заполнения каждого пункта, «Status Bar» будет увеличивать свое значение (или же уменьшать, если пользователь будет откатывать назад свои действия). К примеру, при заполненных уже четырех из пяти пунктов, описанных выше, пользователь будет видеть следующее (рисунок №2.11):



*Рисунок 2.11. Состояние программы при заполненных четырех пунктах*

1. В данной области пользователю предоставляется возможность методом «Drag-and-drop» [15] переместить файлы, которые он бы хотел защитить и замаскировать. Пути всех выбранных файлов будут отображены в соответствующем окне в виде списка (рисунок №2.12).

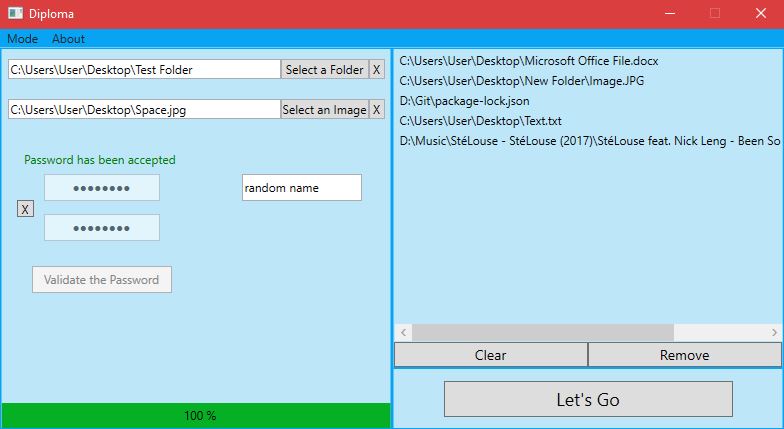


*Рисунок 2.12. Отображение выбранных файлов для защиты и маскировки*

Также пользователь может очистить список выбранных файлов, нажав на кнопку «Clear» или же удалить конкретный файл, выбрав его и нажав на кнопку «Remove». Эти кнопки заблокированы по умолчанию, но при добавлении, по крайней мере, одного файла они будут разблокированы.

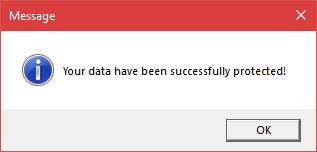
1. В заключительной области окна «Encrypt Mode» располагается кнопка «Let’s Go», с помощью которой пользователь сможет запустить процесс защиты и маскировки данных. По умолчанию данная кнопка заблокирована, но после того как пользователь заполнит все пункты и «Status Bar» отобразит значение в 100%, эта кнопка будет разблокирована.

После выполнения всех шагов пользователь будет наблюдать картину, представленную на рисунке №2.13, и после нажатия на кнопку «Let’s Go» будет запущен процесс защиты и маскировки данных. По завершению данного процесса программное средство также удалит исходные файлы, чтобы на жестком диске пользователя его чувствительные файлы не находились больше в незащищенном виде.



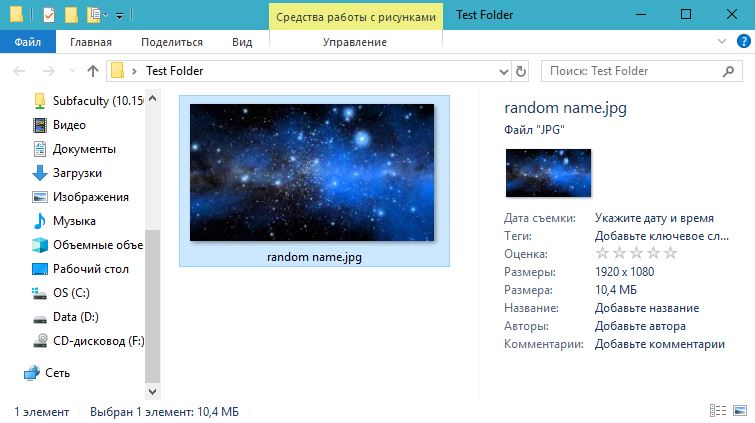
*Рисунок 2.13. Окно «Encrypt Mode»: все пункты заполнены*

В конце, когда программа выполнит все действия, пользователю будет выдано соответствующее сообщение (рисунок №2.14).



*Рисунок 2.14. Сообщение об окончании работы программного средства*

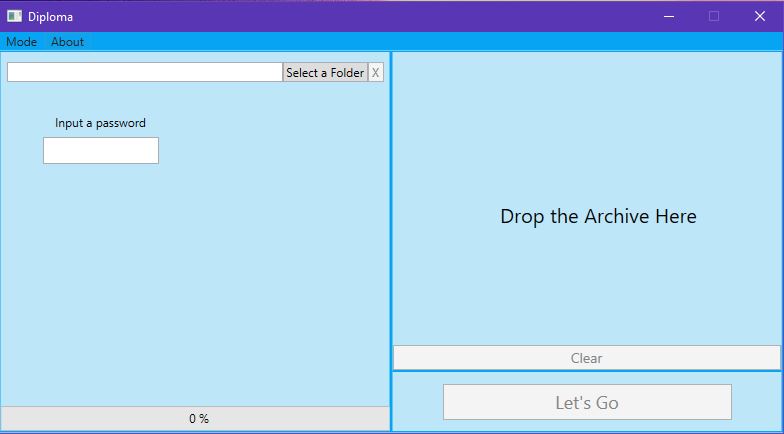
Зайдя в выбранную в первой области папку, пользователь обнаружит изображение, которое он выбирал во второй области программного средства (он также сможет открыть это изображение) (рисунок №2.15). Но данный файл изображения будет только казаться таким простым, так как на самом деле внутри него будут «вшиты» зашифрованные и сжатые (некоторые их них) данные пользователя, которые он выбирал в шестой области программного средства окна «Encrypt Mode».



*Рисунок 2.15. Файл с защищенными данными*

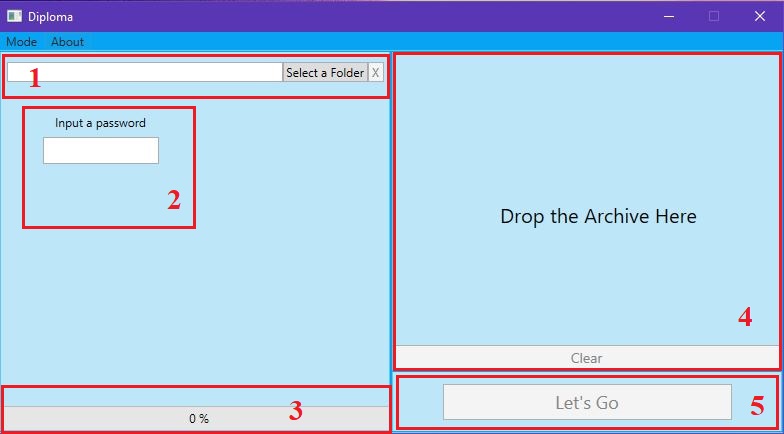
## 2.3.2 Описание «Decrypt Mode»

Теперь пришло время рассмотреть «Decrypt Mode». Выглядит он следующим образом (рисунок №2.16):



*Рисунок 2.16. Окно «Decrypt Mode» программного средства*

В данном режиме определенно меньше логических цепочек, так как и главных составляющих так же меньше. На рисунке №2.17 представлены эти составляющие.



*Рисунок 2.17. Окно «Decrypt Mode»: главные составляющие*

Ниже приведено описание каждой составляющей окна «Decrypt Mode» программного средства и поведение программы в различных ситуациях:

1. В этой области при нажатии на кнопку «Select a Folder» открывается окно, в котором пользователь сможет выбрать папку, в которой он хотел бы увидеть данные в оригинальном виде (с них будет снята защита).

Кнопка «X», как и в окне «Encrypt Mode», позволит пользователю отменить свой выбор.

1. В данной области пользователь вводит пароль от защищенных данных
2. В этой области располагается знакомый уже «Status Bar». В данном режиме основными для заполнения являются три пункта:

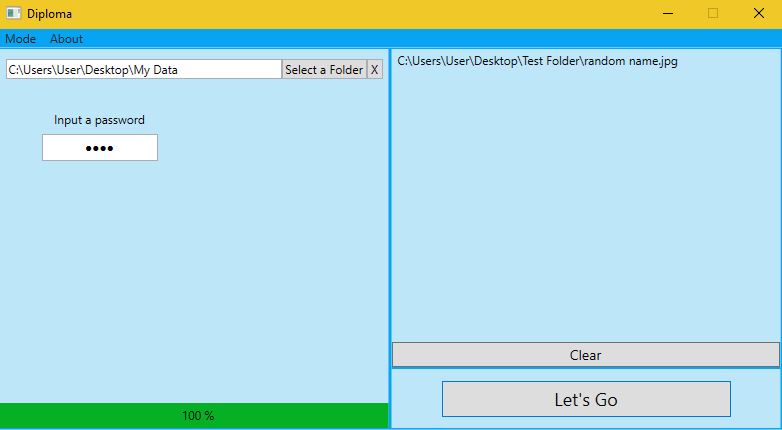
* выбор папки, в которую будут помещены файлы после успешного прохождения процесса снятия защиты;
* ввод пароля для снятия защиты с данных;
* выбор файла с защищенными данными, с которых будет снята защита.

1. В данной области пользователь методом «Drag-and-drop» перемещает файл (и он может быть только один), который содержит защищенные данные. Путь выбранного файла будет отображен в соответствующем окне.

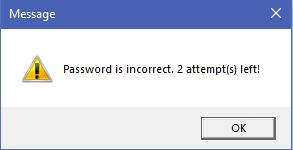
Также пользователь может отменить свой выбор с помощью кнопки «Clear».

1. В заключительной области окна «Decrypt Mode» располагается кнопка «Let’s Go», с помощью которой пользователь сможет запустить процесс снятия защиты. По умолчанию данная кнопка заблокирована, но после того как пользователь заполнит все пункты и «Status Bar» отобразит значение в 100%, эта кнопка будет разблокирована.

После выполнения всех шагов программное средство пользователя будет выглядеть, как представлено на рисунке №2.18. По нажатию на кнопку «Let’s Go» программное средство запустит процесс проверки пароля. Если пароль окажется неправильным, то программа выдаст соответствующее сообщение (рисунок №2.19) и приостановит свои действия.

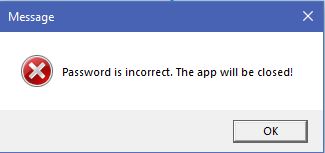


*Рисунок 2.18. Окно «Decrypt Mode»: все пункты заполнены*



*Рисунок 2.19. Сообщение «Пароль неверен»*

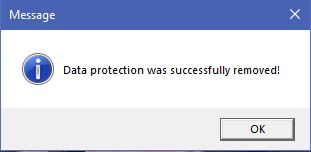
У пользователя есть только три попытки, чтобы введенный пароль оказался верным, иначе же, после трех неверно введенных паролей, программное средство выдаст также соответствующее сообщение (рисунок №2.20) и будет закрыто.



*Рисунок 2.20. Сообщение «Пароль неверен. Программа будет закрыта»*

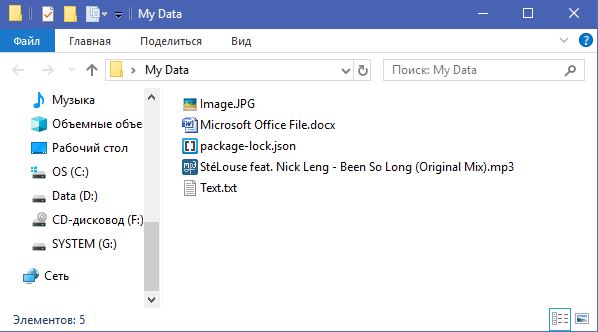
При правильно введенном пароле будет запущен процесс снятия защиты с данных. По завершению данного процесса программное средство удалит файл с защищенными данными, чтобы не занимать лишнего места на жестком диске пользователя.

В конце, когда программа выполнит все действия, пользователю будет выдано соответствующее сообщение (рисунок №2.21).



*Рисунок 2.21. Сообщение об окончании работы программного средства*

Зайдя в выбранную в первой области папку, пользователь увидит те самые файлы, которые он выбирал для защиты в шестой области в режиме защиты данных (рисунок №2.22).



*Рисунок 2.22. Оригинальные файлы, с которых была снята защита*

## 2.3.3 Описание «Menu Bar»

Описав основные моменты, затронем ещё и «Menu Bar», который содержит следующие параметры:

* + «Mode» - позволяет пользователю переключаться между режима в любое время;
  + «About» - отображает информативную информацию о разработчике.

## 2.4 Описание программного средства с технической стороны.

После ознакомления с поведением программного средства, могут возникнуть вопросы типа:

* «А что именно происходит по нажатию на кнопки «Let’s Go» в режимах «Encrypt Mode» и «Decrypt Mode»?»:
* «А как программное средство определяет правильность введенного пароля для снятия защиты с данных?»;
* «А безопасно ли хранятся пароль и данные?»
* «Как происходит маскировка?»

На эти и, возможно, другие вопросы постарается ответить текст, приведенный ниже.

## 2.4.1 Описание процесса защиты данных

В окне «Encrypt Mode» пользователь вводит пароль – обозначим его *password.* Также пользователь выбирает данные для защиты – список файлов – обозначим данный список *files*.

После запуска процесса защиты данных происходит следующее:

* вычисляется хэш (*h1*) от *password* с помощью алгоритма хэширования SHA-256 [16]. Данное значение будет использоваться в генерации итогового ключа для шифрования данных.
* для каждого элемента в списке файлов *files* программное средство:
  + - сжимает данные с помощью алгоритма сжатия данных без потерь LZMA. Тут нужно также добавить, что не все типы файлов подвергаются сжатию, а только те, к которым процесс сжатия данных не бесполезен.
    - затем данные шифруются симметричным алгоритмом шифрования AES-256 [17] с помощью ключа, который генерируется на основе значения *h1* и соли *salt*, которая была сгенерирована случайным образом. Также при шифровании применяется режим сцепления блоков CBC [18].

Можно заменить, что данные действительно хранятся безопасно, так проходят этап шифрования. Но что там с паролем?

Для того чтобы более детально понять данный момент и упросить понимание ответов на остальные вопросы, ниже будет приведено подробное описание структуры файла, который получается после прохождения процесс защита и маскировки данных.

## 2.4.2 Структура выходного файла

Ещё до начала процесса защиты данных и перед тем как начать процесс шифрование, программное средство вычисляет *h2* – хэш функцию от *h1* при помощи алгоритма SHA-256, тем самым не храня в выходном файле значение, которое является основным при генерации ключа для шифрования данных.

В таблице №2.1 представлена структура основного блока выходного файла

|  |  |  |
| --- | --- | --- |
| Offset | Байты | Описание |
| 0 | im | Изображение, с помощью которого маскируются данные |
| im | 32 | *h2* – хэш *h1* |
| im + 32 | 8 | *salt* |
| im +40 | D | Блок c данными, содержащий информацию о названии файла и сам файл |
| im +40+D | 4 | Длина изображения (*im*), с помощью которого маскируются данные |

*Таблица 2.1 Структура выходного файла. Основной блок*

Блок c данными *D* – блок, содержащие в себе все файлы, которые были выбраны пользователем для защиты. Соответственно, данный блок содержит в себе столько повторяющихся типов параметров, сколько файлов для защиты выбрал пользователь.

Таким образом, в таблице №2.2 более детально расписан блок c данными *D* для одного файла:

|  |  |  |
| --- | --- | --- |
| Offset | Байты | Описание |
| 0 | 4 | Длина имени файла (*n*) |
| 4 | n | Зашифрованное имя файла |
| 4 + n | M | Сжатый и зашифрованный файл |
| 4 + n + M | 2 | Индекс окончания файла |

*Таблица 2.2 Структура выходного файла. Блок с данными*

Файл *M* также может содержать в себе циклическую информацию, так как для считывания файла используется блок константной длинны. И если данный блок меньше по размеру, чем общий размер файла, то количество повторяющихся типов параметров будет равно значению, во сколько раз блок константной длины меньше общего размера файла + 1.

В таблице №2.3 расписан один блок файла *M*:

|  |  |  |
| --- | --- | --- |
| Offset | Байты | Описание |
| 0 | 2 | Индекс не окончания файла |
| 2 | 4 | Длина сжатого и зашифрованного блока файла (*len*) |
| 6 | len | Сжатый и зашифрованный блок файла |

*Таблица 2.3 Структура выходного файла. Зашифрованный (и сжатый) файл*

Описав структуру выходного файла, можно ответить на некоторые вопросы

* + - Данные действительно хранятся безопасно, так как проходят процесс шифрования
    - Пароль, который непосредственно участвует в генерации ключа для шифрования, не хранится в выходном файле, а хранится его хэш-значение. А, как известно, по хэш-значению восстановить исходное значение невозможно.
    - Так как в начало выходного файла записывается изображение, которое предназначено для маскировки, и итоговое расширение защищенного файла равно расширению данного изображение, то система Windows способна распознать выходной файл как обычное изображение. Данным способ и производится маскировка защищенных данных.

## 2.4.3 Описание процесса снятия защиты с данных и проверки пароля на достоверность

В окне «Decrypt Mode» пользователь вводит пароль для снятия защиты данных – обозначим его *passwordForDecrypt.* Также пользователь выбирает файл, в котором хранятся его защищенные данные.

Зная уже структуру файла с защищенными данными, очень легко понять, как именно происходит проверки пароля на достоверность:

* + - Программное средство считывает 4 байта с конца файла и узнает длину изображения (*imageLen*), которое хранится вначале файла
    - Программное средство вычисляет хэш-значение (*hh1*) от *passwordForDecrypt*, а затем ещё раз хэш-значение (*hh2*) от *hh1*
    - Далее программное средство пропускает количество байтов = *imageLen* с начала файла и считывает следующие 32 байта, в которых хранится значение *h2*.
    - Далее происходит сравнение *h2* и *hh2*. Если значения равны, то введенный пароль *passwordForDecrypt* является достоверным.

Соответственно, после успешной проверки пароля на достоверность, запускается процесс снятия защиты с данных, который, в какой-то степени, является обратным процесс защиты данных:

* + - Происходит считывание значения *salt,* после чего вместе со значением *hh1*происходит генерация ключа для дешифрования данных.
    - Для каждого файла, находящемся в зашифрованном виде в выходном файле, на основе описанной выше структуры выходного файла, применяются следующие действия:
      * Определяется исходное имя файла
      * Выполняется процесс дешифрования файла на AES-256
      * По имени файла определяется его формат, а после, нужно ли к дешифрованным данным применять и процесс декомпресии
      * В положительном случае выполняется процесс декомпрессии по алгоритму LZMA

Описав все эти моменты, надеюсь, ответы на основные вопросы были предоставлены, и понимание того, как «под капотом» работает программное средство теперь так же более очевидно.

## 2.5 Сравнение разработанного программного средства с уже существующими

Безусловно, пытаться доказать, что данное программное средств будет работать эффективнее, чем всем известные 7-Zip или же WinRar – бесполезно, и это не будет правдой. Но стоит отметить один момент: в разработанном программном средстве реализован фильтр, который позволяет указать, к каким именно типом файлов не нужно применять процесс сжатия, так как данный процесс не принесет никакой пользы, а только затратит больше времени.

Например, алгоритм LZMA сжимает всем известный формат аудио «mp3» всего лишь на 1% от общего размера, что является в какой-то степени бесполезным действием, но программа 7-Zip тратит на это определенное количество времени. А если, допустим, нужно защитить и сжать большой «txt» файл (а к файлам данного типа процесс сжатия очень эффективен) и большой «mp3» файл? Программное средство 7-Zip при определенных настройках будет производить процесс сжатия и шифрования для обоих файлов, а вот разработанное программное средство в случае «mp3» файла опустит процесс сжатия и только зашифрует его, но вот к «txt» файлу будут применены оба процесса.

Также стоит вспомнить о процессе маскировки, который уже не раз упоминался. Как-никак, но это один из этапов защиты данных от несанкционированного использования, чего, лично я, не встречал ни у одной из существующих программ.

## ЗАКЛЮЧЕНИЕ

В данной работе:

* Рассмотрены каналы несанкционированного получения информации
* Рассмотрены методы защиты программ и данных от несанкционированного использования
* Рассмотрены алгоритмы для сжатия данных без потерь
* Произведен анализ эффективности алгоритмов сжатия данных без потерь
* Разработано программное средство, предназначенное для защиты данных от несанкционированного использования
* Произведено описание поведения программного средства и произведено описание структуры выходного файла с защищенными данными

Также хочется добавить, что данное программное средство действительно может способствовать защите данных от несанкционированного использования. В первую очередь – данные защищены шифрованием, что усложняет процесс получения исходных данных. Во-вторых, как-никак, но происходит маскировка данных с помощью изображения. Ведь по-хорошему, обычное изображение вряд ли способно привлечь внимание и заставить думать о том, что в нем хранится какая-либо ещё информация. Ну а сжатие данных для определенных форматов файлов – это приятный бонус для возможности экономия места на жестком диске пользователя.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ожегов, С.И. Словарь русского языка / С.И. Ожегов — М. «Русский язык», 1984 — 816 с.
2. Данные: [Электронный ресурс]. — Режим доступа: <https://ru.wikipedia.org/wiki/Данные>. — Дата доступа: 30.03.2018.
3. Емельянова, Н.3. Защита информации в персональном компьютере: учебное пособие / Н.3. Емельянова, Т.Л. Партыка, И.И. Попов. — М. ФОРУМ, 2009. — 368 с.
4. Моисеенко, Е.В. Информационные технологии в экономике / Е.В. Моисеенко, Е.Г. Лаврушина. — Владивосток: Изд-во ВГУЭС, 2004 — 246 с.
5. Пятничный формат: Как писать код, который никто не сможет сопровождать: [Электронный ресурс]. — Режим доступа: <http://habra2habr.ru/company/friifond/blog/268063>. — Дата доступа: 02.10.2015.
6. Invisible JS obfuscator: [Electronic resource]. — Mode of access: <http://www.freedomscripts.org/js-invis.html>. — Date of access: 27.06.2014.
7. aaencode - Encode any JavaScript program to Japanese style emoticons (^\_^): [Electronic resource]. — Mode of access: <http://utf-8.jp/public/aaencode.html>. — Date of access: 17.10.2013.
8. Алгоритмы, используемые при сжатии данных: [Электронный ресурс]. — Режим доступа: <https://habrahabr.ru/post/132289/>. — Дата доступа: 09.11.2011.
9. Алгоритмы LZW, LZ77 и LZ78: [Электронный ресурс]. — Режим доступа: <https://habrahabr.ru/post/132683/>. — Дата доступа: 15.11.2011.
10. Алгоритм LZMA: [Электронный ресурс]. — Режим доступа: <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_LZMA>. — Дата доступа: 21.12.2016.
11. Обзор программ-архиваторов: [Электронный ресурс]. — Режим доступа: <https://www.softhome.ru/article/luchshie-arhivatory-dlya-windows>. — Дата доступа: 01.03.2014.
12. Zip-file: [Electronic resource]. — Mode of access: <https://en.wikipedia.org/wiki/Zip_(file_format)>. — Date of access: 16.05.2018.
13. Microsoft Visual Studio: [Electronic resource]. — Mode of access: https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide. — Date of access: 02.05.2018.
14. Windows Presentation Foundation: [Electronic resource]. — Mode of access: <https://msdn.microsoft.com/en-us/library/aa663364.aspx>. — Date of access: 20.05.2018.
15. Drag-and-Drop: [Electronic resource]. — Mode of access: <https://techterms.com/definition/drag_and_drop>. — Date of access: 13.09.2011.
16. SHA: [Electronic resource]. — Mode of access: <https://www.cryptocompare.com/coins/guides/how-does-a-hashing-algorithm-work>. — Date of access: 20.05.2018.
17. AES: [Electronic resource]. — Mode of access: <https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm>. — Date of access: 21.04.2017.
18. Block Ciphers Modes of Operation: [Electronic resource]. — Mode of access: <http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.html>. — Date of access: 12.10.2015.

## ПРИЛОЖЕНИЯ

## ПРИЛОЖЕНИЕ А

Листинг «Main Window»

<Window x:Class="Diploma.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:dd="urn:gong-wpf-dragdrop"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Diploma"

mc:Ignorable="d"

Title="Diploma" Height="440" Width="800" ResizeMode="CanMinimize">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="1\*"/>

</Grid.ColumnDefinitions>

<Menu VerticalAlignment="Top" Grid.ColumnSpan="2" Background="#FF06A4F3">

<MenuItem Header="Mode">

<MenuItem Header="Encrypt Mode" Click="encryptModeMenuItem\_Click"/>

<MenuItem Header="Decrypt Mode" Click="decryptModeMenuItem\_Click"/>

</MenuItem>

<MenuItem Header="About" Click="aboutMenuItem\_Click"/>

</Menu>

<Button x:Name="encryptModeButton" Content="Encrypt Mode" HorizontalAlignment="Center" Margin="0,18,0,0" VerticalAlignment="Center" Cursor="Hand" Height="393" Width="397" FontSize="20" Background="#FFE7FEFF" Click="encryptModeButton\_Click" />

<Button x:Name="decryptModeButton" Content="Decrypt Mode" HorizontalAlignment="Center" Margin="0,18,0,0" VerticalAlignment="Center" Cursor="Hand" Height="393" Width="397" FontSize="20" Grid.Column="1" Click="decryptModeButton\_Click" Background="#FFE7FEFF" />

<Frame x:Name="MainFrame" Margin="0,18,0,0" HorizontalAlignment="Left" Width="784" Grid.ColumnSpan="2" Height="382" NavigationUIVisibility="Hidden"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using WinForms = System.Windows.Forms;

using System.Windows.Controls;

using System.IO;

using Diploma.crypto;

namespace Diploma

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

WindowStartupLocation = WindowStartupLocation.CenterScreen;

InitializeComponent();

}

private void encryptModeButton\_Click(object sender, RoutedEventArgs e)

{

encryptModeButton.Visibility = Visibility.Hidden;

decryptModeButton.Visibility = Visibility.Hidden;

MainFrame.Content = new EncryptPage();

}

private void decryptModeButton\_Click(object sender, RoutedEventArgs e)

{

encryptModeButton.Visibility = Visibility.Hidden;

decryptModeButton.Visibility = Visibility.Hidden;

MainFrame.Content = new DecryptPage();

}

private void aboutMenuItem\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("© 2018 Faminchik Evgeniy, BSU FAMCS", "About", MessageBoxButton.OK, MessageBoxImage.Information);

}

private void encryptModeMenuItem\_Click(object sender, RoutedEventArgs e)

{

MainFrame.Content = new EncryptPage();

}

private void decryptModeMenuItem\_Click(object sender, RoutedEventArgs e)

{

MainFrame.Content = new DecryptPage();

}

}

}

## ПРИЛОЖЕНИЕ Б

Листинг «Encrypt Mode»

<Page x:Class="Diploma.EncryptPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Diploma"

mc:Ignorable="d"

d:DesignHeight="382" d:DesignWidth="800"

Title="EncryptPage">

<Grid Background="#FFBEE6F9">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"></ColumnDefinition>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="382"></RowDefinition>

</Grid.RowDefinitions>

<Border BorderBrush="#FF06A4F3" BorderThickness="1.5" >

<StackPanel MinWidth="100">

<DockPanel Margin="6,10,6,10">

<TextBox x:Name="folderNameTB" Height="20" Width="273" HorizontalAlignment="Left" HorizontalContentAlignment="Stretch" VerticalContentAlignment="Center" IsReadOnly="True" Margin="0" TextChanged="folderNameTB\_TextChanged"/>

<Button x:Name="selectFolderButton" Content="Select a Folder" HorizontalAlignment="Right" Click="selectFolderButton\_Click" Height="20" Width="88" Margin="0" BorderBrush="#FFABADB3"/>

<Button x:Name="resetSelectedFolderButton" Content="X" HorizontalAlignment="Right" Margin="-18,0,0,0" Width="16" IsEnabled="False" Click="resetSelectedFolderButton\_Click" BorderBrush="#FFABADB3"/>

</DockPanel>

<DockPanel Margin="6,10,6,20">

<TextBox x:Name="imageNameTB" Height="20" Width="273" HorizontalAlignment="Left" HorizontalContentAlignment="Stretch" VerticalContentAlignment="Center" IsReadOnly="True" Margin="0" TextChanged="imageNameTB\_TextChanged"/>

<Button x:Name="selectImageButton" Content="Select an Image" HorizontalAlignment="Right" Click="selectImageButton\_Click" Height="20" Width="88" Margin="0" BorderBrush="#FFABADB3"/>

<Button x:Name="resetSelectedImageButton" Content="X" HorizontalAlignment="Right" Margin="-18,0,0,0" Width="16" IsEnabled="False" Click="resetSelectedImageButton\_Click" BorderBrush="#FFABADB3"/>

</DockPanel>

<Grid >

<Grid Height="177" HorizontalAlignment="Left" Width="200" >

<Label x:Name="MessageText" Content="Input a password and confirm it" HorizontalAlignment="Center" Margin="20,10,20,147" VerticalAlignment="Center" Width="160" Height="20" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Padding="0" />

<PasswordBox x:Name="passwordBox" Width="116" Height="27" HorizontalAlignment="Center" VerticalAlignment="Center" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Margin="0,-80,0,0" PasswordChanged="passwordBox\_PasswordChanged" />

<PasswordBox x:Name="passwordBoxСonfirm" Height="27" Width="116" VerticalContentAlignment="Center" HorizontalContentAlignment="Center" VerticalAlignment="Center" HorizontalAlignment="Center" Margin="0" PasswordChanged="passwordBoxConfirm\_PasswordChanged" />

<Button x:Name="validatePasswordButton" Content="Validate the Password" HorizontalAlignment="Left" Margin="30,127,0,0" VerticalAlignment="Top" Width="140" Height="27" Click="validatePasswordButton\_Click" IsEnabled="False"/>

<Button x:Name="resetPasswordButton" Content="X" Margin="15,0,0,99" Height="17" RenderTransformOrigin="0.521,0.129" Width="17" HorizontalAlignment="Left" VerticalAlignment="Bottom" IsEnabled="False" Click="resetPasswordButton\_Click" Padding="0"/>

</Grid>

<Grid Height="177" HorizontalAlignment="Left" Width="200" Margin="200,0,0,0" >

<TextBox x:Name="finalArchiveNameTextBox" HorizontalAlignment="Center" Height="27" Margin="0,-80,0,0" TextWrapping="Wrap" Text="Input archive name" VerticalAlignment="Center" Width="120" VerticalContentAlignment="Center" HorizontalContentAlignment="Stretch" Foreground="#FFAAAAAA" LostFocus="finalArchiveNameTextBox\_LostFocus" GotFocus="finalArchiveNameTextBox\_GotFocus" TextChanged="finalArchiveNameTextBox\_TextChanged" />

</Grid>

</Grid>

<Grid Height="112" Width="397">

<ProgressBar x:Name="statusBar" HorizontalAlignment="Left" Minimum="0" Maximum="100" Value="0" Height="25" Margin="0,87,0,0" VerticalAlignment="Top" Width="397"/>

<TextBlock Text="{Binding ElementName=statusBar, Path=Value, StringFormat={}{0:0} %}" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0,0,0,-87" />

</Grid>

</StackPanel>

</Border>

<Border BorderBrush="#FF06A4F3" BorderThickness="1.5" Grid.Column="1" >

<StackPanel Grid.Column="1">

<ListBox x:Name="dataForEncryptionBox" Grid.Column="1" HorizontalAlignment="Left" VerticalAlignment="Top" Height="293" Width="390" Background="#FFBEE6F9" AllowDrop="True" Drop="getFileOnDrop" BorderThickness="0">

<Label x:Name="labelInsideDropArea" Content="Drop the Files Here" Height="40" Width="200" FontSize="20" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Margin="100,140,0,0"/>

</ListBox>

<Grid>

<Button x:Name="clearDataForEncryptionButton" Content="Clear" Width="194" Height="25" FontSize="14" HorizontalAlignment="Left" Click="clearDataForEncryptionButton\_Click" IsEnabled="False"/>

<Button x:Name="removeDataForEncryptionButton" Content="Remove" Width="194" Height="25" FontSize="14" HorizontalAlignment="Left" Click="removeDataForEncryptionButton\_Click" IsEnabled="False" Margin="194,0,0,0"/>

</Grid>

<Border BorderThickness="1" BorderBrush="#FF06A4F3"/>

<Grid>

<Button x:Name="encryptButton" Content="Let's Go" Margin="50,12,50,-48" FontSize="18" IsEnabled="False" Click="encryptButton\_Click" IsDefault="True"/>

</Grid>

</StackPanel>

</Border>

</Grid>

</Page>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using WinForms = System.Windows.Forms;

using System.Windows.Controls;

using System.Windows.Media;

using Diploma.crypto;

using Diploma.utils;

using System.Threading;

namespace Diploma

{

/// <summary>

/// Interaction logic for EncryptPage.xaml

/// </summary>

public partial class EncryptPage : Page

{

private string FolderPath;

private string ImagePath;

private string HashPassword;

private List<string> DataForEncryption;

private string ArchiveName;

private string ArchiveNamePlaceHolder = "Input archive name";

private const int OperationsCount = 5;

public EncryptPage()

{

InitializeComponent();

DataForEncryption = new List<string>();

FolderPath = null;

HashPassword = null;

ArchiveName = null;

}

private bool isPasswordValid(string password, string confirmedPassword)

{

MessageText.Visibility = Visibility.Visible;

if (!password.Equals(confirmedPassword))

{

passwordBoxСonfirm.Clear();

MessageText.Foreground = Brushes.Red;

MessageText.Content = "Passwords don't match!";

return false;

}

MessageText.Foreground = Brushes.Green;

MessageText.Content = "Password has been accepted ";

return true;

}

private void selectFolderButton\_Click(object sender, RoutedEventArgs e)

{

var folderDialog = new WinForms.FolderBrowserDialog();

var dialogResult = folderDialog.ShowDialog();

if (dialogResult == WinForms.DialogResult.OK)

{

folderNameTB.Text = folderDialog.SelectedPath;

}

updateFieldsStatus();

}

private void selectImageButton\_Click(object sender, RoutedEventArgs e)

{

var fileDialog = new WinForms.OpenFileDialog();

fileDialog.Filter = "Image files (\*.jpg, \*.jpeg, \*.jpe, \*.png) | \*.jpg; \*.jpeg; \*.jpe; \*.png";

var dialogResult = fileDialog.ShowDialog();

if (dialogResult == WinForms.DialogResult.OK)

{

imageNameTB.Text = fileDialog.FileName;

}

updateFieldsStatus();

}

private void validatePasswordButton\_Click(object sender, RoutedEventArgs e)

{

string password = passwordBox.Password;

string confirmedPassword = passwordBoxСonfirm.Password;

if (!isPasswordValid(password, confirmedPassword))

{

return;

}

Hash hash = new Hash();

this.HashPassword = hash.getHashSHA256(password);

validatePasswordButton.IsEnabled = false;

passwordBox.IsEnabled = false;

passwordBoxСonfirm.IsEnabled = false;

resetPasswordButton.IsEnabled = true;

updateFieldsStatus();

}

private void resetPasswordButton\_Click(object sender, RoutedEventArgs e)

{

MessageText.Foreground = Brushes.Black;

MessageText.Content = "Input a password and confirm";

validatePasswordButton.IsEnabled = true;

passwordBox.IsEnabled = true;

passwordBoxСonfirm.IsEnabled = true;

resetPasswordButton.IsEnabled = false;

passwordBox.Clear();

passwordBoxСonfirm.Clear();

this.HashPassword = null;

updateFieldsStatus();

}

private void getFileOnDrop(object sender, DragEventArgs e)

{

List<string> dropEventResult = new List<string>();

Utils utils = new Utils();

try

{

dropEventResult = new List<string>((IEnumerable<string>)e.Data.GetData(DataFormats.FileDrop));

List<string> allData = this.DataForEncryption.Union(dropEventResult).ToList();

this.DataForEncryption.Clear();

foreach (string data in allData)

{

if (!utils.IsDirectory(data))

{

this.DataForEncryption.Add(data);

}

}

dataForEncryptionBox.Items.Clear();

foreach (string data in this.DataForEncryption)

{

dataForEncryptionBox.Items.Add(data);

}

}

catch { // ignored }

if (this.DataForEncryption.Count != 0)

{

labelInsideDropArea.Visibility = Visibility.Hidden;

clearDataForEncryptionButton.IsEnabled = true;

removeDataForEncryptionButton.IsEnabled = true;

}

updateFieldsStatus();

}

private void folderNameTB\_TextChanged(object sender, TextChangedEventArgs e)

{

this.FolderPath = folderNameTB.Text;

resetSelectedFolderButton.IsEnabled = !String.IsNullOrEmpty(this.FolderPath);

}

private void imageNameTB\_TextChanged(object sender, TextChangedEventArgs e)

{

this.ImagePath = imageNameTB.Text;

resetSelectedImageButton.IsEnabled = !String.IsNullOrEmpty(this.ImagePath);

}

private void resetSelectedFolderButton\_Click(object sender, RoutedEventArgs e)

{

folderNameTB.Text = String.Empty;

updateFieldsStatus();

}

private void resetSelectedImageButton\_Click(object sender, RoutedEventArgs e)

{

imageNameTB.Text = String.Empty;

updateFieldsStatus();

}

private void passwordBox\_PasswordChanged(object sender, RoutedEventArgs e)

{

validatePasswordButton.IsEnabled = !String.IsNullOrEmpty(passwordBox.Password)

&& !String.IsNullOrEmpty(passwordBoxСonfirm.Password);

}

private void passwordBoxConfirm\_PasswordChanged(object sender, RoutedEventArgs e)

{

validatePasswordButton.IsEnabled = !String.IsNullOrEmpty(passwordBox.Password)

&& !String.IsNullOrEmpty(passwordBoxСonfirm.Password);

}

private void finalArchiveNameTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

if (finalArchiveNameTextBox.Text == this.ArchiveNamePlaceHolder)

{

finalArchiveNameTextBox.Text = String.Empty;

finalArchiveNameTextBox.Foreground = new SolidColorBrush((Color)ColorConverter.ConvertFromString("#FF000000"));

}

}

private void finalArchiveNameTextBox\_LostFocus(object sender, RoutedEventArgs e)

{

if (String.IsNullOrEmpty(finalArchiveNameTextBox.Text))

{

finalArchiveNameTextBox.Text = this.ArchiveNamePlaceHolder;

finalArchiveNameTextBox.Foreground = new SolidColorBrush((Color)ColorConverter.ConvertFromString("#FFAAAAAA"));

this.ArchiveName = null;

}

}

private void finalArchiveNameTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

if (finalArchiveNameTextBox.Text != this.ArchiveNamePlaceHolder)

{

this.ArchiveName =

!String.IsNullOrEmpty(finalArchiveNameTextBox.Text.Trim())

? finalArchiveNameTextBox.Text

: null;

updateFieldsStatus();

}

}

private void updateFieldsStatus()

{

encryptButton.IsEnabled = getCountOperationsDone() == OperationsCount;

statusBar.Value = getStatusBarValue();

clearDataForEncryptionButton.IsEnabled = this.DataForEncryption.Count != 0;

removeDataForEncryptionButton.IsEnabled = this.DataForEncryption.Count != 0;

}

private void encryptButton\_Click(object sender, RoutedEventArgs e)

{

Thread mainThread = new Thread(ArchiveCreation);

mainThread.Start();

Application.Current.MainWindow.Close();

}

private void ArchiveCreation()

{

Archive archive = new Archive();

archive.CreateArchive(this.DataForEncryption, this.FolderPath, this.ImagePath, this.ArchiveName, this.HashPassword);

Utils utils = new Utils();

utils.RemoveFilesByPath(this.DataForEncryption);

MessageBox.Show("Your data have been successfully protected!", "Message", MessageBoxButton.OK, MessageBoxImage.Information);

}

private double getStatusBarValue()

{

return 100.0 / OperationsCount \* getCountOperationsDone();

}

private int getCountOperationsDone()

{

int count = 0;

if (!String.IsNullOrEmpty(this.FolderPath)) count++;

if (!String.IsNullOrEmpty(this.HashPassword)) count++;

if (this.DataForEncryption.Count != 0) count++;

if (!String.IsNullOrEmpty(this.ArchiveName)) count++;

if (!String.IsNullOrEmpty(this.ImagePath)) count++;

return count;

}

private void clearDataForEncryptionButton\_Click(object sender, RoutedEventArgs e)

{

this.DataForEncryption.Clear();

dataForEncryptionBox.Items.Clear();

updateFieldsStatus();

}

private void removeDataForEncryptionButton\_Click(object sender, RoutedEventArgs e)

{

try

{

DataForEncryption.Remove(dataForEncryptionBox.SelectedItem.ToString());

}

catch { // ignored }

dataForEncryptionBox.Items.Remove(dataForEncryptionBox.SelectedItem);

updateFieldsStatus();

}

}

}

## ПРИЛОЖЕНИЕ В

Листинг «Decrypt Mode»

<Page x:Class="Diploma.DecryptPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Diploma"

mc:Ignorable="d"

d:DesignHeight="382" d:DesignWidth="800"

Title="DecryptPage">

<Grid Background="#FFBEE6F9">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"></ColumnDefinition>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="382"></RowDefinition>

</Grid.RowDefinitions>

<Border BorderBrush="#FF06A4F3" BorderThickness="1.5" >

<StackPanel MinWidth="100">

<DockPanel Margin="6,10,6,20">

<TextBox x:Name="folderNameTB" Height="20" Width="276" HorizontalAlignment="Left" HorizontalContentAlignment="Stretch" VerticalContentAlignment="Center" IsReadOnly="True" Margin="0" TextChanged="folderNameTB\_TextChanged"/>

<Button x:Name="selectFolderButton" Content="Select a Folder" HorizontalAlignment="Right" Click="selectFolderButton\_Click" Height="20" Width="85" Margin="0" BorderBrush="#FFABADB3"/>

<Button x:Name="resetSelectedFolderButton" Content="X" HorizontalAlignment="Right" Margin="-18,0,0,0" Width="16" IsEnabled="False" Click="resetSelectedFolderButton\_Click" BorderBrush="#FFABADB3"/>

</DockPanel>

<Grid >

<Grid Height="177" HorizontalAlignment="Left" Width="200" >

<Label x:Name="MessageText" Content="Input a password" HorizontalAlignment="Center" Margin="20,10,20,147" VerticalAlignment="Center" Width="160" Height="20" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Padding="0" />

<PasswordBox x:Name="passwordBox" Width="116" Height="27" HorizontalAlignment="Center" VerticalAlignment="Center" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Margin="0,-80,0,0" PasswordChanged="passwordBox\_PasswordChanged" />

</Grid>

<Grid Height="177" HorizontalAlignment="Left" Width="200" Margin="200,0,0,0" >

</Grid>

</Grid>

<Grid Height="152" Width="397">

<ProgressBar x:Name="statusBar" HorizontalAlignment="Left" Minimum="0" Maximum="100" Value="0" Height="25" Margin="0,127,0,0" VerticalAlignment="Top" Width="397"/>

<TextBlock Text="{Binding ElementName=statusBar, Path=Value, StringFormat={}{0:0} %}" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0,0,0,-127" />

</Grid>

</StackPanel>

</Border>

<Border BorderBrush="#FF06A4F3" BorderThickness="1.5" Grid.Column="1" >

<StackPanel Grid.Column="1">

<ListBox x:Name="dataForDecryptionBox" Grid.Column="1" HorizontalAlignment="Left" VerticalAlignment="Top" Height="293" Width="390" Background="#FFBEE6F9" AllowDrop="True" Drop="getFileOnDrop" BorderThickness="0">

<Label x:Name="labelInsideDropArea" Content="Drop the Archive Here" Height="40" Width="220" FontSize="20" HorizontalContentAlignment="Center" VerticalContentAlignment="Center" Margin="90,140,0,0"/>

</ListBox>

<Button x:Name="clearDataForDecryptionButton" Content="Clear" Width="388" Height="25" FontSize="14" HorizontalAlignment="Left" Click="clearDataForDecryptionButton\_Click" IsEnabled="False"/>

<Border BorderThickness="1" BorderBrush="#FF06A4F3"/>

<Grid>

<Button x:Name="decryptButton" Content="Let's Go" Margin="50,12,50,-48" FontSize="18" IsEnabled="False" Click="decryptButton\_Click" IsDefault="True"/>

</Grid>

</StackPanel>

</Border>

</Grid>

</Page>

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using Diploma.utils;

using WinForms = System.Windows.Forms;

using Diploma.crypto;

using System.Threading;

namespace Diploma

{

/// <summary>

/// Interaction logic for DecryptPage.xaml

/// </summary>

public partial class DecryptPage : Page

{

private string FolderPath;

private string HashPassword;

private string ArchiveFile;

private const int OperationsCount = 3;

private short AttemptsCount = 0;

private short MaxAttemptsCount = 3;

private int ImageLength;

public DecryptPage()

{

InitializeComponent();

ArchiveFile = String.Empty;

FolderPath = null;

HashPassword = null;

}

private void decryptButton\_Click(object sender, RoutedEventArgs e)

{

Hash hash = new Hash();

Archive archive = new Archive();

this.HashPassword = hash.getHashSHA256(passwordBox.Password);

this.ImageLength = archive.GetImageLength(this.ArchiveFile);

if (!archive.IsPasswordCorrect(this.ArchiveFile, this.HashPassword, this.ImageLength))

{

this.AttemptsCount++;

if (this.AttemptsCount != this.MaxAttemptsCount)

{

MessageBox.Show($@"Password is incorrect. {this.MaxAttemptsCount - this.AttemptsCount} attempt(s) left!", "Message",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

else

{

MessageBox.Show("Password is incorrect. The app will be closed!", "Message", MessageBoxButton.OK, MessageBoxImage.Stop);

Application.Current.MainWindow.Close();

}

return;

}

Thread mainThread = new Thread(ArchiveExtraction);

mainThread.Start();

Application.Current.MainWindow.Close();

}

private void ArchiveExtraction()

{

Archive archive = new Archive();

archive.ExtractArchive(this.ArchiveFile, this.FolderPath, this.HashPassword, this.ImageLength);

Utils utils = new Utils();

utils.RemoveFileByPath(this.ArchiveFile);

MessageBox.Show("Data protection was successfully removed!", "Message", MessageBoxButton.OK, MessageBoxImage.Information);

}

private void selectFolderButton\_Click(object sender, RoutedEventArgs e)

{

var folderDialog = new WinForms.FolderBrowserDialog();

var dialogResult = folderDialog.ShowDialog();

if (dialogResult == WinForms.DialogResult.OK)

{

folderNameTB.Text = folderDialog.SelectedPath;

}

updateFieldsStatus();

}

private void resetSelectedFolderButton\_Click(object sender, RoutedEventArgs e)

{

folderNameTB.Text = String.Empty;

updateFieldsStatus();

}

private void folderNameTB\_TextChanged(object sender, TextChangedEventArgs e)

{

this.FolderPath = folderNameTB.Text;

resetSelectedFolderButton.IsEnabled = !String.IsNullOrEmpty(this.FolderPath);

}

private void getFileOnDrop(object sender, DragEventArgs e)

{

List<string> dropEventResult = new List<string>();

Utils utils = new Utils();

try

{

dropEventResult = new List<string>((IEnumerable<string>)e.Data.GetData(DataFormats.FileDrop));

this.ArchiveFile = utils.IsDirectory(dropEventResult[0]) ? this.ArchiveFile : dropEventResult[0];

dataForDecryptionBox.Items.Clear();

dataForDecryptionBox.Items.Add(this.ArchiveFile);

}

catch

{

// ignored

}

if (!String.IsNullOrEmpty(this.ArchiveFile))

{

labelInsideDropArea.Visibility = Visibility.Hidden;

clearDataForDecryptionButton.IsEnabled = true;

if (String.IsNullOrEmpty(folderNameTB.Text))

{

folderNameTB.Text = utils.GetFolderPathFromFullPath(this.ArchiveFile);

}

}

updateFieldsStatus();

}

private void passwordBox\_PasswordChanged(object sender, RoutedEventArgs e)

{

updateFieldsStatus();

}

private void updateFieldsStatus()

{

decryptButton.IsEnabled = getCountOperationsDone() == OperationsCount;

statusBar.Value = getStatusBarValue();

clearDataForDecryptionButton.IsEnabled = !String.IsNullOrEmpty(this.ArchiveFile);

}

private int getCountOperationsDone()

{

int count = 0;

if (!String.IsNullOrEmpty(this.FolderPath)) count++;

if (!String.IsNullOrEmpty(passwordBox.Password)) count++;

if (!String.IsNullOrEmpty(this.ArchiveFile)) count++;

return count;

}

private double getStatusBarValue()

{

return 100.0 / OperationsCount \* getCountOperationsDone();

}

private void clearDataForDecryptionButton\_Click(object sender, RoutedEventArgs e)

{

this.ArchiveFile = null;

dataForDecryptionBox.Items.Clear();

updateFieldsStatus();

}

}

}

## ПРИЛОЖЕНИЕ Г

Листинг класса «Cryptography»

using System;

using System.Text;

using System.Security.Cryptography;

using Diploma.utils;

using System.IO;

using System.Diagnostics;

using System.Windows;

namespace Diploma.crypto

{

public class Cryptography

{

Constants constants = new Constants();

private int SaltArrayLength;

private int KeySize = 256;

private int BlockSize = 128;

private int RfcIterationsCount = 52768;

private byte[] Salt;

public Cryptography ()

{

this.SaltArrayLength = constants.SaltByteArrayLength;

this.Salt = GenerateRandomSalt();

}

public Cryptography(byte[] salt)

{

this.SaltArrayLength = constants.SaltByteArrayLength;

this.Salt = salt;

}

public byte[] GetSalt ()

{

return this.Salt;

}

public byte[] AesEncryption(byte[] originalBytes, string hashPassword)

{

byte[] passwordBytes = Encoding.UTF8.GetBytes(hashPassword);

byte[] encryptedBytes = null;

hashPassword = null;

GC.Collect();

using (Aes AES = new AesManaged())

{

AES.KeySize = this.KeySize;

AES.BlockSize = this.BlockSize;

using (var key = GenerateKey(passwordBytes, this.Salt))

{

AES.Key = key.GetBytes(AES.KeySize / 8);

AES.IV = key.GetBytes(AES.BlockSize / 8);

AES.Padding = PaddingMode.PKCS7;

AES.Mode = CipherMode.CBC;

using (var msEncrypt = new MemoryStream())

{

using (var csEncrypt = new CryptoStream(msEncrypt, AES.CreateEncryptor(), CryptoStreamMode.Write))

{

try

{

csEncrypt.Write(originalBytes, 0, originalBytes.Length);

csEncrypt.Close();

}

catch (Exception ex)

{

Debug.WriteLine("Encryption Error: " + ex.Message);

MessageBox.Show("Encryption Error: " + ex.Message);

}

encryptedBytes = msEncrypt.ToArray();

}

}

}

}

return encryptedBytes;

}

public byte[] AesDecryption(byte[] encryptedBytes, string hashPassword)

{

byte[] passwordBytes = Encoding.UTF8.GetBytes(hashPassword);

byte[] decryptedBytes = null;

GC.Collect();

using (Aes AES = new AesManaged())

{

AES.KeySize = this.KeySize;

AES.BlockSize = this.BlockSize;

using (var key = GenerateKey(passwordBytes, this.Salt))

{

AES.Key = key.GetBytes(AES.KeySize / 8);

AES.IV = key.GetBytes(AES.BlockSize / 8);

AES.Padding = PaddingMode.PKCS7;

AES.Mode = CipherMode.CBC;

using (var msEncrypt = new MemoryStream())

{

using (var csEncrypt = new CryptoStream(msEncrypt, AES.CreateDecryptor(), CryptoStreamMode.Write))

{

try

{

csEncrypt.Write(encryptedBytes, 0, encryptedBytes.Length);

csEncrypt.Close();

}

catch (Exception ex)

{

Debug.WriteLine("Decryption Error: " + ex.Message);

MessageBox.Show("Decryption Error: " + ex.Message);

}

decryptedBytes = msEncrypt.ToArray();

}

}

}

}

return decryptedBytes;

}

private Rfc2898DeriveBytes GenerateKey(byte[] password, byte[] salt)

{

return new Rfc2898DeriveBytes(password, salt, this.RfcIterationsCount);

}

private byte[] GenerateRandomSalt()

{

byte[] result = new byte[this.SaltArrayLength];

using (RNGCryptoServiceProvider provider = new RNGCryptoServiceProvider())

{

provider.GetBytes(result);

}

return result;

}

}

}

## ПРИЛОЖЕНИЕ Д

Листинг класса «Hash»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Cryptography;

namespace Diploma.crypto

{

public class Hash

{

public string getHashSHA256(string input)

{

using (SHA256Managed sha256 = new SHA256Managed())

{

var hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(input));

string hashString = string.Empty;

foreach (byte x in hash)

{

hashString += String.Format("{0:x2}", x);

}

return hashString;

}

}

public string getHashSHA1(string input)

{

using (SHA1Managed sha1 = new SHA1Managed())

{

var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));

string hashString = string.Empty;

foreach (byte x in hash)

{

hashString += String.Format("{0:x2}", x);

}

return hashString;

}

}

}

}

## ПРИЛОЖЕНИЕ Е

Листинг класса «Archive»

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Diploma.crypto;

namespace Diploma.utils

{

public class Archive

{

Constants constants = new Constants();

private int BlockSize;

private byte[] FileEndKeyBytes;

private byte[] FileNotEndKeyBytes;

public Archive()

{

this.FileEndKeyBytes = BitConverter.GetBytes(constants.FileEndKey);

this.FileNotEndKeyBytes = BitConverter.GetBytes(constants.FileNotEndKey);

this.BlockSize = (int)Math.Pow(2, 24);

}

public void CreateArchive (List<string> files, string folderPath, string imagePath, string archiveName, string hashPassword)

{

Hash hash = new Hash();

Cryptography cryptography = new Cryptography();

Utils utils = new Utils();

LZMA lzma = new LZMA();

byte[] salt = cryptography.GetSalt();

byte[] imageBytes = File.ReadAllBytes(imagePath);

byte[] imageLengthBytes = BitConverter.GetBytes(imageBytes.Length);

string doubleHashPassword = hash.getHashSHA256(hashPassword);

byte[] passwordBytes = Encoding.UTF8.GetBytes(doubleHashPassword);

string imageExtension = utils.GetFileExtensionFromPath(imagePath);

string finalPath = $@"{folderPath}\{archiveName}{imageExtension}";

using (var fileStream = new FileStream(finalPath, FileMode.Create))

{

fileStream.Write(imageBytes, 0, imageBytes.Length);

fileStream.Write(passwordBytes, 0, passwordBytes.Length);

fileStream.Write(salt, 0, salt.Length);

passwordBytes = null;

foreach (string file in files)

{

string fileName = utils.GetFileNameFromPath(file);

bool shouldCompress = utils.ShouldCompressAndDecompress(fileName, constants.FileExtensionsToExcludeFromCompressProcess);

byte [] fileNameBlock = cryptography.AesEncryption(Encoding.UTF8.GetBytes(fileName), hashPassword);

byte [] fileNameLengthBlock = BitConverter.GetBytes(fileNameBlock.Length);

fileStream.Write(fileNameLengthBlock, 0, fileNameLengthBlock.Length);

fileStream.Write(fileNameBlock, 0, fileNameBlock.Length);

using (var internalFileStream = new FileStream(file, FileMode.Open, FileAccess.Read))

{

long sumCountOfReadBytes = 0;

long fileLength = new FileInfo(file).Length;

byte[] block = BlockSize < fileLength ? new byte[BlockSize] : new byte[fileLength];

while ((internalFileStream.Read(block, 0, block.Length)) > 0)

{

fileStream.Write(this.FileNotEndKeyBytes, 0, this.FileNotEndKeyBytes.Length);

sumCountOfReadBytes += block.Length;

byte[] compressedBlock = shouldCompress

? lzma.Compress(block)

: block;

byte[] encryptedBlock = cryptography.AesEncryption(compressedBlock, hashPassword);

byte[] encryptedBlockLength = BitConverter.GetBytes(encryptedBlock.Length);

fileStream.Write(encryptedBlockLength, 0, encryptedBlockLength.Length);

fileStream.Write(encryptedBlock, 0, encryptedBlock.Length);

block = BlockSize < (fileLength - sumCountOfReadBytes) ? new byte[BlockSize] : new byte[(fileLength - sumCountOfReadBytes)];

}

fileStream.Write(this.FileEndKeyBytes, 0, this.FileEndKeyBytes.Length);

}

}

fileStream.Write(imageLengthBytes, 0, imageLengthBytes.Length);

hashPassword = null;

}

}

public void ExtractArchive(string archive, string folderPath, string hashPassword, int imageLength)

{

Hash hash = new Hash();

Utils utils = new Utils();

LZMA lzma = new LZMA();

string doubleHashPassword = hash.getHashSHA256(hashPassword);

byte[] passwordBytes = Encoding.UTF8.GetBytes(doubleHashPassword);

long archiveLength = new FileInfo(archive).Length;

int sumCountOfReadData = 0;

int iterationCountOfReadData = 0;

using (var fileStream = new FileStream(archive, FileMode.Open))

{

fileStream.Seek(imageLength + passwordBytes.Length, SeekOrigin.Begin);

byte[] salt = new byte[constants.SaltByteArrayLength];

sumCountOfReadData += fileStream.Read(salt, 0, salt.Length);

Cryptography cryptography = new Cryptography(salt);

salt = null;

byte[] block = new byte[constants.FileNameLengthByteArrayLength]; // fileNameLengthBlock = 4

while ((iterationCountOfReadData = fileStream.Read(block, 0, block.Length)) > 0)

{

sumCountOfReadData += iterationCountOfReadData;

if (archiveLength - imageLength - passwordBytes.Length - sumCountOfReadData == 0)

{

return;

}

var fileNameLength = BitConverter.ToInt32(block, 0);

block = new byte[fileNameLength]; // fileNameBlock

sumCountOfReadData += fileStream.Read(block, 0, block.Length);

string fileName = Encoding.UTF8.GetString(cryptography.AesDecryption(block, hashPassword));

bool shouldDecompress = utils.ShouldCompressAndDecompress(fileName, constants.FileExtensionsToExcludeFromCompressProcess);

string finalPath = $@"{folderPath}\{fileName}";

using (var internalFileStream = new FileStream(finalPath, FileMode.Create))

{

block = new byte[constants.FileEndKeyByteArrayLength]; // file?EndKeyBlock = 2

sumCountOfReadData += fileStream.Read(block, 0, block.Length);

while (BitConverter.ToInt16(block, 0) != constants.FileEndKey)

{

block = new byte[constants.FileLengthByteArrayLength]; // fileLengthBlock = 4

sumCountOfReadData += fileStream.Read(block, 0, block.Length);

var fileBlockLength = BitConverter.ToInt32(block, 0);

byte[] encryptedBytes = new byte[fileBlockLength]; // fileBlock

sumCountOfReadData += fileStream.Read(encryptedBytes, 0, encryptedBytes.Length);

byte[] decryptedBytes = cryptography.AesDecryption(encryptedBytes, hashPassword);

byte[] decompressedBytes = shouldDecompress

? lzma.Decompress(decryptedBytes)

: decryptedBytes;

internalFileStream.Write(decompressedBytes, 0, decompressedBytes.Length);

block = new byte[constants.FileEndKeyByteArrayLength]; // file?EndKeyBlock = 2

sumCountOfReadData += fileStream.Read(block, 0, block.Length);

}

}

block = new byte[constants.FileNameLengthByteArrayLength]; // fileNameLengthBlock = 4

}

}

}

public int GetImageLength(string archive)

{

using (var fileStream = new FileStream(archive, FileMode.Open))

{

fileStream.Seek(-constants.ImageByteArrayLength, SeekOrigin.End);

byte[] imageLengthBytes = new byte[constants.ImageByteArrayLength];

fileStream.Read(imageLengthBytes, 0, imageLengthBytes.Length);

int imageLength = BitConverter.ToInt32(imageLengthBytes, 0);

return imageLength;

}

}

public bool IsPasswordCorrect(string archive, string hashPassword, int imageLength)

{

if (imageLength < 0)

{

return false;

}

Hash hash = new Hash();

string doubleHashPassword = hash.getHashSHA256(hashPassword);

byte[] passwordBytes = Encoding.UTF8.GetBytes(doubleHashPassword);

byte[] extractedPasswordBytes = null;

using (var fileStream = new FileStream(archive, FileMode.Open))

{

fileStream.Seek(imageLength, SeekOrigin.Begin);

extractedPasswordBytes = new byte[passwordBytes.Length];

fileStream.Read(extractedPasswordBytes, 0, extractedPasswordBytes.Length);

}

string extractedDoubleHashPassword = Encoding.UTF8.GetString(extractedPasswordBytes);

return doubleHashPassword.Equals(extractedDoubleHashPassword);

}

}

}

## ПРИЛОЖЕНИЕ Ж

Листинг класса «Utils»

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Diploma.utils

{

class Utils

{

public string GetFileNameFromPath(string path)

{

string[] splitted = path.Split('\\');

return splitted[splitted.Length - 1];

}

public string GetFolderPathFromFullPath(string fullPath)

{

string[] splitted = fullPath.Split('\\');

return String.Join("\\", splitted.SkipLast(1));

}

public string GetFileExtensionFromPath(string path)

{

return Path.GetExtension(path);

}

public bool IsDirectory (string path)

{

FileAttributes attr = File.GetAttributes(path);

return attr.HasFlag(FileAttributes.Directory);

}

public void RemoveFilesByPath(List<string> pathList)

{

foreach (var data in pathList)

{

RemoveFileByPath(data);

}

}

public void RemoveFileByPath(string filePath)

{

if (!IsDirectory(filePath))

{

File.Delete(filePath);

}

}

public bool ShouldCompressAndDecompress(string currentFile, List<string> fileExtensionsToExclude)

{

string fileExtension = GetFileExtensionFromPath(currentFile);

return !fileExtensionsToExclude.Contains(fileExtension.ToLower());

}

}

}