МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-40 05 01 Информационные системы и технологии
Направление специальности 1-40 01 02 03 Информационные системы и технологии (издательско-полиграфический комплекс)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА:

по дисциплине «Защита информации и надежность информационных систем»
Тема: «Защита .net приложений на основе алгоритма шифрования AES»

Исполнитель
Студент 3 курса группы 3 Иванова А. А.
(Ф.И.О.)

Руководитель работы ассистент Копыток Д.В
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой
Председатель Копыток Д.В
(подпись)

Минск 2022

Содержание

Введение

В наше время защита приложений является одной из самых актуальных тем на конференциях и форумах. Как защитить код своего .net приложения? Самый распространённый вариант ответа на данных вопрос – обфускация. С одной стороны — прост в использовании, а с другой — недостаточно надёжно прячет исходные файлы, которые являются важной составляющей любого приложения. В ходе работы, я разработала свой вариант защиты приложений, хорошо подходящий для утилит, использование которых предполагается самим автором (либо доверенным лицам), код которых показывать нежелательно.

Защита будет строиться на шифровании сборок симметричным ключом и динамическом их дешифровании в процессе работы приложения с помощью алгоритма шифрования AES, о котором будет рассказано ниже. Ключ шифрования будет определяться пользователем на этапе развёртывания и вводиться как пароль при запуске.

AES (Advanced Encryption Standard) – алгоритм шифрования, действующий в качестве государственного стандарта в США с 2001 г. Алгоритм шифрования AES, был разработан в 1998г. Винсентом Рэйменом и Йоаном Дайменом, которые, в частности, разработали алгоритмы SHARK [5].

В основу стандарта положен шифр Rijndael. Шифр Rijndael/AES (т. е. рекомендуемый стандартом) характеризуется размером блока 128 битов, длиной ключа 128, 192 или 256 битов и количеством раундов 10, 12 или 14 в зависимости от длины ключа. Основу Rijndael составляют так называемые линейно-подстановочные преобразования. В алгоритме широко используются табличные вычисления, причем все необходимые таблицы задаются константно, т. е. не зависят ни от ключа, ни от данных.

Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES. Национальный институт стандартов и технологий США (англ. National Institute of Standards and Technology, NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур.

Целью работы является создание пользовательской утилиты, с помощью которой пользователь сможет защищать свои приложения, написанные на языке программирования c#.

Результатом данной курсовой работы является проект, содержащий в себе проект-утилиту для шифрования приложения и проект-приложение, которое будет шифроваться в качестве примера.

В процессе выполнения работы были поставлены следующие задачи:

1. Провести аналитический обзор литературы, а также поиск и сравнение аналогов.
2. Спроектировать и разработать программное средство.
3. Описать разработанное программное средство.
4. Провести тестирование разработанного программного средства.
5. Написать руководство пользователя разработанного программного средства.

1. Постановка задачи

Требуется разработать утилиту, которая будет обеспечивать защиту любого приложения, разработанного на языке программирования c#, с помощью шифрования сборок построенном на алгоритме AES.

Требование к средствам разработки:

1) Язык программирования C#.

2) Средство разработки MS Visual Studio.

Проект-утилита должна:

– шифровать сборки проекта при помощи алгоритма AES;

– динамически дешифровывать эти сборки;

– динамическая декомпиляция и компиляция сборок проекта;

– вытягивание ключа любой длины из любого текстового файла и преобразование этого ключа

Проект-приложение должно:

– отображать содержание текстового файла для шифрования;

– отображать результаты шифрования;

– отображать результаты дешифрования;

– принимать ключи для шифрования и расшифрования любой длины;

– Программа также должна также иметь удобный графический интерфейс, и быть максимально простой в использовании и запускаться без установки дополнительных программных средств.

2. Аналитический обзор литературы

Advanced Encryption Standard (ранее известный как Rijndael) - один из способов шифрования информации. Он настолько безопасен, что даже грубая сила не сможет его сломать. Этот передовой стандарт шифрования используется Агентством национальной безопасности (АНБ) во многих отраслях, включая онлайн-банкинг.

AES – это современный стандарт шифрования данных. Он не имеет себе равных по уровню безопасности и защиты, который он предлагает. AES – это блочный шифр, с шифрованием симметричным ключом. AES – это симметричный тип шифрования рисунок 2.1.

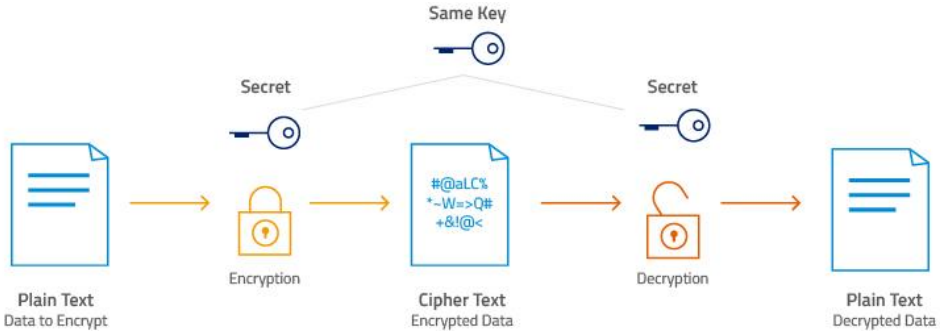

Рисунок 2.1 – Представление симметричного типа шифрования

«Симметричный» означает, что он использует один и тот же ключ для шифрования и дешифрования информация. Кроме того, и то и другое что собой представляет отправитель и получатель данных нужна их копия для расшифровки шифра.

С другой стороны, асимметричный ключевые системы используют разные ключи для каждого из двух процессов: шифрование и дешифрование. Преимущество симметричных систем как AES заключается в том, что они намного быстрее, чем асимметричные единицы. Это связано с тем, что алгоритмы с симметричным ключом требуют меньшей вычислительной мощности. Вот почему асимметричные ключи лучше всего использовать для передачи внешних файлов. Симметричные ключи лучше подходят для внутреннего шифрования. Именно поэтому в данной работе выбран шифр с использованием симметричных ключей.

AES – это еще и то, что мир технологий называет «Блочный шифр». Он называется «блочным», потому что этот тип шифра разделяет информацию, которая должна быть зашифрована (известный как открытый текст) на разделы, называемые блоками.

Чтобы быть более конкретным, AES использует Размер блока 128 бит. Это означает, что данные разделены на массив четыре на четыре содержащий 16 байт. Каждый байт содержит восемь бит. Следовательно, умножение 16 байтов на 8 бит дает всего 128 бит в каждом блоке.

Независимо от этого разделения, размер зашифрованных данных остается прежним. Другими словами, 128 бит открытого текста дают 128 бит зашифрованного текста.

Процесс шифрования AES относительно прост для понимания. Это позволяет простая реализация, а также на самом деле быстрое шифрование и дешифрование**.** Кроме того, AES требует меньше памяти чем другие типы шифрования (например, DES). Наконец, когда вам потребуется дополнительный уровень безопасности, вы можете легко комбинировать AES с различными протоколами безопасности, например, WPA2 или даже другие типы шифрования, такие как SSL.

Старый 56-битный ключ DES можно было взломать менее чем за сутки. Но для AES это займет миллиарды лет чтобы сломаться, используя вычислительные технологии, которые у нас есть сегодня. Исследователи, изучавшие AES, нашли несколько потенциальных способов попасть внутрь [5].

Угроза №1: атаки по ключевым словам. В 2009 году они обнаружили возможную атаку с использованием связанных ключей. Вместо грубой силы эти атаки будут настроить таргетинг на сам ключ шифрования. Этот тип криптоанализа будет пытаться взломать шифр, наблюдая, как он работает с использованием разных ключей. К счастью, атака на связанный ключ только угроза к системам AES. Единственный способ, которым это может работать, — это если хакер знает (или подозревает) связь между двумя наборами ключей. Будьте уверены, криптографы быстро улучшили сложность расписания ключей AES после этих атак, чтобы предотвратить их.

Угроза №2: известная отличительная атака. В отличие от грубой силы, эта атака использовала известный ключ расшифровать структуру шифрования. Однако этот взлом был нацелен только на восьмиэтапную версию AES 128, не стандартная 10-раундовая версия. Однако, это не серьезная угроза.

Угроза № 3: атаки по стороннему каналу. Это основной риск, с которым сталкивается AES. Это работает, пытаясь подобрать любую информацию система протекает. Хакеры могут слушать звуки, электромагнитные сигналы, информация о времени или потребляемая мощность чтобы попытаться выяснить, как работают алгоритмы безопасности. Лучший способ предотвратить атаки по побочным каналам – это устранение утечек информации или маскирование утечек данных (генерируя дополнительные электромагнитные сигналы или звуки).

Угроза №4: Раскрытие ключа. Это достаточно легко доказать, выполнив следующие действия:

– Надежные пароли

– Многофакторная аутентификация

– Межсетевые экраны

– Антивирусная программа

Для AES длина input (блока входных данных) и State (состояния) постоянна и равна 128 бит, а длина шифроключа K составляет 128, 192, или 256 бит. При этом исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин input, State и Cipher Key

в 32-битных словах используется нотация Nb = 4 для input и State, Nk = 4, 6, 8 для Cipher Key соответственно для разных длин ключей.

Вначале зашифровывания input копируется в массив State по правилу $state[r, c] = input[r + 4c]$, для $0 \leq r \leq 4$ и $0 \leq c \leq Nb$. После этого к State применяется процедура AddRoundKey (), и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу $output[r + 4c] = state[r, c]$, для $0 \leq r \leq 4$ и $0 \leq c \leq Nb$.

Отдельные трансформации SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() — обрабатывают State. Массив w[] — содержит key schedule.

Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов, используя таблицу замен (S-box) (Рисунок 2.2).

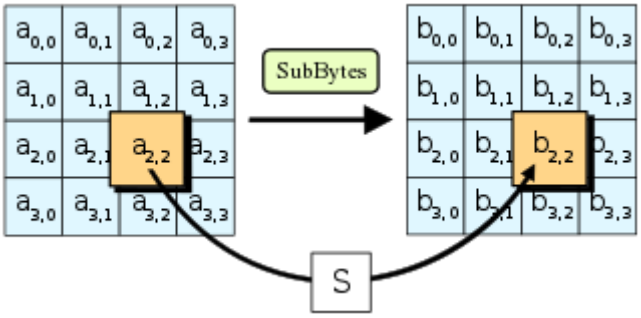

Рисунок 2.2 – Схема работы процедуры SubBytes()

Такая операция обеспечивает нелинейность алгоритма шифрования. Построение S-box состоит из двух шагов. Во-первых, производится взятие обратного числа в поле Галуа GF(2^8) . Для всех операций в этом поле используется неприводимый полином $z^8+z^4+z^3+z+1$. Во-вторых, к каждому байту b, из которых состоит S-box, применяется следующая операция по формуле 2.1.

$$b_i' = b_i \oplus b_{(i+4) mod 8} \oplus b_{(i+5) mod 8} \oplus b_{(i+6) mod 8} \oplus b_{(i+7) mod 8} \oplus c_i \qquad (2.1)$$

где $0 \leq i < 8$, и где есть i-ый бит b, а c_i — i-ый бит константы $c=63_{16}=99_{10}=01100011_2$. Таким образом обеспечивается защита от атак, основанных на простых алгебраических свойствах [5]

ShiftRows работает со строками State. При этой трансформации строки состояния циклически сдвигаются на r байт по горизонтали в зависимости от номера строки. Для нулевой строки r = 0, для первой строки r = 1 Б и т. Д (рисунок 2.2).

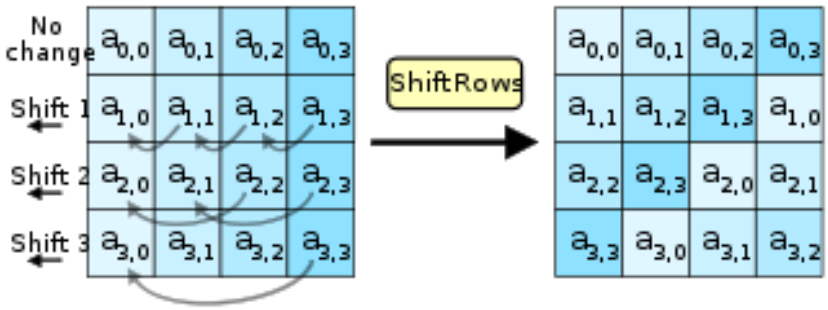

Рисунок 2.2 – Наглядное представление работы алгоритма ShiftRows

Таким образом, каждая колонка выходного состояния после применения процедуры ShiftRows состоит из байтов из каждой колонки начального состояния. Для алгоритма Rijndael паттерн смещения строк для 128- и 192-битных строк одинаков. Однако для блока размером 256 бит отличается от предыдущих тем, что 2-е, 3-и и 4-е строки смещаются на 1, 3 и 4 байта соответственно. Это замечание не относится к AES, так как он использует алгоритм Rijndael только с 128-битными блоками, независимо от размера ключа.

В процедуре MixColums четыре байта каждой колонки State смешиваются, используя для этого обратимую линейную трансформацию. MixColums обрабатывает состояния по колонкам, трактуя каждую из них как полином третьей степени (рисунок 2.3).

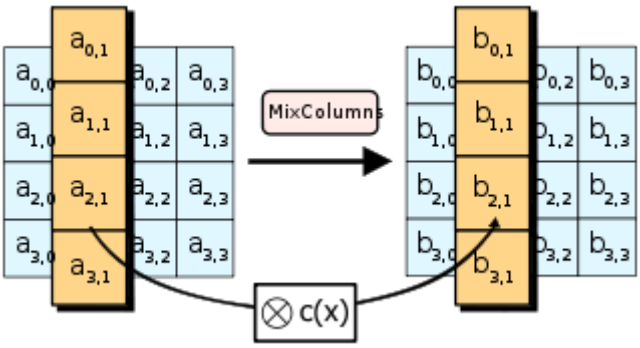

Рисунок 2.3 – Наглядное представление работы алгоритма MixColums

Над этими полиномами производится умножение в $GF(2^8)$ по модулю x^4+1 на фиксированный многочлен по формуле 2.2 [5]

$$c(x) = 3^x + x^2 + x + 2 \qquad (2.2)$$

В процедуре AddRoundKey RoundKey каждого раунда объединяется со State. Для каждого раунда RoundKey получается из ChiperKey с помощью процедуры KeyExposition; каждый RoundKey такого же размера, что и State. Процедура производит побитовый XOR каждого байта State с каждым байтом RoundKey (рисунок 2.4).

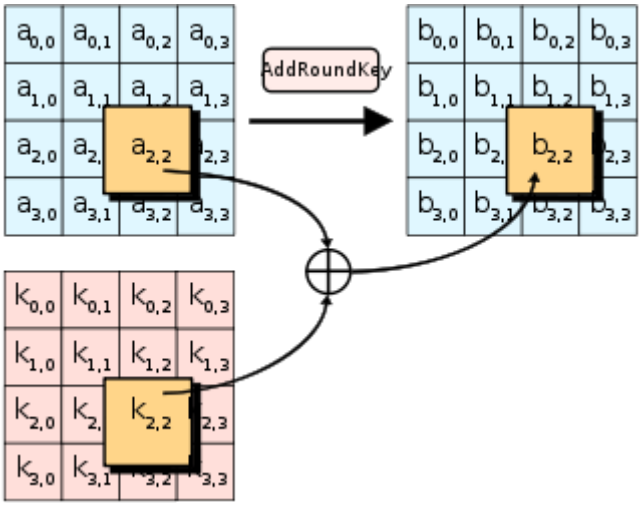

Рисунок 2.4 – Наглядное представление работы процедуры AddRoundKey[5]

Алгоритм обработки ключа состоит из двух процедур:

– Алгоритм генерации раундовых ключей (алгоритм расширения ключа)

– Алгоритм выбора раундового ключа (ключа итерации)

Далее приведем аналоги программ для шифрования, в которых присутствует алгоритм шифрования AES, они не являются утилитами для защиты приложений, однако с помощью этих приложений можно вручную шифровать файлы сборок.

VeraCrypt – один из самых популярных инструментов безопасности, предоставляющий шифрование корпоративного уровня для важных данных. Интерфейс предоставлен на рисунке 2.5.

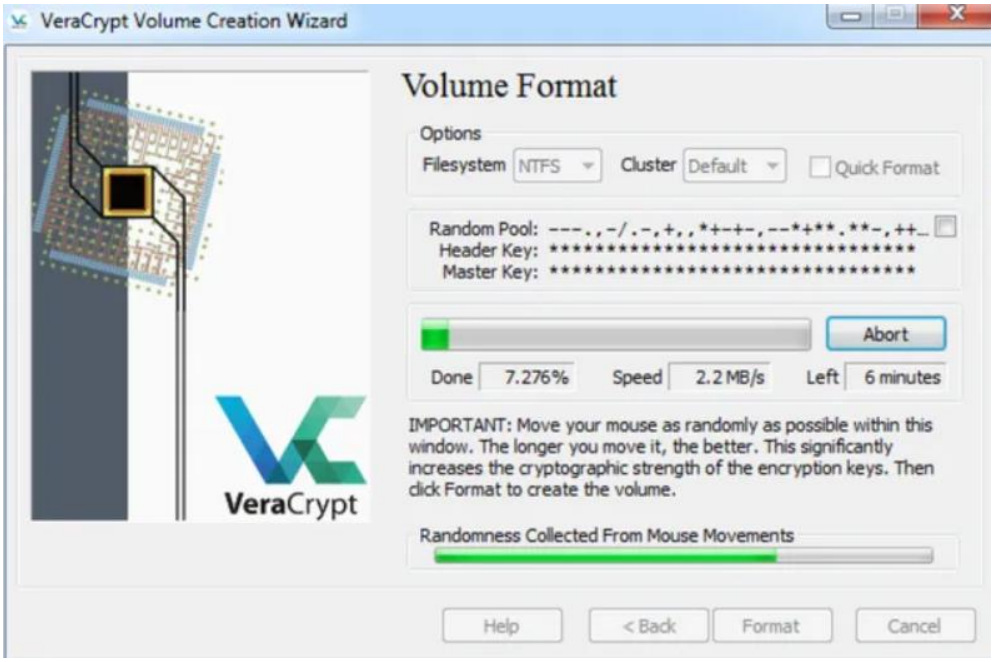

Рисунок 2.5 – Интерфейс программы VeraCrypt

Система довольно проста в использовании, и всё, что она действительно делает, это добавляет зашифрованные пароли к данным и разделам. Всё, что нужно

сделать, это дать инструменту несколько деталей о данных, таких как размер тома, местоположение и алгоритм хеширования.

Отличительной чертой VeraCrypt является то, что она невосприимчива к атаке методом «грубой силы», поэтому пользователю никогда не придется беспокоиться о том, что хакеры смогут расшифровать его пароли и другие конфиденциальные данные. Базовая версия программного обеспечения также полностью бесплатна.

Плюсы VeraCrypt:

- базовая версия полностью бесплатна;
- обеспечивает эффективное шифрование.

Минусы VeraCrypt:

- выборочный подход;
- первоначальная загрузка немного сбивает с толку.

AxCrypt – шифрование для небольших групп и отдельных лиц. Интерфейс предоставлен на рисунке 2.6.

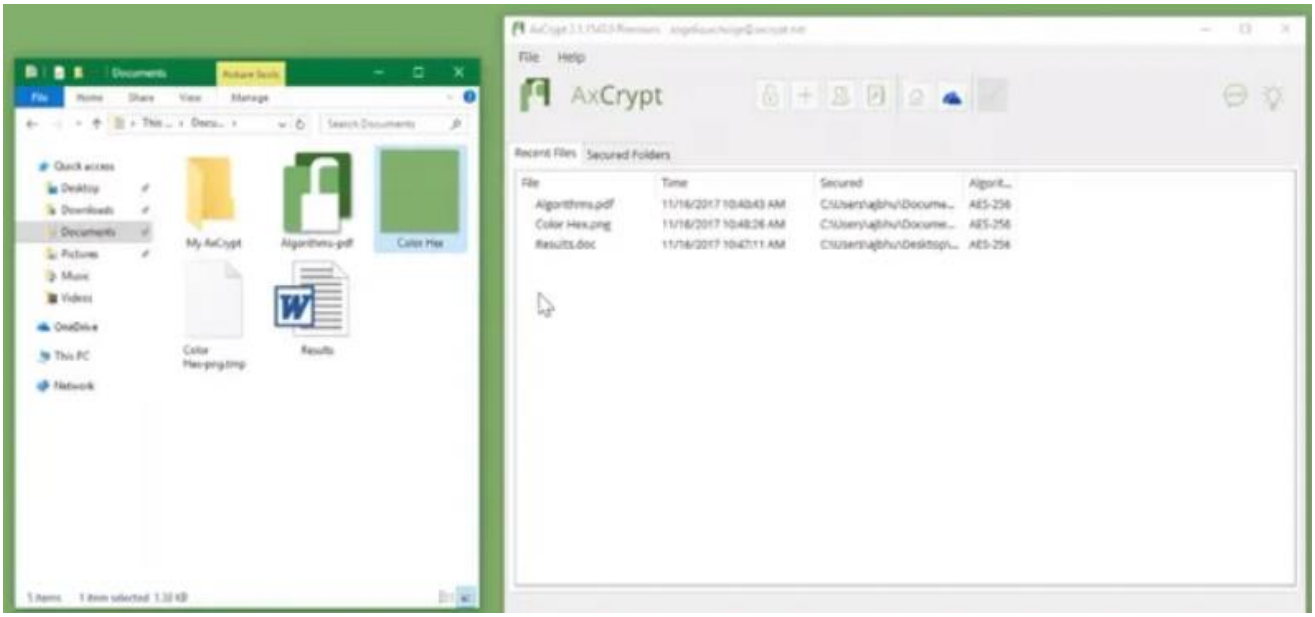

Рисунок 2.6 – Интерфейс программы AxCrypt

Хотя бесплатное программное обеспечение может быть удобным для некоторых, оно не всегда такое мощное, как премиальные предложения, и AxCrypt – хорошая программа, если пользователь хочет что-то надежное. Программное обеспечение было разработано специально для частных лиц и небольших групп в рамках бизнеса.

Она обеспечивает надежную защиту, поскольку файлы защищены 128-битным или 256-битным шифрованием AES, а также шифрованием RC2, что должно помешать любым злоумышленникам. Есть также возможности облачного хранения – программа автоматически защитит файлы, сохраненные в таких сервисах, как Google Drive и Dropbox.

AxCrypt многоязычен и может работать с такими языками, как нидерландский, французский, немецкий, итальянский, корейский, испанский,

шведский, русский и португальский. Кроме того, есть управление паспортами, и можно получить доступ к зашифрованным файлам через приложение для смартфона.

Плюсы AxCrypt:

– Сильное шифрование для личного использования;
– Доступна бесплатная версия.

Минусы AxCrypt:

– Ориентирована на мобильные устройства.

Folder Lock – эффективное шифрование для физических лиц. Интерфейс предоставлен на рисунке 2.7.

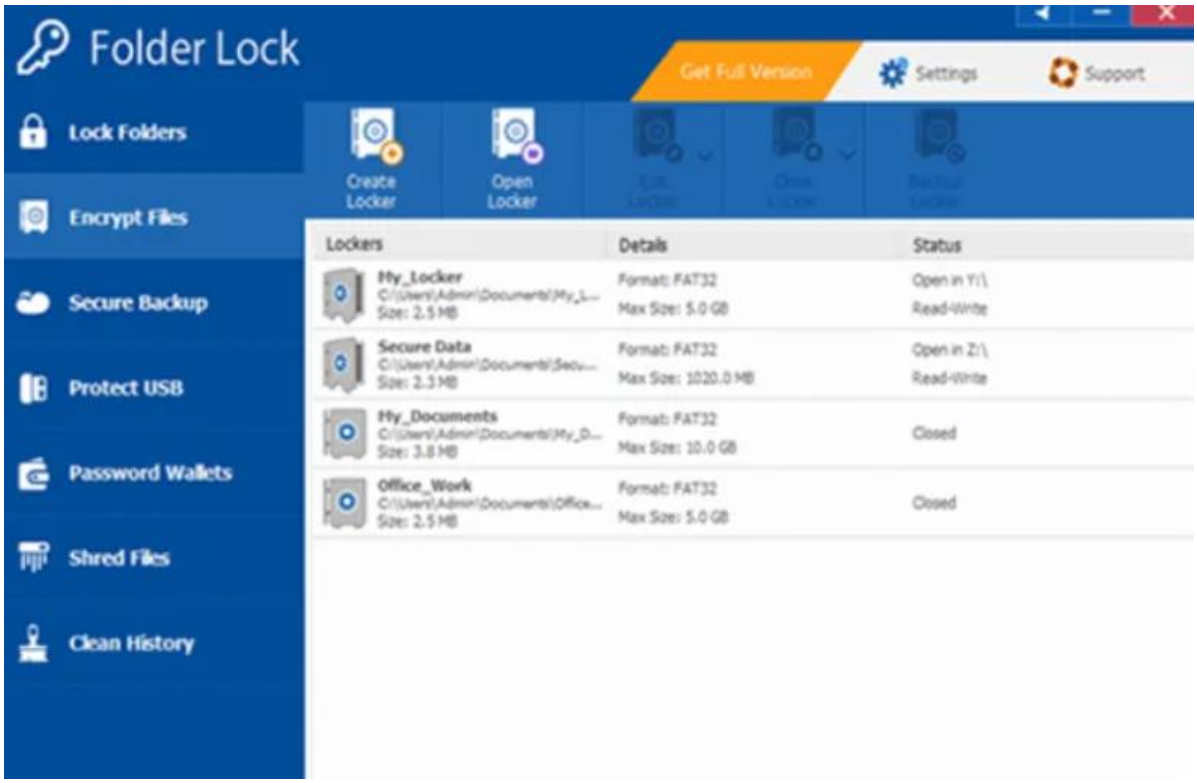

Рисунок 2.7 – Интерфейс программы Folder Lock

Хотя важно защитить активы на компьютерах компании, также важно добавить защиту любому устройству, которое хранит важные данные. Например, большинство сотрудников имеют доступ к электронной почте своей компании и другим учетным записям на своих смартфонах, и они должны быть защищены.

Folder Lock является хорошим вариантом, когда речь идёт о добавлении шифрования на мобильных устройствах. Приложение может защитить личные файлы, фотографии, видео, контакты, заметки и аудиозаписи, хранящиеся в телефоне.

Есть и некоторые другие скрытые функции безопасности. Помимо шифрования, можно установить «пароль-приманку», средства защиты от хакерских атак, регистрировать несанкционированные попытки входа в систему, создать резервную копию всех паролей и получать уведомления о потенциальных атаках методом перебора. Базовое приложение можно загрузить бесплатно, с профессиональной версией доступно больше функций.

Плюсы Folder Lock:

– Бесплатная базовая версия;
– Эффективное личное шифрование.

Минусы Folder Lock:

– Ориентирован на мобильные устройства.

Таким образом, рассмотрев аналоги программ шифрования можно прийти к выводу, что каждая из программ имеет свои особенности, некоторые из них будут полезны как ориентиры и примеры в разработке приложения.

3. Безопасность и криптостойкость алгоритма

Почти сразу после публикации AES вышла работа ряда известных криптологов: Рональда Ривеста, Ларса Кнудсена, Винсента Риджмена и Мэта Робшоу, – в которой исследовалось воздействие дифференциального и линейного криптоанализа на алгоритм AES. Результаты оказались таковы:

– алгоритм не подвержен атаке методом линейного криптоанализа;

– алгоритм может быть теоретически вскрыт методом дифференциального криптоанализа, для чего необходимо наличие не менее 2^{59} пар выбранных открытых текстов и соответствующих им шифртекстов – такая атака вряд ли реализуема на практике.

Не более практичной является атака на связанных ключах, для успешного проведения которой требуется наличие 2^{34} выбранных открытых текстов и соответствующих им шифртектов, причем зашифровывание должно выполняться на ключе, связанном с искомым определенным простым соотношением. Данная атака изобретена не менее известными криптологами: Джоном Келси, Брюсом Шнайером и Дэвидом Вагнером.

Других методов вскрытия алгоритма AES на настоящий момент не известно.

В июне 2003 года Агентство национальной безопасности США постановило, что шифр AES является достаточно надёжным, чтобы использовать его для защиты сведений, составляющих государственную тайну. Вплоть до уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит.

4. Описание программного средства

Для разработки программного средства, была выбрана среда разработки visual studio code, которая на данный момент является лидером по встроенным функциям, позволяющие грамотно отладить проект. Для выбора языка программирования проведем анализ существующих языков программирования и выберем наиболее оптимальный для данного решения.

Рассмотрим популярные и наиболее востребованные языки программирования, используемые на сегодняшний момент:

1. Техническое сообщество не так давно отпраздновало 25-летний юбилей Java. Это один из наиболее широко принятых языков программирования, используемый около 9 миллионами разработчиков, и работает на 7 млрд устройств по всему миру. Это язык программирования, используемый для разработки всех родных приложений Android. Популярность Java-разработчиков исходит из того, что этот язык имеет долгосрочную совместимость, которая гарантирует, что старые приложения продолжат работать и сейчас и в будущем. Единственная сложность заключается в том, что этот язык достаточно сложен в освоении особенно для новичков.

2. JavaScript. Серверные языки сценариев идеально подходят для разработки сложных веб приложений, но каждая такая задача сильно нагружает сервер. Поэтому разработчики делегировали часть функций на сторону клиента и использовали JavaScript. JavaScript – это язык программирования, выполняемый в клиентском браузере и обрабатывает команды на компьютере конечного пользователя, а не сервера, что приводит к снижению нагрузки на сервер и увеличению скорости работы приложения. JavaScript был разработан компанией Netscape и вряд ли есть сайты, которые не используют его.

3. C# принадлежит семье языков программирования Microsoft и был разработан в 2000 году и стал частью первого релиза .NET framework. Язык C# сочетает в себе надежность C++ с дополнительными возможностями Java. Поэтому если программист хорошо знает Java, можно легко переключиться на C# и наоборот. Язык C# позволяет разрабатывать практически любые приложения, которые связаны с Visual Studio IDE [2].

4. Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java и Objective-C. C ценят за его эффективность; он является самым популярным языком для создания системного программного обеспечения. Изучение этого языка ведет к пониманию и других языков. Язык C используется для разработки низкоуровневых приложений, так как считается ближе всего к аппаратному, уступая только ассемблеру [7].

5. Python – высокоуровневый язык программирования, который часто считается самым легким языком благодаря своей простоте, читаемости и синтаксису. 8 из 10 факультетов информатики в США и 27 из 39 лучших университетов используют Python для обучения студентов программированию.

6. PHP – это один из наиболее широко используемых языков для разработки динамических веб сайтов. PHP открытый язык разработки, поэтому написаны уже

тысячи модулей, которые можно модифицировать до требуемой функциональности. На PHP разработано большинство сайтов, ориентированных на большой объём данных.

В результате исследования приложение будет написано на языке программирования C#, так как данный язык является одновременно и гибким, и быстрокомпилируемым языком, что для данной задачи является чрезвычайно важным. Используемая среда MS Visual Studio имеет бесплатную версию, т.е. для разработчика не потребуется никаких материальных затрат, связанных с написанием программы.

Для реализации утилиты шифрования исполняемых файлов был создан отдельный пакет классов Cryptography с методами такие как, как:

- AesCryptography
  - CreateIv
  - Encrypt
  - Decrypt
  - Validate
  - isOneOf
- CryptedData
  - ToArray
  - Store
  - Create
  - Validate
- CryptographyHelper
  - Encrypt
  - Decrypt
  - GetKey
- InsecureString
  - insecureString
  - Initialize
  - Dispose
  - GetEnumerator
  - IEmenurable
  - ExecuteIn

Все методы можно посмотреть на рисунке 4.1. Как видно, не каждый метод принимает входные параметры, т.к. некоторые отвечают за общую обработку глобальных данных классов, а некоторые выполняют сложные операции.

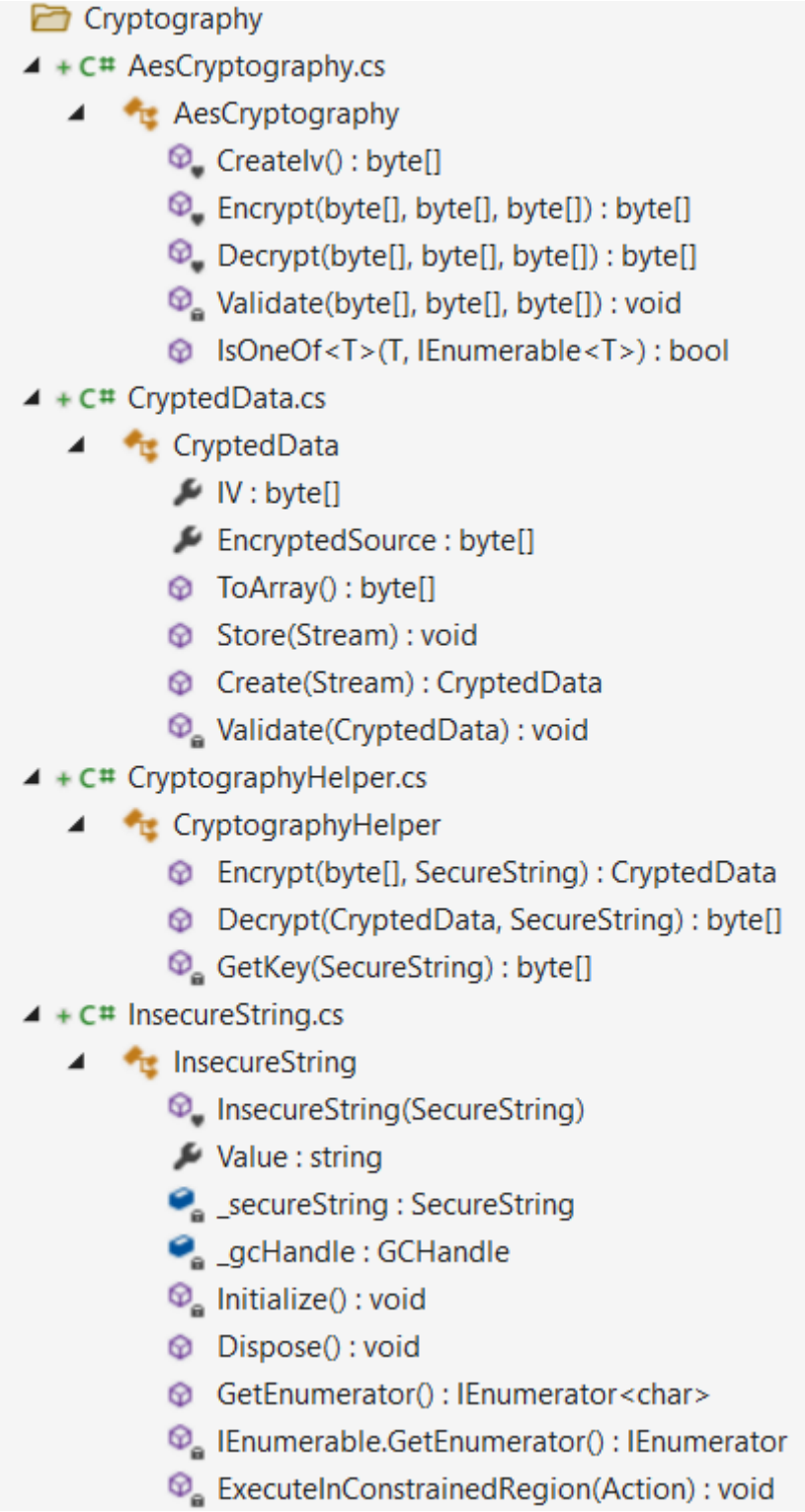

Рисунок 4.1 – Пакет классов Cryptography

Разработанная утилита является пользователем данного пакета, однако внутри проекта утилиты были созданы два вспомогательных класса Program (рис. 4.2) и Wiper (рис. 4.3) с их вспомогательными методами.

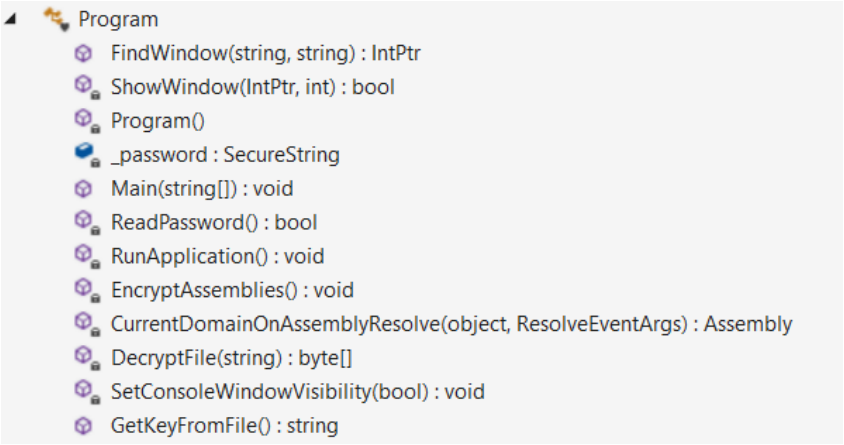

Рисунок 4.2 – Методы, содержащиеся в классе Program

Класс Program является отправной точкой утилиты, а нем происходит управление всеми процессами, как запуск и работа утилиты, так и запуск приложения, которое мы необходимо защитить.

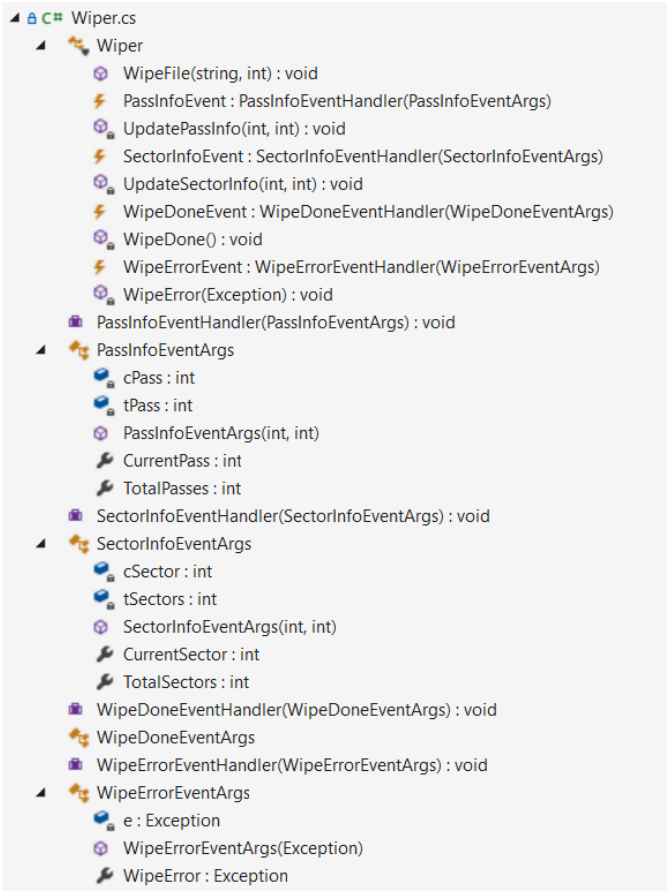

Рисунок 4.3 – Методы, содержащиеся в классе Wiper

Для дизайна элементов управления программного средства была использована библиотека MaterialDesign – библиотека пользовательского интерфейса, разработанная компанией Google.

5. Описание этапов разработки утилиты

Защита приложения будет строиться на шифровании сборок симметричным ключом и динамическом их дешифровании в процессе работы приложения. Ключ шифрования будет определяться пользователем на этапе развёртывания и вводиться как пароль при запуске. Разобьём всё на этапы:

1. Предварительные работы
2. Ввод пароля
3. Дешифровка сборок
4. Переопределение загрузки сборок
5. Запуск утилиты
6. Финальные приготовления
7. Дополнительные настройки проекта
8. Развёртывание и шифрование сборок

Все листинги к методам, находятся в приложении А.

5.1 Предварительные работы

Работа утилиты заключается в динамическом шифровании и дешифровании исполняемых сборок, соответственно, исходя из поставленной задачи, целесообразным ее решением будет выбор создания обертки, которая, будет динамически расшифровывать сборки приложения во время работы, в качестве обертки будет выбрано обычное консольное приложение.

5.2 Ввод пароля

Вводимый пароль так или иначе необходимо сохранять. Обычно для этих целей используются строки, но в .net они неизменяемы, а значит, введённый пароль можно без трудностей исследовать отладчиком проектов. Чтобы избежать этого, воспользуемся специальным классом SecureString (пространство имён System.Security), который хранит свои данные в зашифрованном виде. При вводе на экране не будут отображаться вводимые символы, а после нажатия Enter ввод закончится. Для считывания вводимого пароля, был разработан класс ReadPassword(), который использует _password – поле класса Program, в котором сохраняется введённый пользователем пароль.

5.3 Дешифровка сборок

Шифрование делится на два типа: симметричное и асимметричное. В симметричном для шифрования и дешифрования используется один и тот же ключ, в асимметричном разные. Поскольку нам разные ключи не нужны, остановимся на симметричном шифровании. Чтобы расшифровать что-то зашифрованное, нам нужны три компонента:

– Зашифрованные данные;
– Ключ, которым они зашифрованы;

– Вектор инициализации (IV) – несекретные данные, которые использовались для первого этапа шифрования.

Поскольку вектор инициализации не секретен, его можно хранить вместе с зашифрованными данными. Для облегчения работы мною был написан специальный класс CryptedData. Шифровать сборки мы будем используя алгоритмом AES. Для удобства создадим низкоуровневую обёртку (класс AesCryptography). Для большей модульности был создан класс CryptographyHelper. Данный класс содержит метод GetKey, данный метод является большим помощником для утилиты, так как он решает ряд некоторых вопросов.

Первый момент – длина ключа должна быть равна 128, 192 или 256 битам. А в качестве пароля для запуска может быть строка произвольной длины. Поэтому просто хешируем строку пароля с помощью sha256 и получаем искомую длину [5]. Второй момент покруче и связан с SecureString. Этот класс доступен только на запись, и чтобы получить его содержимое нам необходимо воспользоваться небезопасным (unsafe) кодом. Для этого был написан класс InsecureString.

InsecureString является одинм из самых важных классов данной утилиты. Он выполняет следующие функции:

1. Подготавливаются два кода, один основной, который делает всю работу, второй – код очистки в случае исключения;

2. Основной код создаёт новую строку, в которой будет храниться значение SecureString и которая будет принудительно очищена в методе Dispose;

3. Копирует данные из SecureString во внутреннюю строку через указатели и блокирует внутреннюю строку для сборщика мусора;

4. Через внутреннюю строку мы можем получить данные SecureString [3].

В методе Dispose происходит затирание внутренней строки через указатели. Важно держать «окно жизни» экземпляра InsecureString как можно более коротким, чтобы минимизировать риск чтения данных защищённой строки отладчиком. Хеширование, описанное выше, помогает в этом, поскольку экземпляр InsecureString нам нужен только для получения хеша, а дальше мы работаем с самим хешем, из которого не вытащить первоначальное значение SecureString [6].

5.4 Переопределение загрузки сборок

Поскольку мы планируем использовать зашифрованные сборки, нам нужно поменять стандартный механизм их загрузки. За загрузку сборок отвечает домен приложения (AppDomain), через специальное событие AssemblyResolve в методе CurrentDomainOnAssemblyResolve, который в свою очередь вызовет метод DecryptFile. К моменту входа в Main они могут уже понадобиться, переопределять механизм будем раньше, в конструкторе типа [6]. Там же, для удобства, сделаем обработку исключений.

5.5 Запуск утилиты

Для запуска утилиты был создан класс RunApplication, который в свою очередь запускает уже расшифрованные сборки и скрывает консоль, в которой вводился пароль

5.6 Финальные приготовления

Осталось два момента:
1. Скрывать окно консоли перед появлением приложения;
2. Маскировать само наличие приложения

Для сокрытия окна консоли нам понадобится импортировать неуправляемые методы с помощью [DllImport("user32.dll")]: public static extern IntPtr FindWindow(string lpClassName, string lpWindowName) и static extern bool ShowWindow(IntPtr hWnd, int nCmdShow). Затем необходимо скрыть само окно, для этого мною был разработан класс SetConsoleWindowVisibility [3].

Для маскирования утилиты и приложения, которое ей защищается, был создан класс Main, в котором использовался префикс [STAThread] , класс Main в свою очередь делает первичную ловушку для пользователя. Он выводит несуществующую ошибку, чтобы создать видимость неработающего приложения [4].

5.7 Дополнительные настройки проекта

Для корректной работы, в сборке загрузчика должен быть разрешён небезопасный код. Проще всего это сделать через настройки проекта (рис. 5.1) [1].

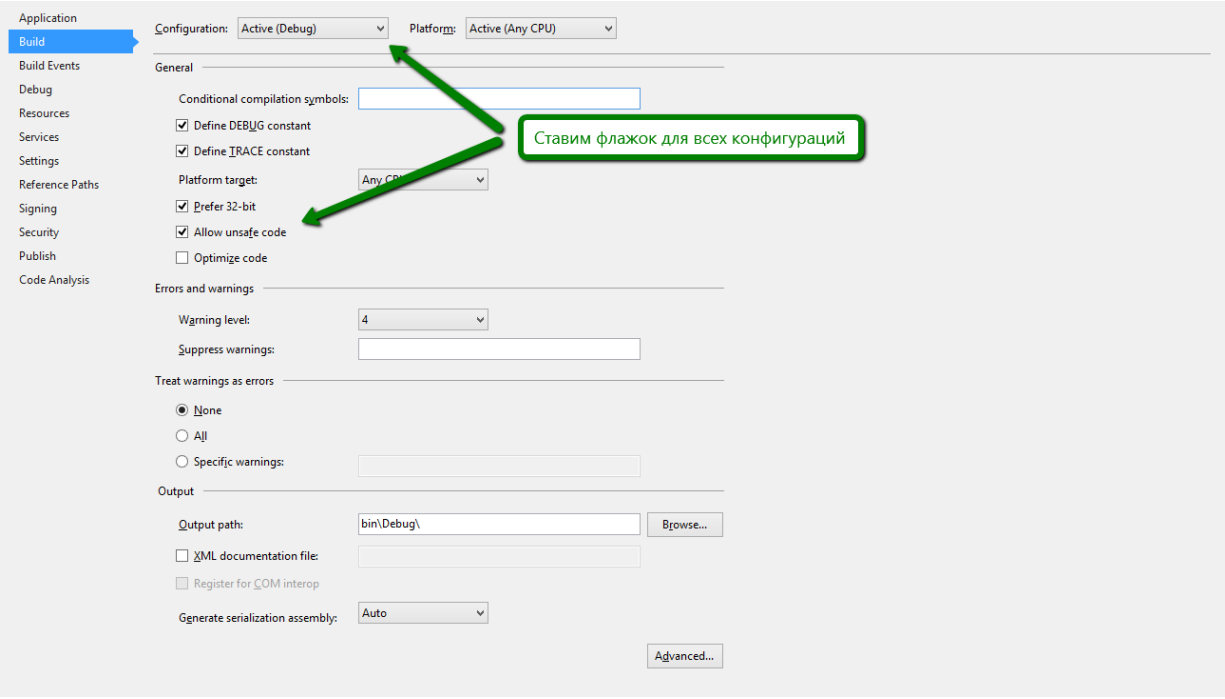

Рисунок 5.1 – настройка разрешение утилиты на небезопасный код

5.8 Развертывание и шифрование сборок

Для зашифровывания сборок сразу после компиляции, мной были написан метод EncryptAssemblies, который в свою очередь копирует содержимое файлов сборок, шифрует это содержимое и генерирует новые файлы сборок с идентичными названиями, далее удаляет оригинальные сборки и вставляет новые файлы, уже зашифрованные. Для того, чтобы удалять оригинальные файлы сборок, был разработан вспомогательный класс Wiper, который в несколько проходов перезаписывает файл случайными данными, а затем удаляет.

6. Тестирование программного средства

Протестируем разработанное программное средство. Стоит заметить, что ключ-пароль необходимо хранить на диске D в файле “text.txt”. Далее проверим программу на запуск. Для этого запустим исполняемый файл программы. После чего мы увидим сообщение на консоли, которое мы выводим специально, для видимости того, что приложение не работает рисунок 6.1.

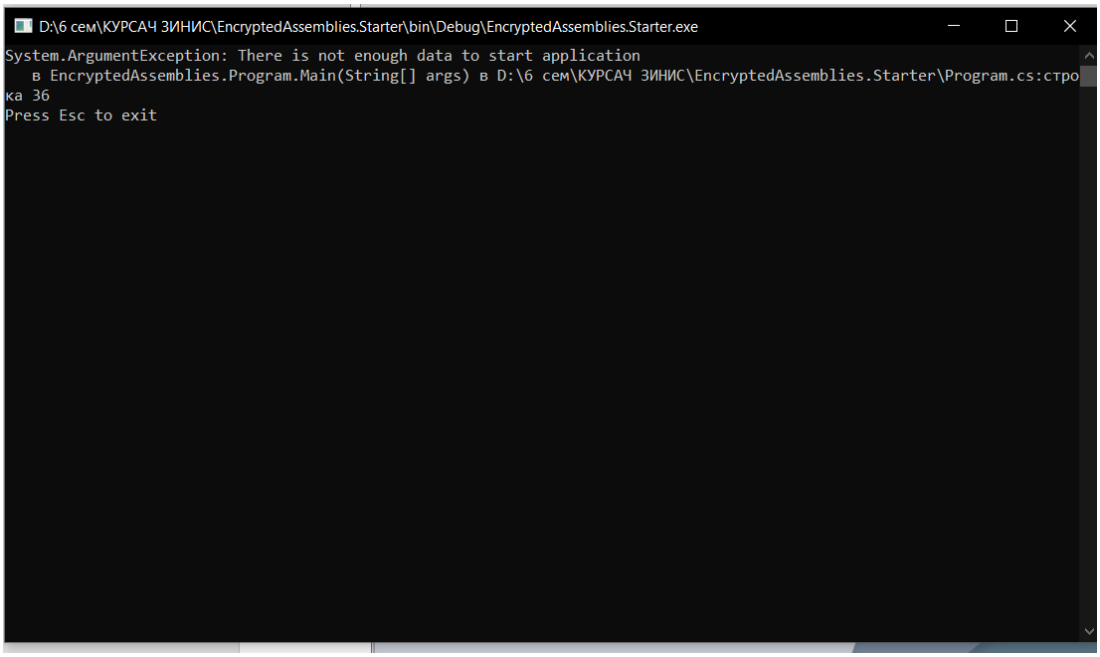

Рисунок 6.1 – Сообщение на консоли о том, что оно запустилось с ошибкой

Как видно из рисунка 6.1 утилита успешно запустилась, далее введем неправильный пароль, результат показан на рисунке 6.2.

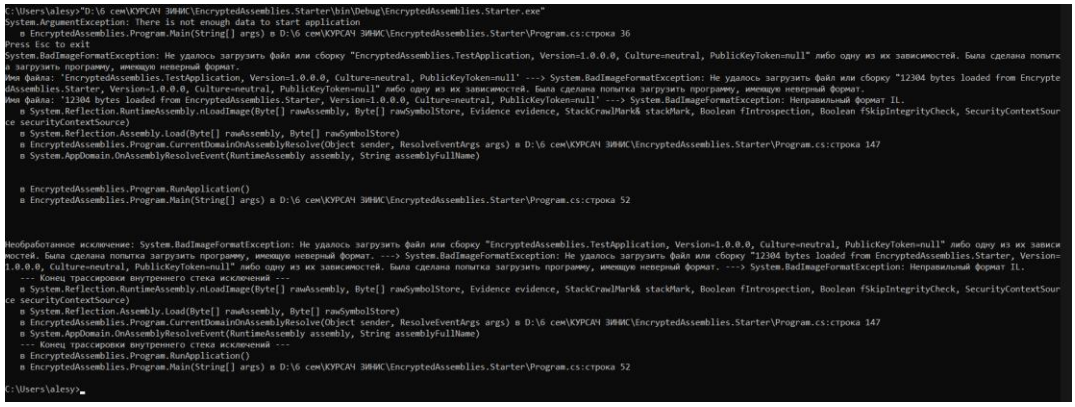

Рисунок 6.2 – Ввод неправильного ключа

Далее при вводе правильного пароля, срабатывает функция открытия приложения, которое необходимо было защитить, результат показан на рисунке 6.3

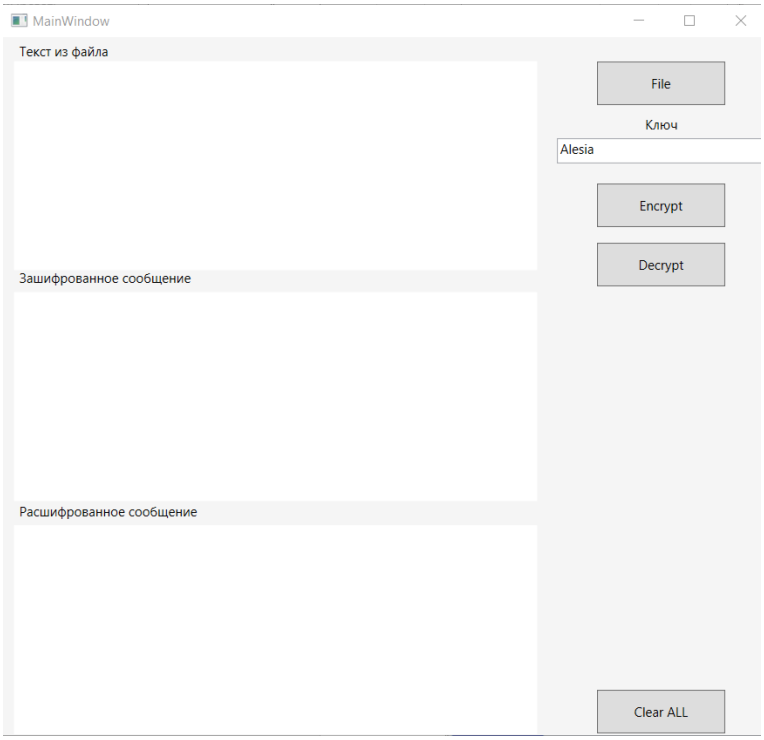

Рисунок 6.3 – Ввод правильного ключа

Как видно из рисунка 6.2 при неправильном вводе ключа, мы получаем ошибку запуска, которая возникает в связи с тем, что сборочные файлы были расшифрованы неверно. На рисунке 6.3 мы видим результат ввода правильного ключа: открывается тестовое приложение.

Далее необходимо выбрать файл, из которого будет взят текст для зашифрования. Для этого необходимо нажать на кнопку “File” и выбрать необходимый файл с исходным текстом. После этого необходимо ввести ключ в поле для ввода ключа. И нажать Encrypt для шифрования текста и Decrypt для расшифрования текста. Результат этих манипуляций показан на рисунке 6.4.

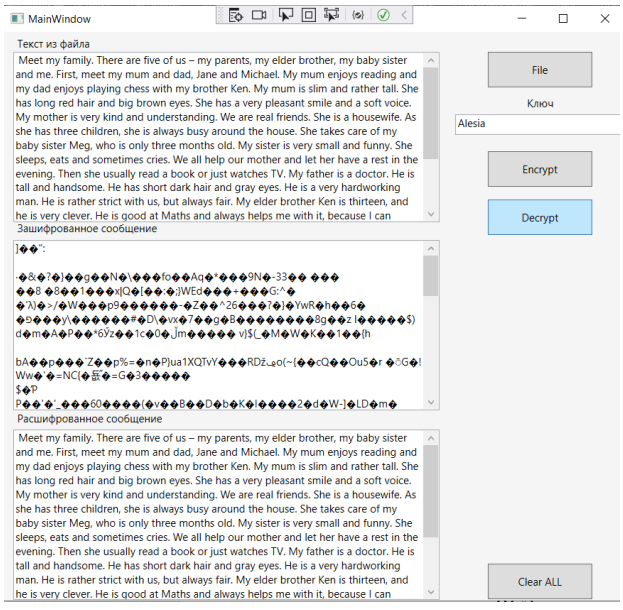

Рисунок 6.4 Результат шифрования и дешифрования сообщения из файла

Если для дешифрования файла написать другой ключ, мы получим следующий результат:

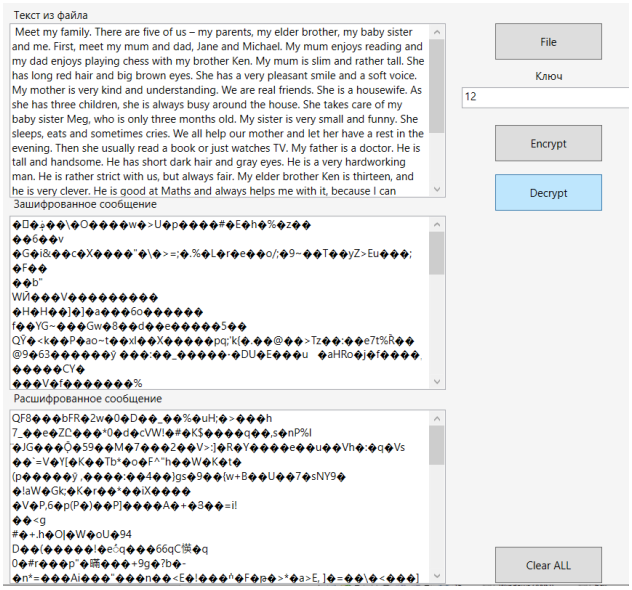

Рисунок 6.5. Результат дешифрования сообщения с неправильным ключом

Откроем файл, в который зашифровали текст через программу «HxD» и посмотрим данные. Результат на рисунке 6.6.

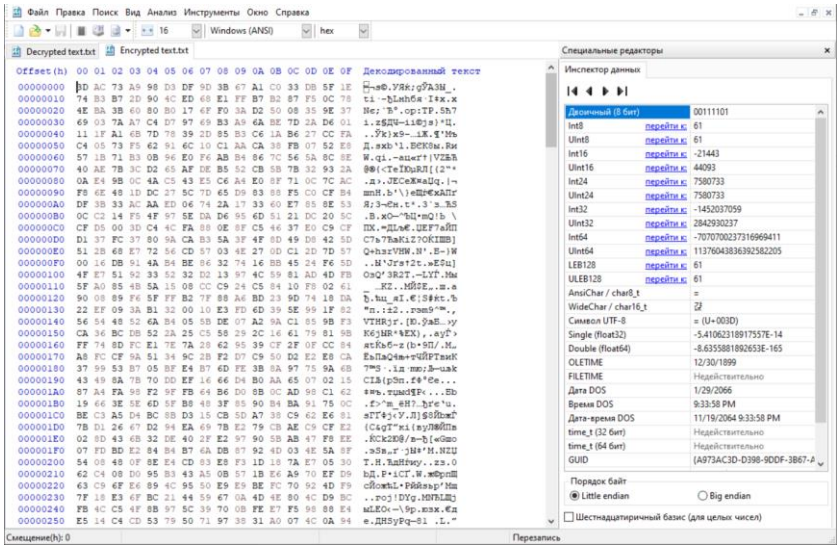

Рисунок 6.6 – Зашифрованный текст в файле

Нажмем на кнопку «Decrypt», и данные из файла будут расшифрованы в другой файл. После открытия в той же программе видим результат расшифровки на картинке 6.7.

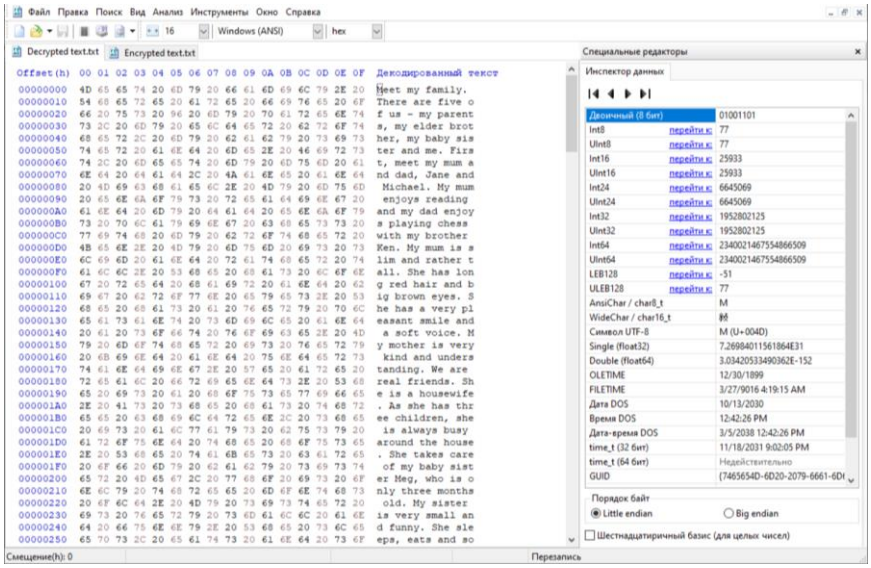

Рисунок 6.7.– Расшифрованный текст в файле

Таким образом, при проведенном тестировании можно говорить о том, что разработанная утилита успешно шифрует и расшифровывает файлы сборки, а также успешно предотвращает ошибочный и некорректный ввод данных, преобразовывая ключи хешированием.

7. Руководство пользователя

Для работы с утилитой необходимо создать на диске D текстовый файл “test.txt” в который поместить ключ для шифрования файлов сборки. После чего запустить программу. Откроется консоль, в которой можно будет ввести ключ для расшифровки файлов. Рисунок 7.1

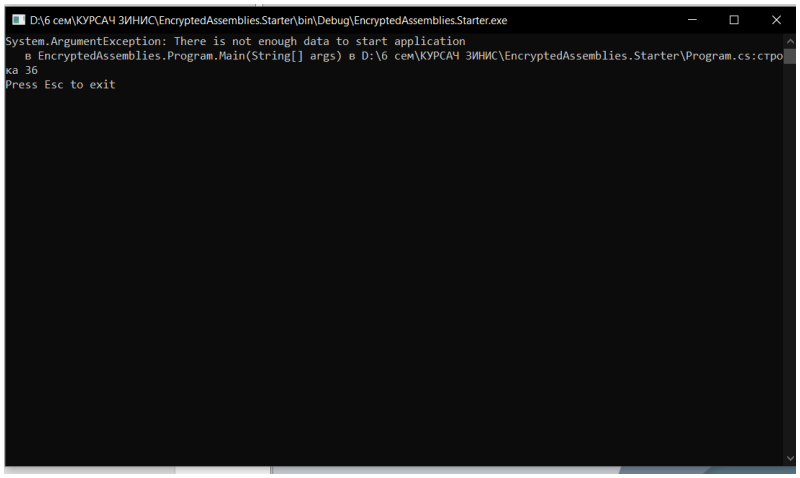

Рисунок 7.1 – Стартовое состояние утилиты

Как видно из рисунка 7.2 после успешного ввода ключа, вы сможете увидеть программу для наглядного показа работы утилиты.

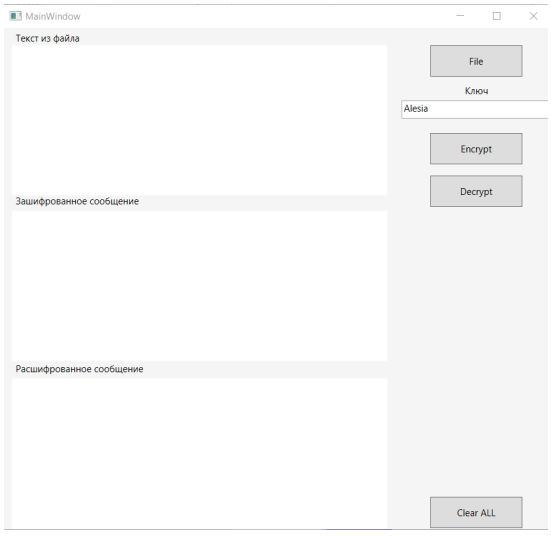

Рисунок 7.2 – Стартовое окно после успешного ввода ключа

Далее требуется ввести ключ для шифрования, ключ может быть абсолютно любым, даже пустым. На рисунке 7.3 поле для ввода ключа обведено красным овалом. Далее необходимо нажать кнопку “File” и выбрать нужный файл с текстом для шифрования. Поле выбора файла вы сможете увидеть его содержимое в верхнем поле с подписью “текст из файла” рисунок 7.3. Затем после нажатия на кнопку “Encrypt” в поле с подписью “Зашифрованное сообщение” выведется зашифрованный текст ключом который мы ввели заранее.

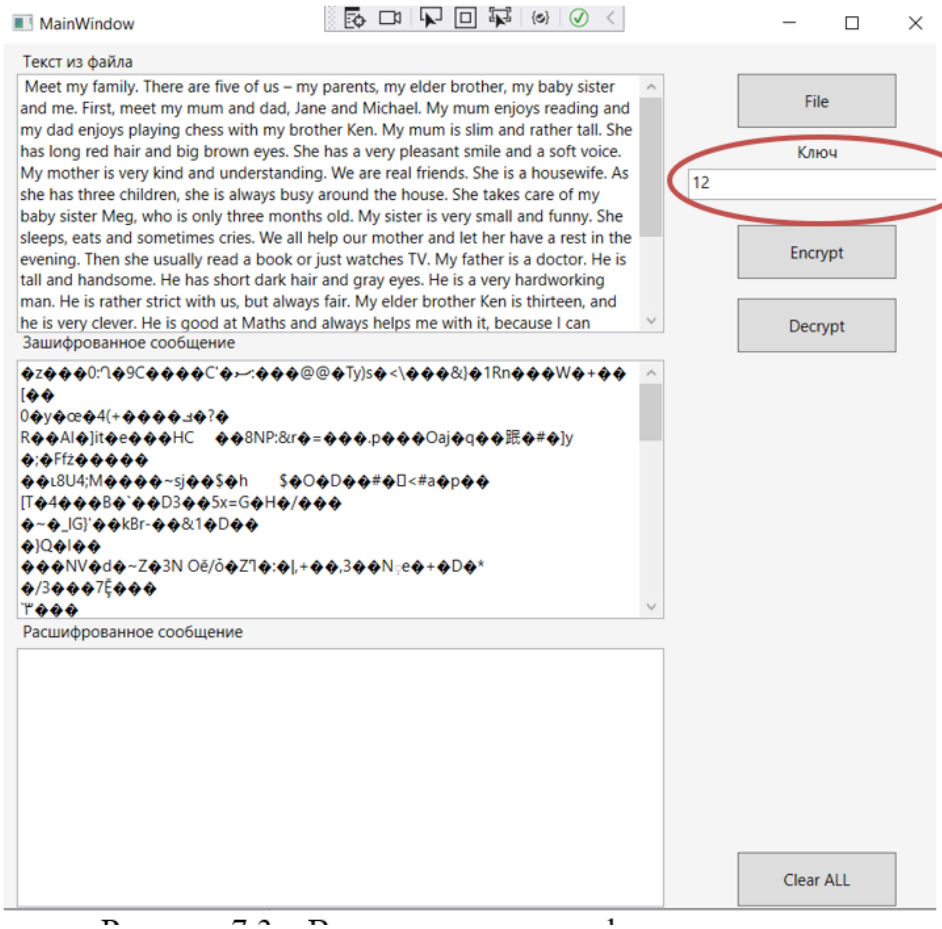

Рисунок 7.3 – Ввод содержимого файла для шифрования, а также результат его шифрования

Для расшифровки сообщения необходимо нажать кнопку “Decrypt” после чего вы увидите результат расшифровки сообщения. Рисунок 7.4

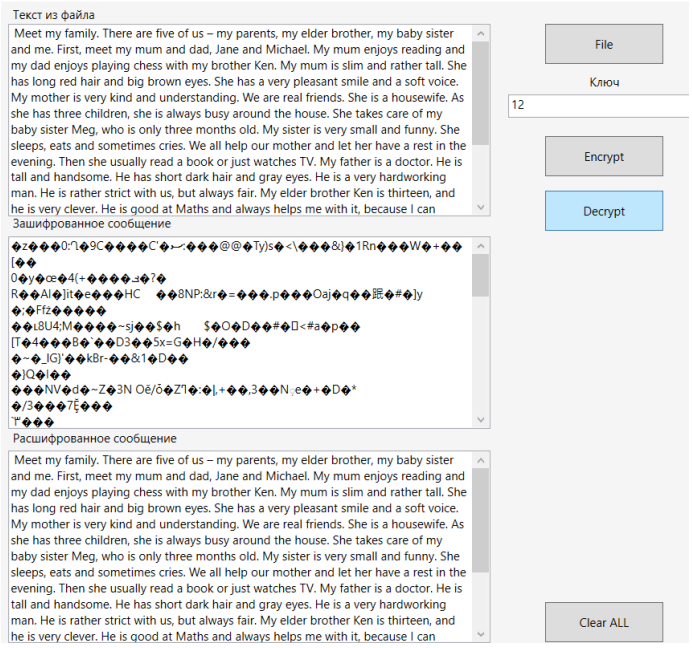

Рисунок 7.4 Результат расшифрования сообщения

Если поменять ключ, то результат расшифровки также изменится. Результат показан на рисунке 7.5.

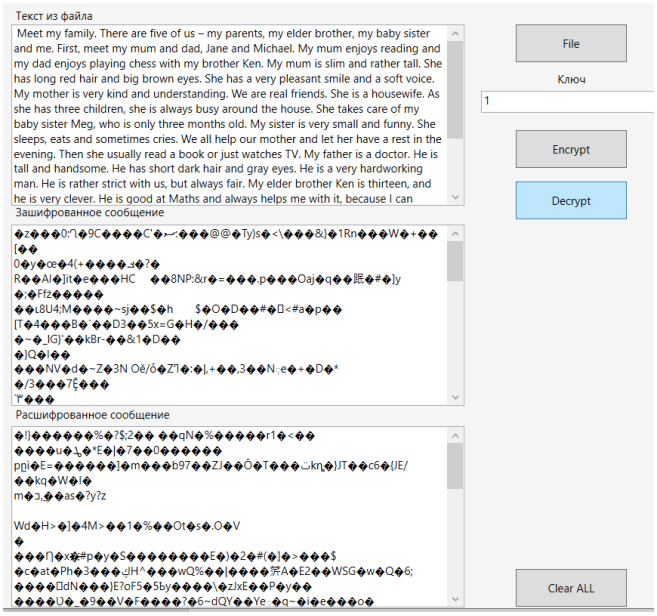

Рисунок 7.5 Результат расшифровки сообщения с ключом отличный от ключа шифрования

Чтобы отчистить поля с шифрованием и дешифрованием необходимо нажать кнопку «Clear All», данная кнопка отчистит поля «Зашифрованное сообщение» и «Расшифрованное сообщение». Чтобы выбрать новый файл, необходимо нажать кнопку «File» и выбрать новый файл для шифрования.

Заключение

В результате выполнения данной работы мною была разработана программная утилита, позволяющая зищитить ПО, написанное на языке программирования c#, а также была разработанна тестовая программа, наглядно показывающая работу утилиты, с помощью шифрования и расшифрования текстовых файлов.

Утилита может:

– Получать доступ к исполняющим файлам сборки

– Шифровать содержимое этих файлов

– Замещать файлы сборки

– Адаптировать любые ключи для шифрования под нужный формат хешированием

– Дешифровывать файлы сборки

– Управлять защищенными строками, хранящие ключ-пароль

Тестовая программа может:

– шифровать содержимое текстовых файлов алгоритмом шифрования файлов сборки;

– расшифровывать содержимое файлов;

– отображать ввод ключа;

– отображать результаты шифрования;

– отображать результаты расшифрования;

В результате курсовой работы были выполнены следующие задачи:

1. Был проведен аналитический обзор литературы а также поиск и сравнение аналогов.
2. Были спроектированы и разработаны утилита шифрования файлов и тестовое программное средство.
3. Были подробно описаны этапы разработки утилиты.
4. Разработанные программные средства были описаны.
5. Проведено тестирование разработанного программного средства.
6. Написано руководство пользователя разработанного программного средства.

Программу можно усовершенствовать путем добавления других алгоритмов шифрования, к примеру DES, RC5 и другие, а также динамической генерации ключей ввода самой программой, а не пользователем.

Список используемых источников

1. Гарнаев А. Самоучитель Visual Studio. – СПб.: BHV, 2012.
2. Прайс Джейсон, Гандэрлой Майк Visual C# 2.0. Полное руководство; Век +, Корона-Век, Энтроп – М., 2007. – 736 с.
3. Культин Никита Борисович Основы программирования в Microsoft Visual C# 2010 (+ CD-ROM); БХВ-Петербург – М., 2011. – 384 с.
4. Культин Никита Основы программирования в Microsoft Visual C# 2010; БХВ-Петербург – М., 2011. – 634 с.
5. Панасенко С.П. Алгоритмы шифрования. БХВ-Петербург, 2009 – 578 с.
6. Петцольд Чарльз Программирование для Microsoft Windows; Питер – М., 2014. – 274 с.
7. Понамарев Вячеслав Программирование на C++/C# в Visual Studio .NET 2003; БХВ-Петербург – М., 2004. – 352 с.

ПРИЛОЖЕНИЕ А

AesCryptography.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;

namespace EncryptedAssemblies.Starter.Cryptography
{
   public static class AesCryptography
   {
      /// <summary>
      /// Возвращает вектор инициализации.
      /// </summary>
      /// <returns></returns>
      internal static byte[] CreateIv()
      {
         using (AesManaged aes = new AesManaged())
         {
            aes.GenerateIV();
            return aes.IV;
         }
      }

      /// <summary>
      /// Зашифровывает данные.
      /// </summary>
      /// <param name="source">Данные.</param>
      /// <param name="key">Ключ шифрования.</param>
      /// <param name="iv">Вектор инициализации.</param>
      /// <returns>Зашифрованные данные.</returns>
      internal static byte[] Encrypt(byte[] source, byte[] key, byte[] iv)
      {
         Validate(source, key, iv);
         using (AesManaged aes = new AesManaged())
         {
            using (ICryptoTransform transform = aes.CreateEncryptor(key, iv))
            {
               using (MemoryStream ms = new MemoryStream())
               {
                  using (CryptoStream cs = new CryptoStream(ms, transform,
CryptoStreamMode.Write))
                  {
```

```
                    cs.Write(source, 0, source.Length);
                }
                byte[] encryptedBytes = ms.ToArray();

                return encryptedBytes;
            }
        }
    }
}

/// <summary>
/// Расшифровывает текст.
/// </summary>
/// <param name="source">Данные для расшифровки.</param>
/// <param name="key">Ключ шифрования.</param>
/// <param name="iv">Вектор инициализации.</param>
/// <returns>Расшифрованные данные.</returns>
internal static byte[] Decrypt(byte[] source, byte[] key, byte[] iv)
{
    try
    {
        Validate(source, key, iv);
        using (AesManaged aes = new AesManaged())
        {
            using (ICryptoTransform transform = aes.CreateDecryptor(key, iv))
            {
                using (MemoryStream ms = new MemoryStream(source))
                {
                    using (CryptoStream cs = new CryptoStream(ms, transform,
CryptoStreamMode.Read))
                    {
                        List<byte> bytes = new List<byte>(1024);
                        int b;
                        while ((b = cs.ReadByte()) != -1)
                        {
                            bytes.Add((byte)b);
                        }
                        return bytes.ToArray();
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
```

```
            Random rnd = new Random();
            Byte[] b = new Byte[source.Length];
            rnd.NextBytes(b);
            return b;
        }
    }

    /// <summary>
    /// Проверяет данные.
    /// </summary>
    /// <param name="source">Данные.</param>
    /// <param name="key">Ключ шифрования.</param>
    /// <param name="iv">Вектор инициализации.</param>
    private static void Validate(byte[] source, byte[] key, byte[] iv)
    {
        if (source == null)
        {
            throw new ArgumentNullException("source");
        }
        else if (source.Length == 0)
        {
            throw new ArgumentException("Данные не могут быть пустыми",
"source");
        }
        if (key == null)
        {
            throw new ArgumentNullException("key");
        }
        else if (key.Length == 0)
        {
            throw new ArgumentException("Ключ не может быть пустым", "key");
        }
        //if (key.Length.IsOneOf(16, 24, 32) == false)
        //{
        //    throw new ArgumentException("Длина ключа должна быть 128, 192
или 256 бит (16, 24, 32 байта)", "key");
        //}
        if (iv == null)
        {
            throw new ArgumentNullException("iv");
        }
        else if (iv.Length != 16)
        {
            throw new ArgumentException("Длина вектора инициализации должна
быть 128 бит (16 байт)", "iv");
```

```
        }
    }

    public static bool IsOneOf<T>(this T value, IEnumerable<T> values)
    {
        if (values == null)
        {
            throw new ArgumentNullException("values");
        }
        foreach (T t in values)
        {
            if (Equals(t, value))
            {
                return true;
            }
        }
        return false;
    }
  }
}
```

CryptedData.cs

```
using System;
using System.IO;

namespace EncryptedAssemblies.Starter.Cryptography
{
public sealed class CryptedData
{
   /// <summary>
   /// Возвращает или устанавливает вектор инициализации.
   /// </summary>
   public byte[] IV
   {
      get;
      set;
   }

   /// <summary>
   /// Возвращает или устанавливает данные.
   /// </summary>
   public byte[] EncryptedSource
   {
      get;
```

```
            set;
        }

        /// <summary>
        /// Возвращает вектор инициализации и зашифрованные данные в виде
единого массива.
        /// </summary>
        public byte[] ToArray()
        {
            using (MemoryStream ms = new MemoryStream())
            {
                Store(ms);
                return ms.ToArray();
            }
        }

        /// <summary>
        /// Сохраняет вектор инициализации и зашифрованные данные в поток.
        /// </summary>
        /// <param name="output">Поток, в который необходимо сохранить
данные.</param>
        public void Store(Stream output)
        {
            Validate(this);
            if (!output.CanWrite)
            {
                throw new ArgumentException("В переданный поток запрещена запись",
"output");
            }

            using (BinaryWriter bw = new BinaryWriter(output))
            {
                bw.Write(IV.Length);
                bw.Write(IV);
                bw.Write(EncryptedSource.Length);
                bw.Write(EncryptedSource);
            }
        }

        /// <summary>
        /// Возвращает зашифрованные данные и вектор инициализации.
        /// </summary>
        /// <param name="input">Поток с входящими данными.</param>
        public static CryptedData Create(Stream input)
        {
```

```
        if (!input.CanRead)
        {
            throw new ArgumentException("Из входящего потока запрещено чтение",
"input");
        }
        CryptedData data = new CryptedData();
        using (BinaryReader reader = new BinaryReader(input))
        {
            int ivLength = reader.ReadInt32();
            data.IV = reader.ReadBytes(ivLength);
            int sourceLength = reader.ReadInt32();
            data.EncryptedSource = reader.ReadBytes(sourceLength);
        }
        Validate(data);
        return data;
    }

    /// <summary>
    /// Проверяет валидность данных.
    /// </summary>
    /// <param name="data">Данные, которые необходимо проверить.</param>
    private static void Validate(CryptedData data)
    {
        if (data.IV == null || data.IV.Length == 0)
        {
            throw new ArgumentException("IV должно быть ненулевой длинны");
        }
        if (data.IV.Length > byte.MaxValue)
        {
            throw new ArgumentException("Длинна IV не может быть больше " +
byte.MaxValue);
        }
        if (data.EncryptedSource == null || data.EncryptedSource.Length == 0)
        {
            throw new ArgumentException("Souce должно быть ненулевой длинны");
        }
    }
}
```

CryptographyHelper.cs

```
using System;
using System.Security;
using System.Security.Cryptography;
using System.Text;

namespace EncryptedAssemblies.Starter.Cryptography
```

```
{
   public static class CryptographyHelper
   {
      /// <summary>
      /// Зашифровывает строку.
      /// </summary>
      /// <param name="source">Исходные данные.</param>
      /// <param name="password">Ключ шифрования.</param>
      /// <returns></returns>
      public static CryptedData Encrypt(byte[] source, SecureString password)
      {
         byte[] iv = AesCryptography.CreateIv();
         byte[] key = GetKey(password);
         byte[] encrypted = AesCryptography.Encrypt(source, key, iv);

         return new CryptedData()
         {
            EncryptedSource = encrypted,
            IV = iv
         };
      }

      /// <summary>
      /// Расшифровывает данные.
      /// </summary>
      /// <param name="data">Данные, которые необходимо
расшифровать.</param>
      /// <param name="password">Пароль шифрования.</param>
      /// <returns>Расшифрованные данные.</returns>
      public static byte[] Decrypt(CryptedData data, SecureString password)
      {
         byte[] key = GetKey(password);
         byte[] decrypted = AesCryptography.Decrypt(data.EncryptedSource, key,
data.IV);
         return decrypted;
      }

      /// <summary>
      /// Дополняет длину ключа при необходимости.
      /// </summary>
      /// <param name="key">Ключ, который необходимо дополнить.</param>
      /// <returns></returns>
      private static byte[] GetKey(SecureString key)
      {
         using (InsecureString insecure = new InsecureString(key))
```

```
        {
            using (SHA256Managed sha256 = new SHA256Managed())
            {
                byte[] rawKey = new byte[key.Length];
                int i = 0;
                foreach (char c in insecure)
                {
                    rawKey[i++] = Convert.ToByte(c);
                }

                byte[] hashedKey = sha256.ComputeHash(rawKey);
                Array.Clear(rawKey, 0, rawKey.Length);

                return hashedKey;}}}}}
```

InsecureString.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Security;

namespace EncryptedAssemblies.Starter.Cryptography
{
   [CLSCompliant(false)]
   public sealed class InsecureString : IDisposable, IEnumerable<char>
   {
      internal InsecureString(SecureString secureString)
      {
         _secureString = secureString;
         Initialize();
      }

      public string Value { get; private set; }

      private readonly SecureString _secureString;
      private GCHandle _gcHandle;

#if !DEBUG
      [DebuggerHidden]
#endif
      private void Initialize()
      {
         unsafe
         {
```

```
            // We are about to create an unencrypted version of our sensitive string and
store it in memory.
            // Don't let anyone (GC) make a copy.
            // To do this, create a new gc handle so we can "pin" the memory.
            // The gc handle will be pinned and later, we will put info in this string.
            _gcHandle = new GCHandle();
            // insecurePointer will be temporarily used to access the SecureString
            IntPtr insecurePointer = IntPtr.Zero;
            RuntimeHelpers.TryCode code = delegate
            {
                // create a new string of appropriate length that is filled with 0's
                Value = new string((char)0, _secureString.Length);
                // Even though we are in the ExecuteCodeWithGuaranteedCleanup,
processing can be interupted.
                // We need to make sure nothing happens between when memory is
allocated and
                // when _gcHandle has been assigned the value. Otherwise, we can't
cleanup later.
                // PrepareConstrainedRegions is better than a try/catch. Not even a
threadexception will interupt this processing.
                // A CER is not the same as ExecuteCodeWithGuaranteedCleanup. A
CER does not have a cleanup.

                Action alloc = delegate { _gcHandle = GCHandle.Alloc(Value,
GCHandleType.Pinned); };
                ExecuteInConstrainedRegion(alloc);

                // Even though we are in the ExecuteCodeWithGuaranteedCleanup,
processing can be interupted.
                // We need to make sure nothing happens between when memory is
allocated and
                // when insecurePointer has been assigned the value. Otherwise, we can't
cleanup later.
                // PrepareConstrainedRegions is better than a try/catch. Not even a
threadexception will interupt this processing.
                // A CER is not the same as ExecuteCodeWithGuaranteedCleanup. A
CER does not have a cleanup.
                Action toBSTR = delegate { insecurePointer =
Marshal.SecureStringToBSTR(_secureString); };
                ExecuteInConstrainedRegion(toBSTR);

                // get a pointer to our new "pinned" string
                char* value = (char*)_gcHandle.AddrOfPinnedObject();
                // get a pointer to the unencrypted string
                char* charPointer = (char*)insecurePointer;
```

```
                // copy
                for (int i = 0; i < _secureString.Length; i++)
                {
                    value[i] = charPointer[i];
                }
            };
            RuntimeHelpers.CleanupCode cleanup = delegate
            {
                // insecurePointer was temporarily used to access the securestring
                // set the string to all 0's and then clean it up. this is important.
                // this prevents sniffers from seeing the sensitive info as it is cleaned up.
                if (insecurePointer != IntPtr.Zero)
                {
                    Marshal.ZeroFreeBSTR(insecurePointer);
                }
            };
            // Better than a try/catch. Not even a threadexception will bypass the
cleanup code
            RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup(code, cleanup, null);
        }
    }

#if !DEBUG
    [DebuggerHidden]
#endif
    public void Dispose()
    {
        unsafe
        {
            // we have created an insecurestring
            if (_gcHandle.IsAllocated)
            {
                // get the address of our gchandle and set all chars to 0's
                char* insecurePointer = (char*)_gcHandle.AddrOfPinnedObject();
                for (int i = 0; i < _secureString.Length; i++)
                {
                    insecurePointer[i] = (char)0;
                }
#if DEBUG
                string disposed = "¡DISPOSED¡";
                disposed = disposed.Substring(0, Math.Min(disposed.Length,
_secureString.Length));
                for (int i = 0; i < disposed.Length; ++i)
                {
                    insecurePointer[i] = disposed[i];
```

```
                }
#endif
                _gcHandle.Free();
            }
        }
    }

    public IEnumerator<char> GetEnumerator()
    {
        if (_gcHandle.IsAllocated)
        {
            return Value.GetEnumerator();
        }
        else
        {
            return new List<char>().GetEnumerator();
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    private static void ExecuteInConstrainedRegion(Action action)
    {
        RuntimeHelpers.PrepareConstrainedRegions();
        try
        {
        }
        finally
        {
            action();
        }}}}
```

Program.cs

```
using EncryptedAssemblies.Starter;
using EncryptedAssemblies.Starter.Cryptography;
using EncryptedAssemblies.TestApplication;
using System;
using System.IO;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Security;
```

```
namespace EncryptedAssemblies
{
   class Program
   {
      [DllImport("user32.dll")]
      public static extern IntPtr FindWindow(string lpClassName, string
lpWindowName);

      [DllImport("user32.dll")]
      static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

      static Program()
      {
         AppDomain.CurrentDomain.UnhandledException += (sender, eventArgs) =>
Console.WriteLine(eventArgs.ExceptionObject.ToString());
         AppDomain.CurrentDomain.AssemblyResolve +=
CurrentDomainOnAssemblyResolve;
         _password = new SecureString();
      }

      private static readonly SecureString _password;

      [STAThread]
      public static void Main(string[] args)
      {
         //Выводит фейковое сообщение об ошибке
         try
         {
            ArgumentException ex = new ArgumentException("There is not enough
data to start application");
            throw ex;
         }
         catch (ArgumentException ex)
         {
            Console.WriteLine(ex.ToString());
            Console.WriteLine("Press Esc to exit");
         }

         EncryptAssemblies();

         _password.Clear();

         if (!ReadPassword())
            return;
```

```
    if (args.Length == 0)
        RunApplication();
    else
    {
        switch (args[0].TrimStart('-', '/', '\\').ToLower())
        {
            case "help":
            case "h":
            case "?":
                Console.WriteLine("Encrypt assemblies: -ea");
                break;

            case "ea": //зашифровать сборки
                EncryptAssemblies();
                break;
        }
    }

}

private static bool ReadPassword()
{
    ConsoleKeyInfo consoleKey = Console.ReadKey(true);

    while (consoleKey.Key != ConsoleKey.Enter)
    {
        if (consoleKey.Key == ConsoleKey.Escape)
        {
            return false;
        }
        _password.AppendChar(consoleKey.KeyChar);
        consoleKey = Console.ReadKey(true);
    }
    return _password.Length > 0;
}

private static void RunApplication()
{
    SetConsoleWindowVisibility(false);

    App app = new App();
    MainWindow window = new MainWindow();
    app.Run(window);
}
```

```
        /// <summary>
        /// Зашифровывает сборки и удаляет оригинальные файлы.
        /// </summary>
        private static void EncryptAssemblies()
        {
            _password.Clear();
            GetKeyFromFile();

            Wiper wiper = new Wiper();
            foreach (string file in
Directory.GetFiles(AppDomain.CurrentDomain.BaseDirectory, "*.dll"))
            {
                byte[] source = File.ReadAllBytes(file);
                CryptedData crypted = CryptographyHelper.Encrypt(source, _password);
                string resultPath = Path.Combine(Path.GetDirectoryName(file),
Path.GetFileNameWithoutExtension(file) + ".edll");
                File.WriteAllBytes(resultPath, crypted.ToArray());
                //удаляем оригинальную сборку
                wiper.WipeFile(file, 3);
                //File.Delete(file);
            }
            string currentAssemblyName =
Assembly.GetEntryAssembly().GetName().Name;
            foreach (string file in
Directory.GetFiles(AppDomain.CurrentDomain.BaseDirectory, "*.pdb"))
            {
                if (Path.GetFileNameWithoutExtension(file) == currentAssemblyName)
                    continue;
                byte[] source = File.ReadAllBytes(file);
                CryptedData crypted = CryptographyHelper.Encrypt(source, _password);
                string resultPath = Path.Combine(Path.GetDirectoryName(file),
Path.GetFileNameWithoutExtension(file) + ".epdb");
                File.WriteAllBytes(resultPath, crypted.ToArray());
                //удаляем оригинальную сборку
                wiper.WipeFile(file, 3);
            }
        }

        /// <summary>
        /// Обработчик подгрузки сборки.
        /// </summary>
        private static Assembly CurrentDomainOnAssemblyResolve(object sender,
ResolveEventArgs args)
        {
```

```
            string[] fileParts = args.Name.Split(",".ToCharArray());

            string assemblyPath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, fileParts[0] + ".edll");
            string symbolsPath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, fileParts[0] + ".epdb");
            byte[] assemblyBytes = null, symbolsBytes = null;
            if (File.Exists(assemblyPath))
            {
                assemblyBytes = DecryptFile(assemblyPath);
            }
            if (File.Exists(symbolsPath))
            {
                symbolsBytes = DecryptFile(symbolsPath);
            }
            return Assembly.Load(assemblyBytes, symbolsBytes);
        }

        /// <summary>
        /// Расшифровывает файл.
        /// </summary>
        /// <param name="path">Путь к файлы.</param>
        /// <returns>Расшифрованные данные файла.</returns>
        private static byte[] DecryptFile(string path)
        {
            CryptedData data;
            using (FileStream fs = File.OpenRead(path))
            {
                data = CryptedData.Create(fs);
            }
            byte[] bytes = CryptographyHelper.Decrypt(data, _password);
            return bytes;
        }

        /// <summary>
        /// Устанавливает видимость окна консоли.
        /// </summary>
        /// <param name="visible">Видимость окна консоли.</param>
        private static void SetConsoleWindowVisibility(bool visible)
        {
            IntPtr hWnd = FindWindow(null, Console.Title);

            if (hWnd != IntPtr.Zero)
            {
                if (visible)
```

```
                ShowWindow(hWnd, 1); //1 = SW_SHOWNORMAL
            else
                ShowWindow(hWnd, 0); //0 = SW_HIDE
        }
    }

    public static void GetKeyFromFile()
    {
        FileStream file1 = new FileStream("D:\\test.txt", FileMode.Open); //создаем файловый поток
        StreamReader reader = new StreamReader(file1); // создаем «потоковый читатель» и связываем его с файловым потоком

        foreach (char c in reader.ReadToEnd())
        {
            _password.AppendChar(c);
        }

        reader.Close(); //закрываем поток
    }
  }
}
```

Wiper.cs

```
using System;
using System.IO;
using System.Security.Cryptography;

namespace EncryptedAssemblies.Starter
{
  internal sealed class Wiper
  {
    /// <summary>
    /// Deletes a file in a secure way by overwriting it with
    /// random garbage data n times.
    /// </summary>
    /// <param name="filename">Full path of the file to be deleted</param>
    /// <param name="timesToWrite">Specifies the number of times the file should be overwritten</param>
    public void WipeFile(string filename, int timesToWrite)
    {
        try
        {
            if (File.Exists(filename))
```

```
            {
                // Set the files attributes to normal in case it's read-only.
                File.SetAttributes(filename, FileAttributes.Normal);

                // Calculate the total number of sectors in the file.
                double sectors = Math.Ceiling(new FileInfo(filename).Length / 512.0);

                // Create a dummy-buffer the size of a sector.
                byte[] dummyBuffer = new byte[512];

                // Create a cryptographic Random Number Generator.
                // This is what I use to create the garbage data.
                using (RNGCryptoServiceProvider rng = new
RNGCryptoServiceProvider())
                {
                    // Open a FileStream to the file.
                    FileStream inputStream = new FileStream(filename, FileMode.Open);
                    for (int currentPass = 0; currentPass < timesToWrite; currentPass++)
                    {
                        UpdatePassInfo(currentPass + 1, timesToWrite);

                        // Go to the beginning of the stream
                        inputStream.Position = 0;

                        // Loop all sectors
                        for (int sectorsWritten = 0; sectorsWritten < sectors;
sectorsWritten++)
                        {
                            UpdateSectorInfo(sectorsWritten + 1, (int)sectors);

                            // Fill the dummy-buffer with random data
                            rng.GetBytes(dummyBuffer);
                            // Write it to the stream
                            inputStream.Write(dummyBuffer, 0, dummyBuffer.Length);
                        }
                    }
                    // Truncate the file to 0 bytes.
                    // This will hide the original file-length if you try to recover the file.
                    inputStream.SetLength(0);
                    // Close the stream.
                    inputStream.Close();

                    // As an extra precaution I change the dates of the file so the
                    // original dates are hidden if you try to recover the file.
                    DateTime dt = new DateTime(2037, 1, 1, 0, 0, 0);
```

```
                File.SetCreationTime(filename, dt);
                File.SetLastAccessTime(filename, dt);
                File.SetLastWriteTime(filename, dt);

                File.SetCreationTimeUtc(filename, dt);
                File.SetLastAccessTimeUtc(filename, dt);
                File.SetLastWriteTimeUtc(filename, dt);

                // Finally, delete the file
                File.Delete(filename);
            }

            WipeDone();
        }
    }
    catch (Exception e)
    {
        WipeError(e);
    }
}

# region Events
public event PassInfoEventHandler PassInfoEvent;
private void UpdatePassInfo(int currentPass, int totalPasses)
{
    PassInfoEventHandler handler = PassInfoEvent;
    if (handler != null)
    {
        handler(new PassInfoEventArgs(currentPass, totalPasses));
    }
}

public event SectorInfoEventHandler SectorInfoEvent;
private void UpdateSectorInfo(int currentSector, int totalSectors)
{
    SectorInfoEventHandler handler = SectorInfoEvent;
    if (handler != null)
    {
        handler(new SectorInfoEventArgs(currentSector, totalSectors));
    }
}

public event WipeDoneEventHandler WipeDoneEvent;
private void WipeDone()
{
```

```
        WipeDoneEventHandler handler = WipeDoneEvent;
        if (handler != null)
        {
            handler(new WipeDoneEventArgs());
        }
    }

    public event WipeErrorEventHandler WipeErrorEvent;
    private void WipeError(Exception e)
    {
        WipeErrorEventHandler handler = WipeErrorEvent;
        if (handler != null)
        {
            handler(new WipeErrorEventArgs(e));
        }
    }
    # endregion
}

# region Events
# region PassInfo
public delegate void PassInfoEventHandler(PassInfoEventArgs e);
public class PassInfoEventArgs : EventArgs
{
    private readonly int cPass;
    private readonly int tPass;

    public PassInfoEventArgs(int currentPass, int totalPasses)
    {
        cPass = currentPass;
        tPass = totalPasses;
    }

    /// <summary> Get the current pass </summary>
    public int CurrentPass { get { return cPass; } }
    /// <summary> Get the total number of passes to be run </summary>
    public int TotalPasses { get { return tPass; } }
}
# endregion

# region SectorInfo
public delegate void SectorInfoEventHandler(SectorInfoEventArgs e);
public class SectorInfoEventArgs : EventArgs
{
    private readonly int cSector;
```

```
        private readonly int tSectors;

        public SectorInfoEventArgs(int currentSector, int totalSectors)
        {
            cSector = currentSector;
            tSectors = totalSectors;
        }

        /// <summary> Get the current sector </summary>
        public int CurrentSector { get { return cSector; } }
        /// <summary> Get the total number of sectors to be run </summary>
        public int TotalSectors { get { return tSectors; } }
    }
    # endregion

    # region WipeDone
    public delegate void WipeDoneEventHandler(WipeDoneEventArgs e);
    public class WipeDoneEventArgs : EventArgs
    {
    }
    # endregion

    # region WipeError
    public delegate void WipeErrorEventHandler(WipeErrorEventArgs e);
    public class WipeErrorEventArgs : EventArgs
    {
        private readonly Exception e;

        public WipeErrorEventArgs(Exception error)
        {
            e = error;
        }

        public Exception WipeError { get { return e; } }
    }
    # endregion
    # endregion
}
```