МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация 1-40 05 01-03 Информационные системы и технологии

(издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

по дисциплине: «Защита информации и надежность информационных систем»

по теме: «Реализация и исследование блочных алгоритмов AES,Twofish и Serpent»

Выполнил студент 3 курса 1 группы Парибок Илья Александрович

(Ф.И.О., подпись)

Руководитель проекта ассистент Савельева М. Г

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 5](#_Toc134983175)

[1 Аналитический обзор литературы 6](#_Toc134983176)

[1.1 Конкурс AES 6](#_Toc134983177)

[1.2 Вывод по разделу 9](#_Toc134983178)

[2 Описание метода 10](#_Toc134983179)

[2.1 Алгоритм шифрования AES 10](#_Toc134983180)

[2.2 Алгоритм шифрования TwoFish 12](#_Toc134983181)

[2.3 Алгоритм шифрования Serpent 14](#_Toc134983182)

[2.4 Сравнение алгоритмов Serpent, AES, Twofish 16](#_Toc134983183)

[2.5 Сравнение скорости работы 17](#_Toc134983184)

[2.6 Лавинный эффект. 18](#_Toc134983185)

[2.7 Вывод по разделу 19](#_Toc134983186)

[3 Разработка программного средства 20](#_Toc134983187)

[3.2 Структура приложения 20](#_Toc134983188)

[3.3 Руководство пользователя 22](#_Toc134983189)

[3.4 Тестирование программного средства 23](#_Toc134983190)

[3.5 Вывод по разделу 25](#_Toc134983191)

[Заключение 26](#_Toc134983192)

[Список литературных источников 27](#_Toc134983193)

[Приложение А 28](#_Toc134983194)

[Приложение Б 30](#_Toc134983196)

[Приложение В 31](#_Toc134983197)

[Приложение Г 33](#_Toc134983198)

# Введение

Защита информации и надежность информационных систем являются ключевыми аспектами в современном мире, где данные и информация имеют большую ценность. Различные атаки на информационные системы и утечки данных могут привести к серьезным последствиям, таким как финансовые потери, угроза безопасности и даже угроза жизни людей. Поэтому защита информации и надежность информационных систем становятся все более важными.

Цель этого курсового проекта — реализация и исследование блочных алгоритмов AES, Twofish и Serpent. AES, Twofish и Serpent – одни из наиболее распространенных блочных шифров в мире. Каждый из них имеет свои особенности и преимущества, что делает их привлекательными для использования в различных сферах, таких как финансы, медицина, правительство и многие другие.

В рамках курсового проекта предлагается решить следующие задачи:

– изучить принципы работы шифров AES, Twofish и Serpent;

– реализовать AES, Twofish и Serpent алгоритмы;

– произвести тестирование разработанного ПО на наличие сбоев;

– сравнить производительность разработанных алгоритмов;

– оформить результаты исследования в виде отчета.

В работе будут рассмотрены основные принципы работы и реализации данных шифров, и проведены сравнение их характеристик и надежности. Благодаря этому, получится более полная картина о том, как работают данные шифры и какие преимущества и недостатки у каждого из них.

Изучение и сравнительный анализ блочных шифров AES, Twofish и Serpent позволяет получить более полное представление о том, как работают данные шифры и какие преимущества и недостатки у каждого из них. Это помогает повысить уровень знаний о защите информации и надежности информационных систем, что в свою очередь позволяет более эффективно защищать данные и информацию, а также уменьшить риск утечки данных и других серьезных последствий, связанных с нарушением безопасности информации.

1. Аналитический обзор литературы

Блочные шифры – это тип криптографических алгоритмов, которые работают с фиксированными блоками данных одинакового размера. Каждый блок данных проходит через несколько итераций (раундов), в результате которых происходит перестановка и замена битов в блоке. Такой процесс шифрования позволяет достичь высокой степени защиты передаваемых данных.

Блочные шифры широко используются в современных системах защиты информации. Они применяются для шифрования конфиденциальных данных, таких как пароли, банковские данные, медицинские записи и т.д. Благодаря своей эффективности и надежности, блочные шифры считаются одними из самых сильных криптографических алгоритмов.

В целом, блочные шифры являются критически важной частью современных систем защиты информации. Изучение и сравнительный анализ различных блочных шифров помогает определить наиболее подходящий алгоритм для конкретных целей и обеспечить максимальную степень защиты передаваемых данных.

Конкурс AES

В 1980-х годах был разработан стандарт симметричного криптоалгоритма DES (Data Encryption Standard), который получил широкое распространение в США. Однако на сегодняшний день этот стандарт не пригоден для использования по двум причинам. Во-первых, его ключ имеет длину всего 56 бит, что на современном этапе развития компьютерной техники является недостаточным для обеспечения безопасности передачи данных. Во-вторых, при разработке алгоритма было ориентировано на аппаратную реализацию, то есть на выполнение операций на микропроцессорах за слишком большое время (например, таких, как перестановка бит внутри машинного слова по определенной схеме).

Так как DES был устаревшим и небезопасным, в 1997 году Американский институт стандартизации NIST – National Institute of Standards & Technology объявил конкурс на новый стандарт симметричного криптоалгоритма. На этот раз были учтены ошибки прошлого шифра, а к разработке нового алгоритма были привлечены ведущие центры по криптографии со всего мира. Победитель конкурса получил название AES – Advanced Encryption Standard и стал де-факто мировым криптостандартом на ближайшие 10-20 лет.

Требования, предъявленные к кандидитам на AES в 1998 году, были предельно просты:

* + алгоритм должен быть симметричным,
  + алгоритм должен быть блочным шифром,
  + алгоритм должен иметь длину блока 128 бит, и поддерживать три длины ключа: 128, 192 и 256 бит.

Дополнительно кандидатам рекомендовалось:

* + использовать операции, легко реализуемые как аппаратно (в микрочипах), так и программно (на персональных компьютерах и серверах),
  + ориентироваться на 32-разрядные процессоры,
  + не усложнять без необходимости структуру шифра для того, чтобы все заинтересованные стороны были в состоянии самостоятельно провести независимый криптоанализ алгоритма и убедиться, что в нем не заложено каких-либо недокументированных возможностей.

Конкурс на новый стандарт симметричного криптоалгоритма, объявленный NIST в 1997 году, привлек многих участников со всего мира. Среди 15 заявок, полученных на первом этапе, были представлены следующие алгоритмы: 3-Way, CAST-256, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael (победитель, получивший название AES), Serpent, Twofish, SAFER+**.**

Из этих алгоритмов в финальный этап прошли CAST-256, MARS, RC6, Rijndael (AES) и Serpent.

Все пять финалистов конкурса AES – это итерационные блочные алгоритмы шифрования, которые определяют преобразование, которое повторяется определенное число раз над блоком шифруемых или дешифруемых данных. Входным блоком для шифрования является plaintext, а результатом шифрования является ciphertext. Для дешифрования используется ciphertext как входной блок данных. Каждый финалист также определяет метод создания серии ключей из исходного ключа, который называется управлением ключом. Полученные ключи называются подключами. Функции раунда используют в качестве входа различные подключи для конкретного блока данных.

У каждого финалиста первая и последняя криптографические операции являются некоторой формой перемешивания подключей и блока данных. Такие операции, используемые на начальном шаге первого раунда и заключительном шаге последнего раунда, называются pre- и post-забеливанием (whitening) и могут быть определены отдельно.

Существуют также некоторые другие технические особенности финалистов. Четыре финалиста определяют таблицы подстановки, называемые S-box: AxB-битный S-box заменяет *А* входных битов на *В* выходных битов. Три финалиста определяют функции раунда, являющиеся сетью Фейстеля. В классической сети Фейстеля одна половина блока данных используется для модификации другой половины блока данных, затем половины меняются местами. Два финалиста не используют сеть Фейстеля, в каждом раунде обрабатывают параллельно весь блок данных, применяя подстановки и линейные преобразования; таким образом, эти два финалиста являются примерами алгоритмов, использующих линейно-подстановочное преобразование.

Далее рассмотрим каждый из алгоритмов в алфавитном порядке.

MARS выполняет последовательность преобразований в следующем порядке: сложение с ключом в качестве pre-whitening, 8 раундов прямого перемешивания без использования ключа, 8 раундов прямого преобразования с использованием ключа, 8 раундов обратного преобразования с использованием ключа, 8 раундов обратного перемешивания без использования ключа и вычитание ключа в качестве post-whitening. 16 раундов с использованием ключа называются криптографическим ядром. Раунды без ключа используют два 8х16-битных S-boxes и операции сложения и XOR. В дополнение к этим элементам раунды с ключом используют 32-битное умножение ключа, зависимые от данных циклические сдвиги и добавление ключа. Как раунды перемешивания, так и раунды ядра являются раундами модифицированной сети Фейстеля, в которых четверть блока данных используется для изменения остальных трех четвертей блока данных. MARS предложен корпорацией IBM.

RC6 является параметризуемым семейством алгоритмов шифрования, основанных на сети Фейстеля; для AES было предложено использовать 20 раундов. Функция раунда в RC6 задействует переменные циклические сдвиги, которые определяются квадратичной функцией от данных. Каждый раунд также включает умножение по модулю 32, сложение, XOR и сложение с ключом. Сложение с ключом также используется для pre и post-whitening. RC6 был предложен лабораторией RSA.

Rijndael представляет собой алгоритм, использующий линейно-подстановочные преобразования и состоящий из 10, 12 или 14 раундов в зависимости от длины ключа. Блок данных, обрабатываемый с использованием Rijndael, делится на массивы байтов, и каждая операция шифрования является байт-ориентированной. Функция раунда Rijndael состоит из четырех слоев. В первом слое для каждого байта применяется S-box размерностью 8х8 бит. Второй и третий слои являются линейными перемешиваниями, в которых строки рассматриваются в качестве сдвигаемых массивов и столбцы перемешиваются. В четвертом слое выполняется операция XOR байтов подключа и каждого байта массива. В последнем раунде перемешивание столбцов опущено. Rijndael предложен Joan Daemen (Proton World International) и Vincent Rijmen (Katholieke Universiteit Leuven).

Serpent является алгоритмом, использующим линейно-подстановочные преобразования и состоящим из 32 раундов. Serpent также определяет не криптографические начальную и заключительную перестановки, которые облегчают альтернативный режим реализации, называемый bitslice. Функция раунда состоит из трех слоев: операция XOR с ключом, 32-х параллельное применение одного из восьми фиксированных S-boxes и линейное преобразование. В последнем раунде слой XOR с ключом заменен на линейное преобразование. Serpent предложен Ross Anderson (University of Cambridge), Eli Biham (Technion) и LarsKnudsen (University of California San Diego).

Twofish является сетью Фейстеля с 16 раундами. Сеть Фейстеля незначительно модифицирована с использованием однобитных ротаций. Функция раунда влияет на 32-битные слова, используя четыре зависящих от ключа S-boxes, за которыми следуют фиксированные максимально удаленные отдельные матрицы в GF (28), преобразование псевдо-Адамара и добавление ключа. Twofish был предложен Bruce Schneier, John Kelsey и Niels Ferguson (Counterpane Internet Security, Inc.), Doug Whiting (Hi/fn, Inc.), David Wagner (University of California Berkley) и Chris Hall (Princeton University).

В финальном этапе конкурса AES было уделено больше внимания сравнению качественных характеристик алгоритмов, чем в первом этапе, где в основном оценивались количественные параметры. При выборе победителя финального раунда было учтено не только соответствие алгоритма заданным критериям, но и его практическая применимость, адаптивность и устойчивость к различным атакам и угрозам безопасности. Таблица 1.1 представляет преимущества, которые определили места на пьедестале, а также количество голосов, отданных за каждый алгоритм.

Таблица 1.1 – Сравнение финалистов конкурса AES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преимущество | Алгоритмы | | | | |
| RIJNDAEL | SERPENT | TWOFISH | RC6 | MARS |
| Быстродействие:  аппаратная реализация | + | + |  |  |  |
| Программная реализация:  на слабых ВС  на мощных ВС | + | + |  | +  + |  |
| Этап расширения ключа | + |  | + |  |  |
| Распараллеливаемость | + |  |  |  |  |
| Этап дешифрирования |  |  | + | + | + |
| Запас криптостойкости | Оптимум | Завышен | Завышен | Оптимум | Завышен |
| Количество голосов | 86 | 59 | 31 | 23 | 13 |

Вывод по разделу

Rijndael был выбран в качестве нового стандарта AES, поскольку он сочетал высокую стойкость к криптоанализу и хорошую производительность, а также был наиболее простым для реализации среди финалистов. Однако, другие финалисты также продолжали использоваться в различных областях, и многие из них остаются криптографически стойкими и в настоящее время.

1. Описание метода

Алгоритм шифрования AES

Двумя специалистами по криптографии из Бельгии был разработан нетрадиционный блочный шифр AES (Rijndael), который не использует сеть Фейстеля для криптопреобразований. Шифр представляет каждый блок данных в виде двумерного массива байт размером 4х4, 4х6 или 4х8, в зависимости от длины блока, и на соответствующих этапах преобразования производятся операции над столбцами, строками или байтами таблицы.

Все преобразования в шифре основаны на строгом математическом обосновании. Структура и последовательность операций позволяют эффективно выполнять шифрование на 8-битных и 32-битных процессорах. В структуре алгоритма заложена возможность параллельного выполнения некоторых операций, что может увеличить скорость шифрования в 4 раза на многопроцессорных рабочих станциях. Алгоритм состоит из некоторого количества раундов (от 10 до 14 – это зависит от размера блока и длины ключа.

Начальная обработка. Перед шифрованием производится начальная обработка блока данных. Сначала каждый байт блока данных XOR-ится с соответствующим байтом ключа. Затем производится замена байтов (SubBytes) по заранее определенной таблице замен (S-Box). Далее производится сдвиг строк (ShiftRows) – каждая строка блока данных циклически сдвигается на некоторое количество позиций влево. Наконец, производится смешивание столбцов (MixColumns) – каждый столбец блока данных умножается на определенную матрицу.

Раунды шифрования AES использует 10 раундов шифрования для ключей длиной 128 бит. Каждый раунд состоит из четырех шагов:

* SubBytes: каждый байт блока данных заменяется на соответствующий байт из S-Box таблицы замен (рис. 2.1).
* ShiftRows: производится циклический сдвиг строк блока данных влево на некоторое количество позиций (рис. 2.2).
* MixColumns: каждый столбец блока данных умножается на определенную матрицу (рис. 2.3).
* AddRoundKey: блок данных XOR-ится с ключом раунда (рис. 2.4).

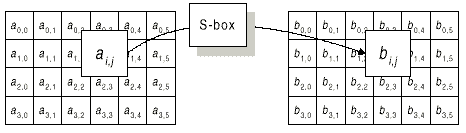


Рисунок 2.1 – Операция SubBytes

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.2 – Операция ShiftRows

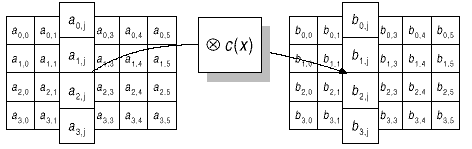


Рисунок 2.3 – Операция MixColumns

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.4 – Операция AddRoundKey

Расшифровка AES происходит обратным порядком выполнения шагов, используемых при шифровании. Это означает, что сначала производится финальная обработка блока данных, затем происходит обратный проход раундов шифрования, в которых шаги выполняются в обратном порядке, и, наконец, производится обратная начальная обработка.

Таким образом, расшифровка AES-шифра требует наличия того же ключа, что и при шифровании, и последовательного применения обратных шагов шифрования к зашифрованным данным, чтобы получить исходный текст.

Преимущества AES:

* Безопасность: AES считается одним из наиболее надежных алгоритмов шифрования. Он обеспечивает высокую стойкость к атакам посредством перебора и других методов.
* Эффективность: AES обеспечивает высокую скорость шифрования и расшифровки, что делает его идеальным для защиты крупных объемов данных.
* Гибкость: AES может быть использован с различными ключами длиной 128, 192 или 256 битов в зависимости от требований безопасности. Кроме того, он может быть реализован в различных режимах работы.
* Стандартизация: AES является стандартом шифрования, который поддерживается многими операционными системами и устройствами.

Недостатки:

* Атака c помощью анализа боковых каналов: AES, как и любой другой алгоритм шифрования, подвержен атакам посредством анализа боковых каналов, таких как электромагнитные излучения и время выполнения операций.
* Реализация: недостатки в реализации алгоритма могут привести к уязвимостям в системе шифрования.
* Квантовые вычисления: в теории, AES может быть подвержен атакам с использованием квантовых компьютеров. Однако на сегодняшний день квантовые компьютеры еще недостаточно развиты, чтобы реализовать такую атаку.

В целом, AES — это надежный и эффективный алгоритм, который широко используется во многих системах для защиты данных. Но, как и любой другой алгоритм, он не является идеальным и может иметь некоторые недостатки и уязвимости, которые необходимо учитывать при использовании.

Алгоритм шифрования TwoFish

Алгоритм разработан компанией Counterpain Security Systems, возглавляемой Брюсом Шнайером (англ. Bruce Schneier). Предыдущая программная разработка этой фирмы, называвшаяся BlowFish, являлась и до сих пор является признанным криптостойким алгоритмом.

В алгоритме TwoFish разработчики оставили некоторые удачные решения из проекта-предшественника, кроме этого произвели тщательные исследования по перемешиванию данных в сети Фейштеля. Алгоритм представляет собой сеть Фейштеля смешанного типа: первая и вторая ветвь на нечетных раундах производят модификацию третьей и четвертой, на четных раундах ситуация меняется на противоположную. В алгоритме используется криптопреобразование Адамара (англ. Pseudo-Hadamar Transform) – обратимое арифметическое сложение первого потока со вторым, а затем второго с первым. Схема работы алгоритма представлена на рисунке 2.5.

Изображение выглядит как черный, темнота

Автоматически созданное описание

Рисунок 2.5 – Схема алгоритма TwoFish

Для шифрования данных с использованием алгоритма Twofish выполняются следующие шаги:

* Подготовка ключа: ключ шифрования разбивается на несколько подключей, каждый из которых преобразуется с помощью функции расширения ключа.
* Инициализация: блок данных разбивается на 4 слова состоящих из 32 бит. Каждое слово проходит через функцию инициализации, которая изменяет его содержимое в соответствии с ключом и номером раунда.
* Раунды: алгоритм TwoFish имеет переменное количество раундов (от 16 до 128), в зависимости от длины ключа. В каждом раунде применяются следующие шаги. Добавление ключа: слова данных складываются по модулю 2 с соответствующими подключами. Линейное преобразование: слова проходят через линейную комбинацию над полем Галуа. Нелинейное преобразование: слова применяются к нелинейной функции, которая использует элементы S-блока. Перестановка: слова данных переставляются между собой.
* Финальный раунд: в последнем раунде применяются те же шаги, что и в обычных раундах, но без перестановки и нелинейного преобразования.
* Финальное преобразование: после завершения раундов, все слова проходят через финальное преобразование.

Шифрование TwoFish обратимо, то есть существует процедура расшифровки, которая противоположна шагам шифрования.

Преимущества алгоритма Twofish:

* Безопасность: Twofish считается одним из самых безопасных алгоритмов шифрования и предлагает высокий уровень защиты данных.
* Гибкость: Twofish может использоваться с ключами различной длины, что позволяет выбрать оптимальный уровень защиты для конкретной задачи.
* Быстродействие: алгоритм Twofish работает достаточно быстро даже на старых компьютерах и мобильных устройствах.
* Открытость: Twofish является открытым алгоритмом, что означает, что его исходный код доступен для ознакомления и проверки.

Недостатки алгоритма Twofish:

* Сложность: Twofish имеет более сложную структуру и требует больше вычислительных ресурсов, чем некоторые другие алгоритмы шифрования.
* Размер ключа: хотя Twofish может использоваться с ключами различной длины, длина ключа все же ограничена и не превышает 256 бит.
* Недостаточная распространенность: Twofish не так широко распространен и не поддерживается многими программными средствами, поэтому может быть неудобным выбором для определенных сценариев.

Twofish — это сильный и безопасный алгоритм симметричного блочного шифрования, который может использоваться с ключами различной длины. Он предлагает высокий уровень безопасности и гибкости, но при этом может потребовать больше вычислительных ресурсов. Twofish также является открытым алгоритмом с доступным исходным кодом, что обеспечивает дополнительную прозрачность и доверие к его безопасности.

Алгоритм шифрования Serpent

Алгоритм Serpent был разработан группой ученых из разных исследовательских центров по всему миру. Он состоит из сети Фейстеля с четырьмя ветвями смешанного типа: две из четырех ветвей меняют значения нечетных ветвей вместе, затем меняются местами. Алгоритм использует только операцию исключающего «ИЛИ», табличные подстановки и битовые сдвиги в качестве криптографических преобразований, и состоит из 32 раундов. Раунды были специально разработаны таким образом, что добавление ключевых материалов к ветвям на первом и последнем раундах формирует входное и выходное забеливание. Схема алгоритма представлена на рисунке 2.6.

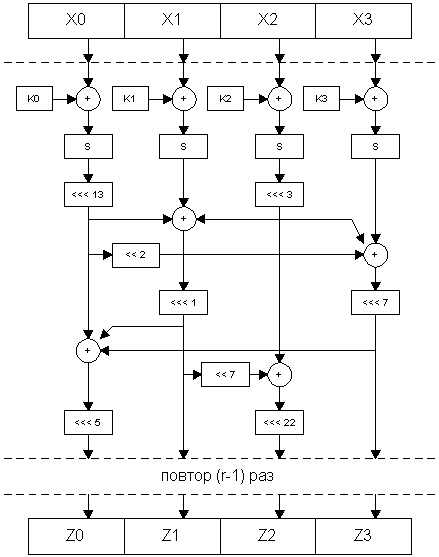


Рисунок 2.6 – Схема алгоритма Serpent

Алгоритм шифрования Serpent состоит из следующих шагов:

* Начальный этап – ключ разбивается на подключи и происходит дополнительная обработка ключа.
* Разделение блока – каждый блок данных разделяется на четыре 32-битных слова.
* Раунды подстановки – каждое слово блока проходит через заменительную таблицу (S-блок), которая заменяет исходное значение на другое значение из таблицы.
* Раунды перестановки – слова блока переставляются между собой.
* Раунд линейного преобразования – выполняется операция, которая обрабатывает каждое слово блока и изменяет его с помощью линейного преобразования.
* Объединение блока – измененные слова блока объединяются вместе.
* Конечный этап – на выходе получается зашифрованный блок данных.

Эти шаги повторяются несколько раз (обычно 32 или 64 раза), чтобы обеспечить высокую стойкость и безопасность алгоритма.

Дешифрование данных с использованием алгоритма Serpent происходит путем обратного применения операций шифрования, то есть в обратном порядке. Как и при шифровании, эти шаги повторяются в обратном порядке несколько раз (обычно 32 или 64 раза), чтобы корректно расшифровать данные и получить исходный текст.

Преимущества алгоритма Serpent:

* Serpent является одним из наиболее безопасных алгоритмов шифрования блочного типа на сегодняшний день.
* Алгоритм обладает большой силой и не подвержен известным атакам на криптографическую стойкость.
* Serpent не имеет известных уязвимостей или слабых мест, которые могут быть использованы злоумышленниками для расшифровки сообщений.

Недостатки алгоритма Serpent:

* Serpent является медленным в сравнении с некоторыми другими алгоритмами шифрования, такими как AES.
* Serpent требует большой объем памяти и вычислительных ресурсов для выполнения операций шифрования и расшифровки сообщений, что может оказаться проблемой на устройствах с ограниченными ресурсами.
* Алгоритм не так широко распространен, как AES, что может привести к низкой степени поддержки и сложностям в интеграции в различные программные и аппаратные системы.

Алгоритм шифрования Serpent является одним из наиболее стойких и безопасных алгоритмов шифрования блочного типа, который используется для защиты конфиденциальности данных. Он состоит из сетей Фейстеля и использует только простые операции XOR, замены и перестановки битов для обеспечения стойкости и безопасности. Несмотря на некоторые недостатки, Serpent является очень надежным алгоритмом шифрования и продолжает применяться в качестве криптографической защиты данных в различных областях, включая правительственные и корпоративные секторы.

Сравнение алгоритмов Serpent, AES, Twofish

AES, Twofish и Serpent – это три известных алгоритма блочного шифрования, которые используются для обеспечения безопасности данных в различных системах. Ниже представлено сравнение этих алгоритмов по ключевым параметрам.

Размер блока:

* AES использует блоки размером 128 бит.
* Twofish – 128, 192 или 256 бит.
* Serpent – 128, 192 или 256 бит.

Размер ключа:

* AES использует ключи длиной 128, 192 или 256 бит.
* Twofish – до 256 бит.
* Serpent – до 256 бит.

Безопасность:

* AES, Twofish и Serpent считаются очень безопасными алгоритмами блочного шифрования.
* В настоящее время считается, что AES обеспечивает наилучшую безопасность из трех алгоритмов, но Twofish и Serpent также считаются очень надежными.

Скорость:

* AES считается самым быстрым из трех алгоритмов. Это связано с тем, что AES использует простые операции, которые легко оптимизируются для выполнения на современных процессорах.
* Twofish и Serpent обычно медленнее, чем AES, из-за их сложной структуры и большего количества раундов.

Сравнение скорости работы

Для оценки производительности алгоритмов были проведены тестирования на различных размерах сообщений (от 512 байт до 16384 байт) с использованием средней скорости процессора.

Тестирование проводилось путем измерения времени, затраченного на шифрование и дешифрования сообщения определенным алгоритмом. Результаты измерений представлены в таблицах 2.1–2.2, где время шифрования сообщения в миллисекундах отображается для каждого алгоритма и каждого размера сообщения. На рисунках 2.7–2.8 представлены сравнительные графики скорости выполнения шифровании и дешифрования в зависимости от длины сообщения.

Таблица 2.1– Скорость выполнения шифрования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Шифр | Размер блока данных в битах | | | | | |
| 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
| AES | 0,09 | 0,12 | 0,18 | 0,32 | 0,55 | 1,8 |
| Twofish | 0,11 | 0,22 | 0,44 | 0,86 | 1,5 | 3,4 |
| Serpent | 0,13 | 0,22 | 0,43 | 0,82 | 1,62 | 3,2 |

Таблица 2.2– Скорость выполнения дешифрования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Шифр | Размер блока данных в битах | | | | | |
| 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
| AES | 0,08 | 0,09 | 0,17 | 0,22 | 0,57 | 1,6 |
| Twofish | 0,11 | 0,22 | 0,44 | 0,87 | 1,7 | 3,6 |
| Serpent | 0,1 | 0,2 | 0,4 | 0,75 | 1,3 | 2,7 |

Рисунок 2.7 – Сравнительный график скорости шифрования алгоритмов AES, Twofish, Serpent

Рисунок 2.8 – Сравнительный график скорости дешифрования алгоритмов AES, Twofish, Serpent

По представленным данным можно сделать следующие выводы:

* AES является самым быстрым алгоритмом среди трех, как для шифрования, так и для дешифрования сообщений различной длины.
* Twofish и Serpent имеют схожие показатели скорости, однако Twofish в целом немного медленнее Serpent при шифровании и быстрее при дешифровании.
* Время шифрования и дешифрования для всех алгоритмов увеличивается с увеличением длины сообщения. Это ожидаемо, поскольку больше данных требует больше времени для обработки.
* Разница во времени шифрования и дешифрования для каждого алгоритма не является существенной, что указывает на хорошую производительность алгоритмов в обеих операциях.

В зависимости от требований к безопасности и производительности, можно выбрать подходящий алгоритм из представленных. AES является наиболее быстрым и широко используемым алгоритмом, однако Twofish и Serpent тоже являются надежными и могут быть предпочтительными в некоторых случаях.

Лавинный эффект.

Лавинный эффект в криптографии означает, что даже небольшие изменения в исходных данных, таких как ключ или текст, приводят к значительным изменениям в зашифрованных данных. Другими словами, лавинный эффект описывает свойство криптографических алгоритмов, при котором каждое изменение входных данных приводит к масштабным изменениям в выходных данных. Например, если зашифровать одно и то же сообщение с помощью двух разных ключей, которые незначительно отличаются, то зашифрованные данные будут сильно различаться.

Например, если взять текст «Сезон» и зашифровать его ключами «Красный» и «Красны», то полученные результаты будут существенно отличаться друг от друга. Результаты шифрования при изменении ключа представлены в таблице 2.3.

Таблица 2.3 – Сравнение лавинного эффекта при смене ключа

|  |  |  |
| --- | --- | --- |
| Алгоритм | Ключ | |
| Красный | Красны |
| AES | xGthyAnO9xpsZPwv1A2iAA== | RDLxy7kCfyVL7Vu+7ImPYw== |
| Serpent | GK1r8UIuLXYmX8pUUef1hQ== | rI+kyWx6Zi7pne42EY7QCg== |
| Twofish | Fx8gMj0/TU5UV3uCkZOb/t5GTX8H7FEuBOsLK2ZXTTY= | ABIgMjM2TFtlcJi0vM/R1tVznIQjPfOuL+6puG4MHcI= |

Так же лавинный эффект проявляется при изменении сообщения. Результаты шифрования при изменении сообщения представлены в таблице 2.4.

Таблица 2.4 ­– Сравнение лавинного эффекта при смене сообщения

|  |  |  |
| --- | --- | --- |
| Алгоритм | Сообщение | |
| Сезон | езон |
| AES | xGthyAnO9xpsZPwv1A2iAA== | PLmoeV5cgpcIDXuOetbD7Q== |
| Serpent | GK1r8UIuLXYmX8pUUef1hQ== | A9nZOaGOgHiTp81mqeOGkw== |
| wofish | Fx8gMj0/TU5UV3uCkZOb/t5GTX  8H7FEuBOsLK2ZXTTY= | Fx8gMj0/TU5UV3uCkZOb/pXhIDxcPdImLE46QmDt4UM= |

Зашифрованные сообщения, полученные при использовании криптографических алгоритмов, отличаются друг от друга, даже если используются похожие ключи. Этот эффект называется лавинным и он является важным фактором для обеспечения безопасности криптографических систем. Лавинный эффект в условиях работы с конфиденциальной информацией служит признаком защиты от атак, выполненных путем перебора ключей. В случае, если лавинный эффект недостаточно силен, криптографическая система может быть подвержена таким атакам, как например, атакам методом грубой силы или дифференциального криптоанализа.

Таким образом, лавинный эффект является важным фактором в криптографии и используется для обеспечения безопасности данных и защиты от различных видов атак. При разработке криптографических систем следует учитывать множество факторов, в том числе и лавинный эффект, для обеспечения высокой надежности и безопасности работы с конфиденциальной информацией.

Вывод по разделу

В целом, AES, Twofish и Serpent все являются безопасными алгоритмами блочного шифрования, которые могут использоваться для защиты данных в различных системах. Выбор между ними может зависеть от конкретных требований к безопасности и скорости в каждом конкретном случае.

1. Разработка программного средства

## Возможности программного средства

Для реализации алгоритмов шифрования Aes, Twofish и Serpent была использована технология WPF с шаблоном проектирования MVVM. Данный подход позволил разделить приложение на три основных компонента: модель представления (ViewModel), представление (View) и модель данных (Model), что обеспечило более гибкую и удобную разработку.

Для работы с WPF будет использоваться объектно-ориентированный язык программирования с С-подобным синтаксисом – С#.

Функционально приложение поддерживает следующие операции:

* Выбор алгоритма;
* Ввод ключа;
* Ввод сообщения с клавиатуры;
* Чтение сообщения из файла;
* Зашифрование и расшифрование использование выбранного алгоритма;
* Сохранение зашифрованного сообщения в файл;
* Очистка полей;
* Вывод информации о времени работы алгоритма;

Сначала вводятся все данные, необходимые для работы приложения, а далее происходит вызов функций для шифрования или дешифрование с параметрами, заданными при запуске, после чего сообщение о результате работы программы.

Visual Studio 2023 – это интегрированная среда разработки, которая предлагает множество шаблонов, в том числе для создания клиентского приложения Windows Presentation Foundation и будет использована при разработке. Для программирования основных методов и обработки событий используется язык C#, а для разработки пользовательского интерфейса – расширяемый язык разметки XAML.

Структура приложения

Для реализации алгоритмов была использована технология WPF с шаблоном проектирования MVVM. Данный подход позволил разделить приложение на три основных компонента: модель представления (ViewModel), представление (View) и модель данных, что обеспечило более гибкую и удобную разработку.

В частности, использование технологии WPF позволило упростить разработку графического интерфейса пользователя и обеспечить его более простым и интуитивно понятным интерфейсом. Также была использована технология MVVM, которая позволила улучшить архитектуру приложения и разделить его на три основных компонента: модель представления (ViewModel), представление (View) и модель данных. Это обеспечило более гибкую и удобную разработку приложения, а также более удобную поддержку и сопровождение его в будущем.

Структура приложения представлена в рисунке 3.1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.1 – Структура приложения

В Assets содержится различные ресурсы, необходимые для работы программного обеспечения.

В Model были реализованы методы для шифрования и дешифрования данных с использованием алгоритмов Aes, Twofish и Serpent. Для каждого алгоритма был создан объект класса, реализующего данный алгоритм, и использовался для шифрования и дешифрования данных. Представления алгоритмов изложены в приложение А, Б и B.

В ViewModel были реализованы команды, которые вызывают соответствующие методы для шифрования и дешифрования данных. В приложении Г имеется реализация класса AESViewModel.cs, а остальные классы в приложении повторяют его структуру и являются аналогами этого класса.

В View были созданы элементы управления для ввода и вывода данных, а также кнопки для запуска шифрования и дешифрования. Для связывания элементов управления с соответствующими свойствами и командами в ViewModel были использованы привязки данных. Для визуализации данных использовались стандартные элементы WPF, такие как текстовые поля и кнопки.

Кроме того, в проекте была использованы библиотеки Rebex.Security.Cryptography и Org.BouncyCastle.Crypto которая содержит реализацию различных алгоритмов шифрования. Это позволило использовать высокоуровневые методы шифрования и дешифрования данных, а также гарантировать безопасность приложения.

Таким образом, проект по реализации алгоритмов шифрования Aes, Twofish и Serpent с использованием C#, WPF и MVVM представляет собой пример безопасного приложения, которое может быть использовано для защиты данных пользователей. Он обладает гибкой архитектурой, простым пользовательским интерфейсом и быстрой производительностью.

Руководство пользователя

После того, как пользователь запускает приложение, он перенаправляется на главную страницу, которая открывается автоматически. На этой странице пользователь может выбрать любой из доступных шифров, используя кнопки.

После выбора шифра пользователь может ввести ключ и сообщение, которое необходимо зашифровать или расшифровать, в соответствующие текстовые поля. Если пользователь хочет использовать заранее подготовленное сообщение, то он может загрузить его из файла.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.2 – Кнопки для выбора алгоритма



Рисунок 3.3 – Поля ввода ключа и сообщения

Для выполнения шифрования, пользователю необходимо обратится к нижней панели с кнопками, и выбрать «Зашифровать». Для расшифровки пользователю необходимо вставить шифр в текстовое поле «Зашифрование» и нажать на кнопку «Расшифровать». Результат операции будет выведен в поле текстовое поле «Сообщение». Так же в панели есть кнопки для чтения сообщения из файла, записи шифра в файл и очисти полей «Сообщение» и «Зашифрование».



Рисунок 3.4 – Панель функциональных кнопок

Результат скорости шифрование и дешифрования, а также длина сообщения выводятся в правом углу экрана.

Изображение выглядит как текст

Автоматически созданное описание

Рис 3.5 – Информация о времени выполнения и длине сообщения

Тестирование программного средства

Для проверки корректной работы программы необходимо провести ее тестирование. В процессе тестирования следует убедиться, что исходное сообщение может быть успешно зашифровано и дешифровано. В качестве тестовых данных взято сообщение «Сообщение123» и ключ «Ключ». Результат выполнения шифрования представлен на рисунках 3.6–3.8, и результат выполнения расшифрования должен быть аналогичным.

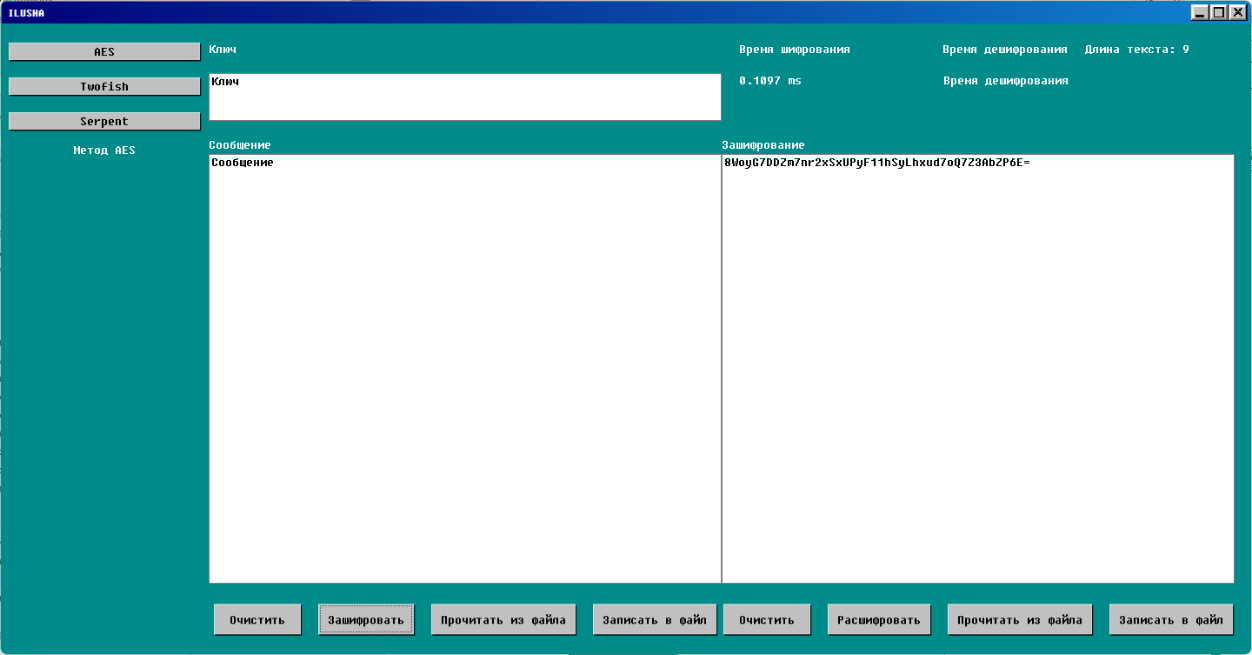


Рисунок 3.6 – Шифрование сообщения с использованием AES

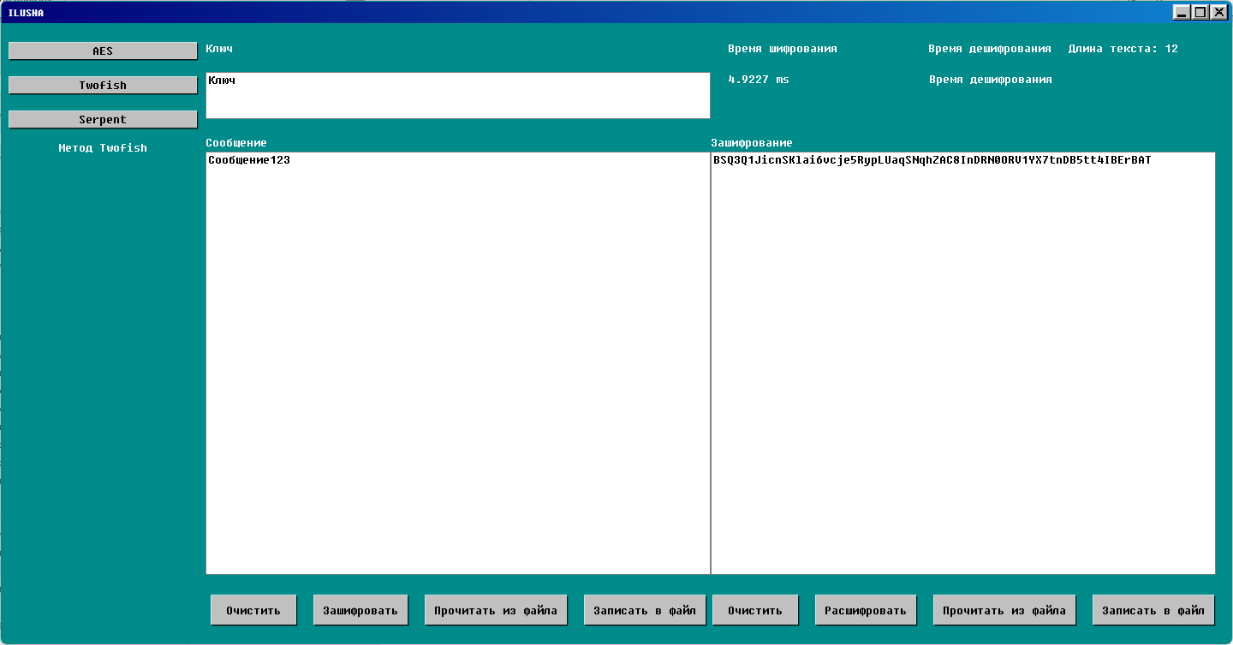


Рисунок 3.7 – Шифрование сообщения с использованием Twofish

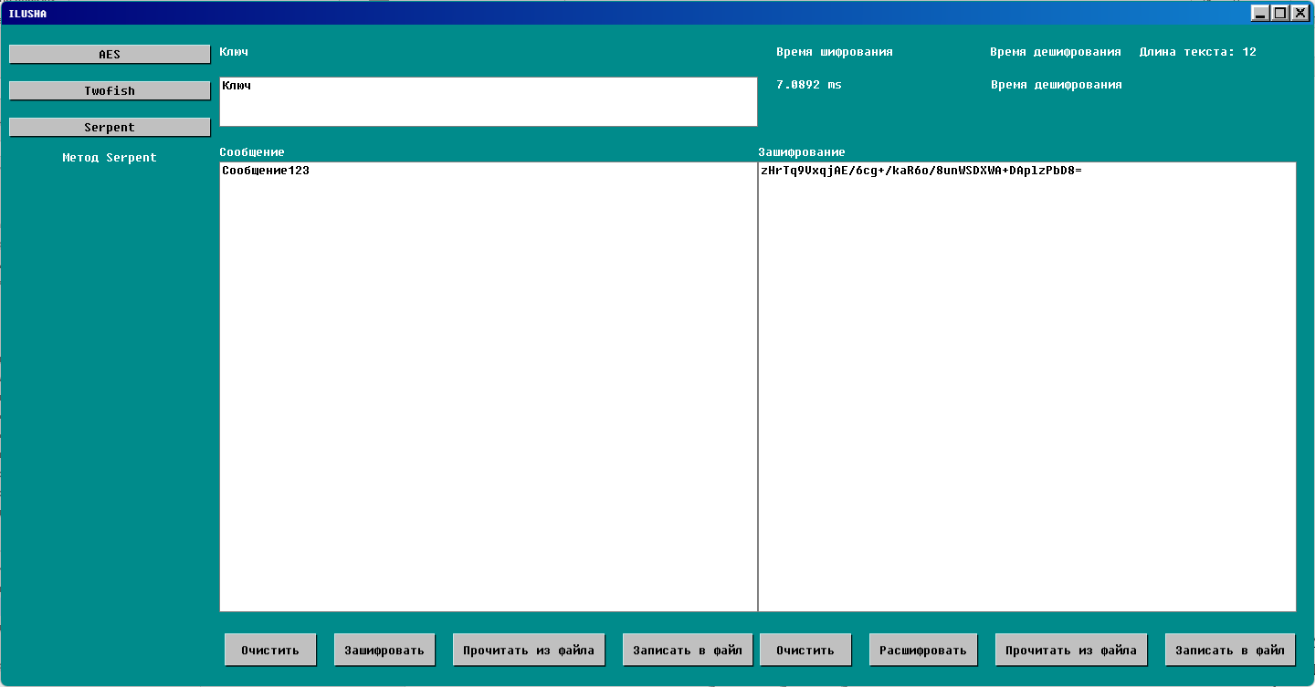


Рисунок 3.8 – Шифрование сообщения с использованием Serpent

Приложение обладает функцией обнаружения ошибок и вывода информации об этих ошибках. Например, если пользователь вводит некорректный шифротекст, то приложение сообщит об ошибке, как показано на рисунке 3.9. Такая информация об ошибке предоставляется благодаря реализации алгоритма, что позволяет обработать больше ошибок и предоставить более полную и точную информацию, чем простая проверка вводимых пользователем данных.

. Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.9 – Сообщение об ошибке

Для того чтобы предотвратить открытие пользователем файлов неподходящего формата, в окне выбора файлов был установлен фильтр, позволяющий открывать только файлы с текстовым форматом, как показано на рисунке 3.10.

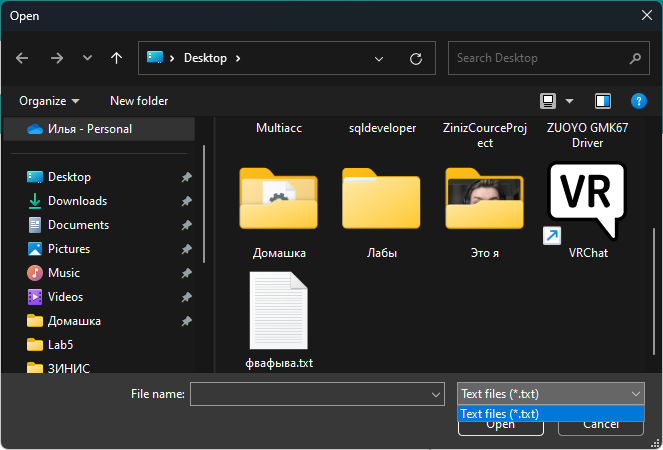


Рисунок 3.10 – Окно с выбором файла

**Вывод по разделу**

В данном разделе было проведено комплексное тестирование программного обеспечения, которое включало проверку работы главных страниц, а также методов шифрования и расшифрования. Руководство пользователя было написано с упором на простоту использования приложения, а также подробным описанием каждой функции. Все действия интуитивно понятны, что делает приложение доступным для широкого круга пользователей.

Заключение

В данной курсовой работе были реализованы три алгоритма шифрования: Aes, Twofish и Serpent. Реализация была выполнена с использованием технологии WPF с шаблоном проектирования MVVM на языке программирования C#.

Для реализации алгоритмов шифрования была использована библиотека System.Security.Cryptography, которая содержит реализацию различных алгоритмов шифрования. Библиотека обладает высокой степенью безопасности, что является важным фактором при работе с конфиденциальной информацией.

Кроме этого, были использованы дополнительные библиотеки Rebex.Security.Cryptography и Org.BouncyCastle.Crypto для реализации некоторых дополнительных алгоритмов шифрования и поддержки дополнительных функций.

Использование этих библиотек позволяет расширить возможности приложения и обеспечить дополнительный уровень безопасности. Однако, перед использованием этих библиотек, необходимо убедиться в их соответствии требованиям безопасности и правильности их использования в конкретной ситуации.

Технология WPF была выбрана из-за ее возможностей создания графических интерфейсов пользователя, а шаблон проектирования MVVM из-за его гибкости и удобства разработки. MVVM разделяет приложение на три основных компонента: модель представления (ViewModel), представление (View) и модель данных, что обеспечивает более гибкую и удобную разработку.

Реализация алгоритмов шифрования является важным шагом в создании безопасных приложений, которые могут защитить данные пользователей. В курсовой работе было продемонстрировано, как реализовать алгоритмы шифрования с использованием C#, WPF и MVVM. Созданное приложение обладает гибкой архитектурой, простым пользовательским интерфейсом и быстрой производительностью.

Таким образом, данная курсовая работа является важным шагом в изучении безопасности приложений и демонстрирует, как использовать различные средства и технологии для создания безопасных приложений, которые защищают данные пользователей.

Список литературных источников

1 Урбанович, П. П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 2. Криптографические и стеганографические методы защиты информации / П. П. Урбанович, Н. П. Шутько. – Минск: БГТУ, 2020. – 40 с.

2 Joan Daemen, Vincent Rijmen. The Design of Rijndael: AES – The Advanced Encryption Standard / Joan Daemen, Vincent Rijmen. – Springer, 2002. – 220с.

3 Al-Maathidi, M. A., Al-Rashed, R. A., & Al-Saif, A. A. (2015). A Comparative Study of Block Ciphers: AES, DES, 3DES and Blowfish. Journal of Applied Sciences, 15(1), 137-149с.

4 Schneier, B. (1998). Twofish: A 128-Bit Block Cipher [Электронный ресурс].

Режим доступа: https://www.schneier.com/twofish.html. Дата доступа: 23.04.2023

5 Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson. The Twofish Encryption Algorithm / Joan Daemen, Vincent Rijmen. The Design of Rijndael – Springer, 2002. – 208с.

Приложение А

AesEncryptor.cs

public class AesEncryptor

{

public static string EncryptionTime { get; private set; }

public static string DecryptionTime { get; private set; }

public static string Encrypt(string plainText, string key)

{

byte[] keyBytes = GetHash(key);

byte[] iv = (byte[])keyBytes.Clone();

Array.Sort(iv);

byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);

Stopwatch stopwatch = Stopwatch.StartNew();

IBufferedCipher cipher = CipherUtilities.GetCipher("AES/CBC/PKCS7Padding");

KeyParameter keyParam = new KeyParameter(keyBytes);

ParametersWithIV parameters = new ParametersWithIV(keyParam, iv);

cipher.Init(true, parameters);

byte[] encryptedBytes = cipher.DoFinal(plainTextBytes);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

EncryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Convert.ToBase64String(encryptedBytes);

}

public static string Decrypt(string cipherText, string key)

{

byte[] encryptedBytes = Convert.FromBase64String(cipherText);

byte[] keyBytes = GetHash(key);

byte[] iv = (byte[])keyBytes.Clone();

Array.Sort(iv);

Stopwatch stopwatch = Stopwatch.StartNew();

IBufferedCipher cipher = CipherUtilities.GetCipher("AES/CBC/PKCS7Padding");

KeyParameter keyParam = new KeyParameter(keyBytes);

ParametersWithIV parameters = new ParametersWithIV(keyParam, iv);

cipher.Init(false, parameters);

byte[] decryptedBytes = cipher.DoFinal(encryptedBytes);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

DecryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Encoding.UTF8.GetString(decryptedBytes);

}

public static string GetEncriptionTime()

{

return EncryptionTime;

}

public static string GetDecriptionTime()

{

return DecryptionTime;

}

public static byte[] GetHash(string input)

{

using (MD5 md5Hash = MD5.Create())

{

// Преобразуем входную строку в массив байтов

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

// Вычисляем хэш-код для входных байтов

byte[] hashBytes = md5Hash.ComputeHash(inputBytes);

return hashBytes;

}

}

}

# Приложение Б

**SerpnetEncryptor.cs**

internal class SerpnetEncryptor

{

public static string EncryptionTime { get; private set; }

public static string DecryptionTime { get; private set; }

public static string GetEncriptionTime()

{

return EncryptionTime;

}

public static string GetDecriptionTime()

{

return DecryptionTime;

}

public static string Encrypt(string inputText, string key)

{

byte[] inputBytes = Encoding.UTF8.GetBytes(inputText);

byte[] keyBytes = AesEncryptor.GetHash(key);

byte[] ivBytes = (byte[])keyBytes.Clone();

Array.Sort(ivBytes);

Stopwatch stopwatch = Stopwatch.StartNew(); // запускаем таймер

IBufferedCipher cipher = CipherUtilities.GetCipher("Serpent/CBC/PKCS7Padding");

cipher.Init(true, new ParametersWithIV(new KeyParameter(keyBytes), ivBytes));

byte[] encryptedBytes = cipher.DoFinal(inputBytes);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

EncryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Convert.ToBase64String(encryptedBytes);

}

public static string Decrypt(string encryptedText, string key)

{

byte[] encryptedBytes = Convert.FromBase64String(encryptedText);

byte[] keyBytes = AesEncryptor.GetHash(key);

byte[] ivBytes = (byte[])keyBytes.Clone();

Array.Sort(ivBytes);

Stopwatch stopwatch = Stopwatch.StartNew(); // запускаем таймер

IBufferedCipher cipher = CipherUtilities.GetCipher("Serpent/CBC/PKCS7Padding");

cipher.Init(false, new ParametersWithIV(new KeyParameter(keyBytes), ivBytes));

byte[] decryptedBytes = cipher.DoFinal(encryptedBytes);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

DecryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Encoding.UTF8.GetString(decryptedBytes); }}

# Приложение В

**TwoFishEncryptor.cs**

internal class TwoFishEncryptor

{

public static string EncryptionTime { get; private set; }

public static string DecryptionTime { get; private set; }

public static string GetEncriptionTime()

{

return EncryptionTime;

}

public static string GetDecriptionTime()

{

return DecryptionTime;

}

public static string EncryptTwofish(string plaintext, string MsgKey)

{

byte[] plaintextBytes = Encoding.UTF8.GetBytes(plaintext);

byte[] key = AesEncryptor.GetHash(MsgKey);

byte[] iv = (byte[])key.Clone();

Array.Sort(iv);

using (TwofishManaged twofish = new TwofishManaged())

{

twofish.Key = key;

twofish.IV= iv;

using (var encryptor = twofish.CreateEncryptor())

{

Stopwatch stopwatch = Stopwatch.StartNew(); // запускаем таймер

byte[] cipherText = encryptor.TransformFinalBlock(plaintextBytes, 0, plaintextBytes.Length);

byte[] result = new byte[iv.Length + cipherText.Length];

Buffer.BlockCopy(iv, 0, result, 0, iv.Length);

Buffer.BlockCopy(cipherText, 0, result, iv.Length, cipherText.Length);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

EncryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Convert.ToBase64String(result);

}

}

}

public static string DecryptTwofish(string cipherText, string MsgKey)

{

byte[] cipherTextBytes = Convert.FromBase64String(cipherText);

byte[] key = AesEncryptor.GetHash(MsgKey);

byte[] iv = (byte[])key.Clone();

Array.Sort(iv);

using (TwofishManaged twofish = new TwofishManaged())

{

twofish.Key = key;

twofish.IV = iv;

Buffer.BlockCopy(cipherTextBytes, 0, iv, 0, iv.Length);

using (var decryptor = twofish.CreateDecryptor())

{

Stopwatch stopwatch = Stopwatch.StartNew(); // запускаем таймер

byte[] plainTextBytes = decryptor.TransformFinalBlock(cipherTextBytes, iv.Length, cipherTextBytes.Length - iv.Length);

stopwatch.Stop(); // останавливаем таймер

TimeSpan ts = stopwatch.Elapsed; // получаем затраченное время

DecryptionTime = $"{ts.TotalMilliseconds} ms"; // устанавливаем значение свойства, связанного с текстом времени выполнения шифрования

return Encoding.UTF8.GetString(plainTextBytes);

}

}

}

# Приложение Г

**AESViewModel.cs**

internal class AESViewModel:INotifyPropertyChanged

{

private string \_plainText = "Сообщение";

private string \_cipherText = "Шифр Текст";

private string \_key = "Ключ";

private readonly byte[] IV = new byte[16]; // 16 байт = 128 бит

public event PropertyChangedEventHandler PropertyChanged;

public string PlainText

{

get { return \_plainText; }

set

{

\_plainText = value;

OnPropertyChanged(nameof(PlainText));

OnPropertyChanged(nameof(TextSize));

}

}

public string CipherText

{

get { return \_cipherText; }

set

{

\_cipherText = value;

OnPropertyChanged(nameof(CipherText));

}

}

public string Key

{

get { return \_key; }

set

{

\_key = value;

OnPropertyChanged(nameof(Key));

}

}

public string TextSize

{

get { return "Длина сообщения: " + CipherText.Length; }

}

private string encryptionTime = "Время шифрования";

public string EncryptionTime

{

get => encryptionTime;

set

{

encryptionTime = value;

OnPropertyChanged(nameof(EncryptionTime));

}

}

private string decryptionTime = "Время дешифрования";

public string DecryptionTime

{

get => decryptionTime;

set

{

decryptionTime = value;

OnPropertyChanged(nameof(DecryptionTime));

}

}

public ICommand EncryptCommand { get; }

public ICommand DecryptCommand { get; }

public AESViewModel()

{

EncryptCommand = new RelayCommand(Encrypt);

DecryptCommand = new RelayCommand(Decrypt);

using (var rng = new System.Security.Cryptography.RNGCryptoServiceProvider())

{

rng.GetBytes(IV);

}

}

private void Encrypt()

{

try

{

CipherText = AesEncryptor.Encrypt(PlainText, Key);

EncryptionTime = AesEncryptor.GetEncriptionTime();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

// Handle exception

}

}

private void Decrypt()

{

try

{

PlainText = AesEncryptor.Decrypt(CipherText, Key);

DecryptionTime = AesEncryptor.GetDecriptionTime();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

// Handle exception

}

}

private void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

public ICommand SaveToFileCommand => new RelayCommand(SaveToFile);

private void SaveToFile()

{

var dialog = new SaveFileDialog();

dialog.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (dialog.ShowDialog() == true)

{

using (var writer = new StreamWriter(dialog.FileName))

{

writer.Write(this.CipherText);

}

}

}

public ICommand OpenFileCommand => new RelayCommand(() =>

{

var dialog = new OpenFileDialog();

dialog.Filter = "Text files (\*.txt)|\*.txt";

if (dialog.ShowDialog() == true)

{

var text = File.ReadAllText(dialog.FileName);

PlainText = text;

}

});

public ICommand ClearMsgCommand => new RelayCommand(() =>

{

PlainText = "";

});

public ICommand ClearCipherCommand => new RelayCommand(() =>

{

CipherText = "";});}