МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 Программное обеспечение информационной безопасности мобильных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Защита информации и надежность информационных систем»

Тема: «Сравнительный анализ производительности и безопасности симметричных алгоритмов шифрования Three Fish и SM4»

Исполнитель

Студент 4 курса 7 группы Воликов Д. А.

(Ф.И.О.)

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

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 4](#_Toc185053470)

[1 Постановка задачи 5](#_Toc185053471)

[1.1 Вывод 5](#_Toc185053472)

[2 Описание методов 7](#_Toc185053473)

[2.1 Алгоритм симметричного шифрования Threefish 7](#_Toc185053474)

[2.2 Алгоритм симметричного шифрования SM4 10](#_Toc185053475)

[2.3 Вывод 12](#_Toc185053476)

[3 Описание программного средства 13](#_Toc185053477)

[3.1 Вывод 17](#_Toc185053478)

[4 Тестирование программного средства 18](#_Toc185053479)

[4.1 Вывод 22](#_Toc185053480)

[5 Сравнительный анализ методов 23](#_Toc185053481)

[5.1 Вывод 27](#_Toc185053482)

[Заключение 28](#_Toc185053483)

[Список используемых источников 29](#_Toc185053484)

[ПРИЛОЖЕНИЕ А 30](#_Toc185053485)

[ПРИЛОЖЕНИЕ Б 31](#_Toc185053486)

[ПРИЛОЖЕНИЕ В 32](#_Toc185053487)

[ПРИЛОЖЕНИЕ Г 39](#_Toc185053488)

[ПРИЛОЖЕНИЕ Д 55](#_Toc185053489)

# Введение

В современном цифровом мире защита данных является первостепенной задачей. Утечки информации могут привести к серьезным последствиям, поэтому криптография, и, в частности, симметричное шифрование, играет ключевую роль в обеспечении безопасности. Симметричное шифрование использует один и тот же ключ для шифрования и расшифрования, что обеспечивает высокую скорость обработки данных и широкое применение в различных областях, от защиты файлов до сетевых коммуникаций.

Данная работа посвящена сравнительному анализу двух известных симметричных блочных шифров: Threefish и SM4. Threefish, разработанный группой криптографов под руководством Брюса Шнайера, отличается высокой скоростью и относительно простой структурой. SM4, китайский национальный стандарт, делает акцент на криптографическую стойкость, используя сложные математические преобразования.

Целью работы является создание проекта, реализующего оба алгоритма — Threefish и SM4. Это позволит провести сравнительный анализ их ключевых характеристик: скорости зашифрования и расшифрования.

# Постановка задачи

В рамках курсового проекта необходимо провести сравнительный анализ производительности и безопасности симметричных алгоритмов шифрования Threefish и SM4. Для достижения этой цели следует рассмотреть ряд задач, представленных ниже. Основное внимание будет уделено различиям в скорости шифрования и расшифрования, а также уровням безопасности, которые обеспечивают оба алгоритма. Это позволит понять сильные и слабые стороны каждого из алгоритмов, а также определить их роль в различных сферах жизни.

Требования к средствам разработки включают использование языка *C*# и платформы Avalonia UI для создания кроссплатформенных настольных приложений. *C*# – это мощный и типобезопасный язык, входящий в экосистему .NET, который предлагает широкий спектр возможностей для разработки приложений разного уровня сложности. Avalonia UI позволяет создавать современные и отзывчивые пользовательские интерфейсы, которые могут работать на различных операционных системах, таких как Windows, macOS и Linux. В качестве среды разработки предполагается использование Microsoft Visual Studio, которая предоставляет интегрированную среду с мощными инструментами для написания, отладки и профилирования кода на *C*#.

Разрабатываемое приложение должно обеспечивать функциональность шифрования и расшифрования текстовых сообщений. Эта возможность является ключевой для любого современного инструмента, направленного на обеспечение цифровой безопасности. Применение C# и Avalonia UI позволит создать надежное и кроссплатформенное приложение, соответствующее современным требованиям к безопасности и удобству использования.

В разработке приложения использовалась библиотека Bouncy Castle, которая предоставляет мощные криптографические функции и алгоритмы. Эта библиотека поддерживает широкий спектр стандартов и протоколов, включая шифрование, расшифрование, создание и проверку цифровых подписей. Bouncy Castle обеспечивает высокую степень безопасности и гибкости, что позволяет разработчикам легко интегрировать криптографические механизмы в свои приложения. Благодаря этой библиотеке, реализованные функции шифрования и расшифрования текстовых сообщений становятся надежными и эффективными, соответствуя современным требованиям безопасности.

Этот проект предоставляет практическое применение алгоритмов Threefish и SM4, а также оценивает их работу по критерию скорости шифрования и расшифрования. Это позволит пользователю определить какой из двух алгоритмов более подходит под его задачи, а также под его данные, с которым он работает.

* 1. Вывод

В данном курсовом проекте был выполнен сравнительный анализ производительности симметричных алгоритмов шифрования Threefish и SM4 с использованием языка *C*# и фреймворка Avalonia UI в среде MS Visual Studio. Созданное приложение позволяет шифровать и расшифровывать различные файлы разного размера, а также фиксирует время, затраченное на эти процессы, что позволяет оценить эффективность алгоритмов. Интуитивно понятный интерфейс упрощает ввод данных и выбор алгоритма. Результаты тестирования показывают различия в производительности, что позволяет делать выводы о предпочтительности каждого алгоритма в зависимости от задач конечного пользователя, а также от информационной системы, в которой выполняются поставленные задачи.

# Описание методов

* 1. Алгоритм симметричного шифрования Threefish

Threefish — в криптографии симметричный блочный криптоалгоритм, разработанный группой специалистов во главе с автором Blowfish и Twofish, американским криптографом Брюсом Шнайером в 2008 году для использования в хеш-функции Skein и в качестве универсальной замены существующим блочным шифрам. Основными принципами разработки шифра были: минимальное использование памяти, необходимая для использования в хеш-функции устойчивость к атакам, простота реализации и оптимизация под 64-разрядные процессоры.

Threefish имеет очень простую структуру и может быть использован для замены алгоритмов блочного шифрования, будучи быстрым и гибким шифром, работающим в произвольном режиме шифрования. Threefish *S*-блоки не использует, основан на комбинации инструкций исключающего или, сложения и циклического сдвига. Шифр реализован в виде подстановочно-перестановочной сети на обратимых операциях, не являясь шифром сети Фейстеля.

Алгоритм предусматривает использование tweak-значения, своего рода вектора инициализации, позволяя изменять таким образом значение выхода, без изменения ключа, что имеет положительный эффект как для реализации новых режимов шифрования, так и на криптостойкости алгоритма.

Как результат мнения авторов, что несколько сложных раундов зачастую хуже применения большого числа простых раундов, алгоритм имеет нетрадиционно большое число раундов — 72 или 80 при ключе 1024 бит, однако, по заявлению создателей, его скоростные характеристики опережают AES примерно в два раза. Стоит заметить, что по причине 64-битной структуры шифра, данное заявление имеет место лишь на 64-разрядной архитектуре. Поэтому Threefish, как и Skein, основанный на нём, на 32-разрядных процессорах показывает значительно худшие результаты, чем на «родном» оборудовании.

Ядром шифра является простая функция MIX, преобразующая два 64-битных беззнаковых числа, в процессе которой происходит сложение, циклический сдвиг и сложение по модулю 2 (XOR).

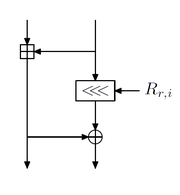


Рисунок 2.1 – Схема функции MIX

Нелинейная функция смешивания и перестановки MIXd,j принимает на вход два аргумента (*x*0, *x*1) и возвращает (*y*0, *y*1):

*y*0 = (*x*0 + *x*1) mod 264, (1.1)

*y*1 = (*x*1 <<< *R* (d mod 8), j) ⊕ *y*0, (1.2)

где «<<<» – оператор побитового циклического сдвига влево, а константа *R*d,j определяется из таблиц, представленной на рисунке ниже:

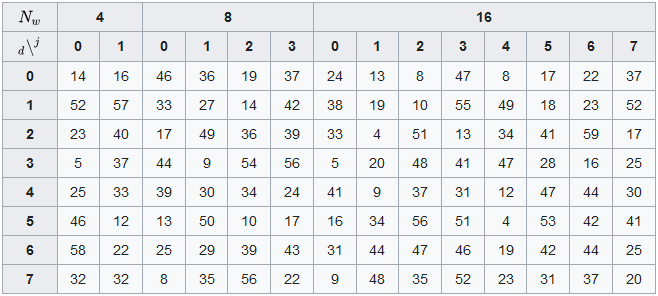


Рисунок 2.2 – Таблица отношения константы *R*d, j к параметрам алгоритма

Размер блока данных, с которым работает алгоритм — 256, 512 или 1024 бит. Длина ключа равна выбранному размеру блока. Размер tweak-значения для любого из размеров блока — 128 бит.

Определим функцию шифрования *E* (*K*, *T*, *P*), где:

* параметр *K* – ключ шифрования, строка длиной 32, 64, или 128 байт (256, 512, или 1024 бит);
* параметр *T* – tweak-значение, строка длиной 16 байт (128 бит);
* параметр *P* – открытый текст для шифрования.

Для обработки блока данные представляются в виде массива 64-битных слов (целых чисел от 0 до 264 − 1). Определим параметр *N*w как число 64-битных слов в ключе (и в блоке), тогда.

* ключ шифрования *K* = {*k*0, *k*1, …, *k*Nw-1};
* tweak-значение *T* = {*t*0, *t*1};
* открытый текст *P* = {*p*0, *p*1, …, *p*Nw-1}.

Количество раундов, *N*r для алгоритма Threefish определяется по таблице, представленной на рисунке ниже:



Рисунок 2.3 – Таблица отношений параметров *N*w и *N*r к длине ключа

Алгоритм использует *N*r / 4 + 1 раундовых ключей, а основной ключ и  
tweak-значение дополняются двумя 64-битными словами:

*t*2 = *t*0 ⊕ *t*1, (1.3)

*k*Nw = *C*240 ⊕ *k*0 ⊕ *k*1 ⊕ … ⊕ *k*Nw-1, (1.4)

где *C*240 = 0*x*1bd11bdaa9fc1a22.

Далее определяется подключ *S* = {*k*s,0, *k*s,1, …, *k*s,Nw-1}, где *S* = 0, 1, …, Nr / 4. Таким образом, соотношение элементов подключа *S* исходит из следующей системы, которая представлена ниже:

(1.5)

При зашифровании инициализируют внутренне состояние алгоритма массивом элементов *V*d,i, где *i* = {0, 1, …, *N*w-1}, а *d* = {0, 1, …, Nr-1}. Начальное внутреннее состояние определяется массивом *V*0,i = {*p*0, *p*1, …, *N*w-1}.

Каждый раунд состоит из нескольких этапов. На первом этапе раунда к внутреннему состоянию добавляется раундовый ключ *k*s,i как указанно на формулах ниже:

*e*d,i = (*V*d,i + *k*d/4,i) mod 264, (1.6)

если *d* mod 4 = 0.

*e*d,i = *V*d,i, (1.7)

если *d* mod 4 ≠ 0.

На следующем этапе раунда используется нелинейная функция MIXd,j, описанная выше:

(*f*d,2j, *f*d,2j+1) = MIXd,j(*e*d,2j, *e*d,2j+1), (1.8)

где *j* = {0, 1, …, *N*w / 2 - 1}. Таким образом, следующее внутреннее состояние определяется по следующей формуле:

*V*d+1,i = *f*d,p(i), (1.9)

где *i* = {0, 1, …, Nw-1}.

После выполнения всех раундов выходом алгоритма является шифротекст *с*i:

*c*i = (*V*Nr,i + *k*Nr/4,i) mod 264,

где *i* = {0, 1, …, *N*w-1}.

Процедура расшифрования обратна процедуре шифрования. Раундовые ключи используются в обратном порядке, и каждый раунд состоит из обратных операций. Вместо функции MIXd,j используется функция deMIXd,j, которая выполняет операции вычитания по модулю 264 и побитовый циклический сдвиг вправо. Этапы каждого раунда расшифрования также выполняются в обратном порядке.

* 1. Алгоритм симметричного шифрования SM4

SM4 — алгоритм блочного шифрования используемый в Китае как национальный стандарт для беспроводных локальных сетей (WLAN Authentication and Privacy Infrastructure (WAPI)).

SM4 был предложен как шифр используемый в стандарте IEEE 802.11i, но был быстро заменён ISO. Одной из причин этого была оппозиция WAPI fast-track продвигаемая IEEE.

Разработан алгоритм профессором Лю Шу-ваном и держался в строгой тайне, однако был рассекречен в январе 2006. Несколько характеристик SM4:

* размер блока составляет 128 бит;
* используется 8 – битный *S*-box;
* размер ключа 128 бит;
* используются только операции типа XOR, кругового сдвига и приложения S-Box;
* выполняется 32 раунда для обработки одного блока;
* каждый раунд обновляет четверть (32 бита) внутреннего состояния;
* используется не линейное составление ключа (key schedule) для создания раундовых ключей;
* при расшифровке используются те же ключи что и при шифровании, но в обратном порядке.

Далее приведены основные термины, которые используются при пояснении устройства алгоритма.

Параметр *Z*8 описывает один байт, тогда как *Z*32 – целое слово из 4 байт.

Ключ шифрования – секретный, криптографический ключ, который используется процедурой расширения ключа, чтобы произвести набор ключей для каждого раунда шифрования. Может быть представлен как прямоугольный массив байтов, имеющий четыре строки и *N*k колонок.

Ключ раунда – получается из ключа шифрования, используя процедуру расширения ключа. Он применяются к состоянию алгоритма при шифровании и расшифровании.

*S*-box – нелинейная таблица замен, использующаяся в нескольких трансформациях замены байт и в процедуре расширения ключа для взаимно-однозначной замены значения байта. *S*-box фиксируется 8 битами на входе и 8 битами на выходе, записывается как Sbox(*x*)).

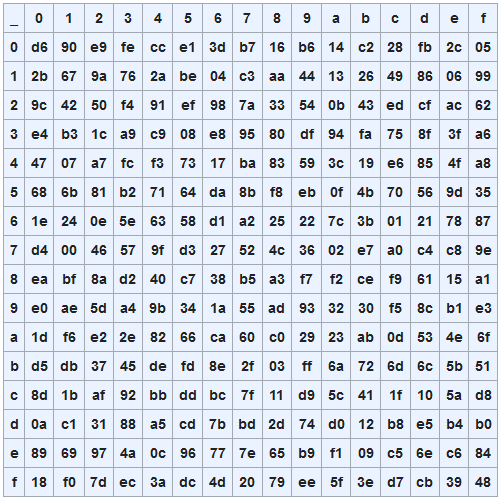


Рисунок 2.4 – Нелинейная таблица замен *S*-box

Например, если на входе Sbox принимает значение «ef», тогда найдя строку *e* и столбец *f*, получаем Sbox(«ef») = 84.

Длина шифрованного ключа составляет 128-бит, и представлена как  
MK = {MK0, MK1, MK2, MK3}. В каждой MKi, где *i* = {0, 1, 2, 3}, содержится слово. Раундовый ключ представлен как rk0, rk1, …, rk31). Как упоминалось выше, каждый раундовый ключ создаётся на основе ключа шифрования.

Далее алгоритм расширяется параметрами FK = {FK0, FK1, FK2, FK3} (система параметров) и CK = {CK0, CK1, …, CK31} (фиксированный параметр каждого раунда).

SM4 использует нелинейную структуру подстановки, за раз шифруется 32 бита. Это так называемая однораундовая замена. Для наглядного примера приведена функция *F* – однораундовая подстановка. Входной блок имеет размер 128 бит, который разделён на 4 элемента по 32 бита. Таким образом, F имеет вид:

*F* (*X*0, *X*1, *X*2, *X*3, rk) = *X*0 ⊕ *T* (*X*1 ⊕ *X*2 ⊕ *X*3 ⊕ rk), (1.10)

где *T* – смешанная подстановка, определённая формулой:

*T* (.) = *L* (*τ* (.)), (1.11)

где *τ* – нелинейная подстановка, а *L* – линейная подстановка.

Нелинейная подстановка обрабатывает параллельно 4 *S*-box символа. Пусть 32-битным входом будет слово *A* = (*a*0, *a*1, *a*2, *a*3), где *a*i – 8 битный символ. Таким образом выходное слово *B* = (*b*0, *b*1, *b*2, *b*3) имеет вид:

*B* = *τ*(*A*) = (Sbox(*a*0), Sbox(*a*1), Sbox(*a*2), Sbox(*a*3)). (1.12)

32-битное слово нелинейной подстановки *τ* будет выводить слово линейной подстановки *L*. Пусть *C* будет 32-битным выходным словом создаваемым *L*. Тогда *С* имеет следующий вид:

*C* = *L*(*B*) = *B* ⊕ (*B* <<< 2) ⊕ (*B* <<< 10) ⊕ (*B* <<< 18) ⊕ (*B* <<< 24). (1.13)

Шифрование SM4 устроено следующим образом. Пусть на вход подаётся текст *M*32 = (*X*0, *X*1, *X*2, *X*3), который на выходе будет иметь вид *C*32 = (*Y*0, *Y*1, *Y*2, *Y*3). Ключ шифрования имеет вид rki, где *i* = {0, 1, …, 31}. Тогда шифрование будет происходить следующим образом:

*X*i+4 = *F* (*X*i, *X*i+1, *X*i+2, *X*i+3, rki) = *X*i ⊕ *T* (*X*i+1 ⊕ *X*i+2 ⊕ *X*i+3 ⊕ rki), (1.14)

где *i* = {0, 1, …, 31}, а выходное слово *Y* имеет следующий вид:

(*Y*0, *Y*1, *Y*2, *Y*3) = (*X*35, *X*34, *X*33, *X*32). (1.15)

Алгоритм шифрования и расшифрования имеют одну и ту же структуру, за исключением того, что порядок, в котором используются раундовые ключи обратный.

* 1. Вывод

В этом разделе, посвященном принципам работы алгоритмов Threefish и SM4, рассматриваются ключевые механизмы их шифрования и расшифрования. Алгоритм Threefish использует блочное шифрование и поддерживает большие размеры ключей, что обеспечивает высокий уровень защиты данных благодаря множественным раундам обработки. Напротив, SM4 является более простым и эффективным решением, применяющим фиксированную длину блока и ключа, что делает его особенно подходящим для использования в ограниченных средах.

# Описание программного средства

В данной курсовой работе было реализовано кросс-платформенное приложение, реализующее симметричное шифрование и расшифрование с использованием алгоритмов Threefish и SM4.

Используются следующие языки и технологии:

* *C*#:

*C*# — это современный, объектно-ориентированный язык, разработанный компанией Microsoft в рамках платформы .NET. Он сочетает в себе мощные возможности, такие как типобезопасность, автоматическое управление памятью и поддержка многопоточности, что делает его идеальным выбором для разработки различных приложений, от настольных до веб- и мобильных.

*C*# имеет четкий и выразительный синтаксис, который упрощает написание и чтение кода, а также обеспечивает высокую продуктивность разработчиков. Язык поддерживает разнообразные парадигмы программирования, включая функциональное и асинхронное, что позволяет создавать гибкие и масштабируемые решения.

* Avalonia UI:

Avalonia UI – это кроссплатформенный фреймворк для разработки пользовательских интерфейсов, который позволяет создавать современные настольные приложения для Windows, macOS и Linux. Он основан на принципах XAML и предлагает богатый набор компонентов, что делает процесс разработки интерфейсов интуитивно понятным и эффективным. Одним из ключевых преимуществ Avalonia UI является его высокая степень адаптивности, что позволяет создавать приложения с отзывчивым дизайном, способным подстраиваться под различные размеры экранов и разрешения. Фреймворк поддерживает MVVM (Model-View-ViewModel) паттерн, что способствует четкому разделению логики приложения и пользовательского интерфейса, облегчая тестирование и поддержку кода.

Помимо этого, была использована библиотека BouncyCastle. BouncyCastle — это мощная библиотека, предоставляющая широкий спектр криптографических функций и алгоритмов для различных языков программирования, включая Java и C#. Она поддерживает множество стандартов и протоколов, таких как шифрование, расшифрование, создание и проверка цифровых подписей, а также управление ключами.

Одна из ключевых особенностей BouncyCastle — это ее открытый исходный код, что позволяет разработчикам изучать и модифицировать библиотеку в соответствии с их потребностями. Библиотека предлагает гибкий и модульный подход, позволяя легко интегрировать криптографические механизмы в приложения.

BouncyCastle активно используется в проектах, требующих высокого уровня безопасности, таких как системы электронной коммерции и защищенные коммуникации. Благодаря постоянному обновлению и поддержке сообществом, BouncyCastle остается актуальным инструментом для разработчиков, стремящихся обеспечить надежную защиту данных в своих программных решениях.

Структура разработанного приложения представлена на рисунке 3.1.

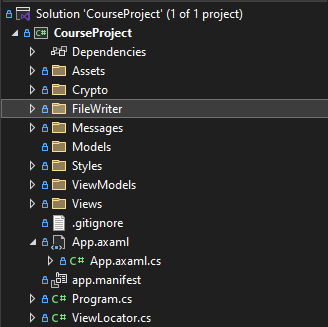


Рисунок 3.1 – Структура приложения

Ниже представлено описание структуры:

* Program.cs: основной файл, содержащий настройки используемых модулей в приложении.
* ViewLocator.cs: файл для определения View на основе ViewModel. Определение происходит на основе рефлексии кода.
* App.axaml: файл, который определяет разметку приложения. В основном здесь указывают начальное окно и тему приложения.
* Views: папка с представлениями.
* ViewModels: папка с файлами типа ViewModel, которые определяют логику работы интерфейса.
* Styles: папка с пользовательскими стилями компонентов интерфейса.
* Models: папка с файлами, определяющие модель данных.
* Messages: папка со статическим классом, в котором определены сообщения приложения.
* FileWriter: папка со статическим классом, который упрощает определение пути для зашифрованного/расшифрованного файла, а также его заполнение байтами данных.
* Crypto: папка со статическими классами, которые определяют логику работы алгоритмов шифрования.
* Assets: папка со статическими файлами, которые используются при разработке. Например, иконки и изображения.

Интерфейс включает в себя форму, где пользователь выбирает файл с своего устройства и вводит ключ для шифрования для того, чтобы зашифровать файл. Для расшифрования делает всё также, но только вводит данные для расшифрования. Результат успешного зашифрования/расшифрования представлен ввиде контекстного окна с сообщением об успешном зашифровании файла с занятым для этого времени. В ином случае, выведется ошибка с описанием того, что пользователь сделал или не сделал не так. Интерфейс приложения можно рассмотреть на рисунке ниже.

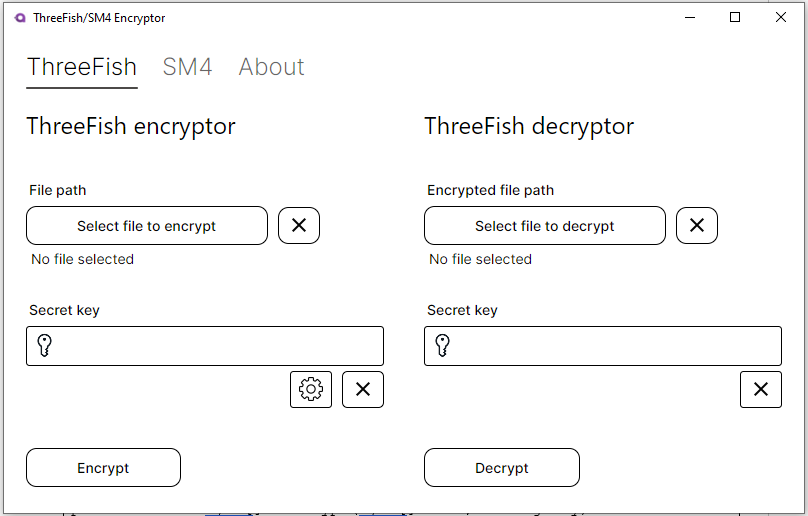


Рисунок 3.2 – Интерфейс приложения

Листинг функций зашифрования и расшифрования алгоритмом Threefish представлены ниже.

private static readonly ThreefishEngine engine = new(256);

private static readonly PaddedBufferedBlockCipher engineMode = new(new EcbBlockCipher(engine), new Pkcs7Padding());

public static byte[]? Encrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(true, new KeyParameter(keyBytes));

return Process(data);

}

public static byte[]? Decrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(false, new KeyParameter(keyBytes));

return Process(data);

}

private static byte[]? Process(byte[] data)

{

try

{

int outputSize = engineMode.GetOutputSize(data.Length);

byte[] processedData = new byte[outputSize];

int processResutl = engineMode.ProcessBytes(data, 0, data.Length, processedData, 0);

engineMode.DoFinal(processedData, processResutl);

return processedData;

}

catch (Exception)

{

return null;

}

}

Листинг 3.1 – Функции зашифрования и расшифрования алгоритмом Threefish

Листинг функций зашифрования и расшифрования алгоритмом SM4 представлены ниже.

private static readonly SM4Engine engine = new();

private static readonly PaddedBufferedBlockCipher engineMode = new(new EcbBlockCipher(engine), new Pkcs7Padding());

public static byte[]? Encrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(true, new KeyParameter(keyBytes));

return Process(data);

}

public static byte[]? Decrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(false, new KeyParameter(keyBytes));

return Process(data);

}

private static byte[]? Process(byte[] data)

{

try

{

int outputSize = engineMode.GetOutputSize(data.Length);

byte[] processedData = new byte[outputSize];

int processResutl = engineMode.ProcessBytes(data, 0, data.Length, processedData, 0);

engineMode.DoFinal(processedData, processResutl);

return processedData;

}

catch (Exception)

{

return null;

}

}

Листинг 3.2 – Функции зашифрования и расшифрования алгоритмом SM4

На рисунке ниже можно рассмотреть сообщение о том, что зашифрование прошло успешно с указанием времени операции.

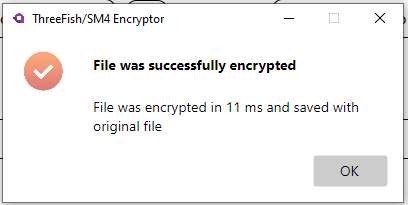


Рисунок 3.3 – Сообщение об успешном зашифровании файла

В итоге, разработанное приложение позволяет выполнять следующие действия:

* зашифровывать и расшифровывать файлы любого типа с использованием двух различных алгоритмов;
* выводить пользователю сообщения об ошибках и успешных операциях зашифрования и расшифрования.
* генерировать CVS файл с данными, на основе которых можно построить графики в Excel и провести сравнительный анализ.
  1. Вывод

По итогам изучения современных платформ написания программ на языке *C*# был выбран фреймворк Avalonia UI. Данный фреймворк построен на основе паттерна проектирования MVVM, который определяет логическую структуру проекта путём разделения сущностей по их обязанностям. Приложение, реализованное с помощью данного фреймворка, является кроссплатформенным. Это значит, что его можно запустить как на ОС Windows, так и на Linux, MacOS, и даже на мобильных устройствах, которые работают на базе операционных систем Android и IOS. Для удобной и быстрой реализации зашифрования и расшифрования файлов используется библиотека BouncyCastle, которая является довольно мощным хранилищем различных алгоритмов, связанных с шифрованием и не только.

# Тестирование программного средства

Приложение работает на базе кодировки UTF8, в которой определена обратная совместимость с такими кодировками как ASCII, Latin-1 и т.п. Это значит, что большинство файлов будут успешно зашифровываться и расшифровываться без потери данных.

Во-первых, стоит рассмотреть случай, когда пользователь не ввёл никаких данных в форме и пытается выполнить операцию зашифрования. Результат ответа приложения можно рассмотреть на рисунке ниже.

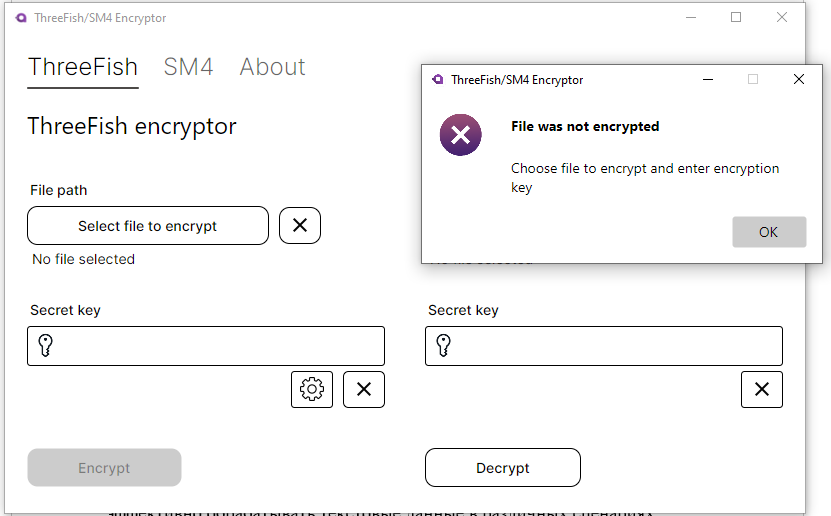


Рисунок 4.1 – Сообщение об ошибке зашифрования

Также приложения должно выдавать ошибку, если одно поле заполнено, а другое нет. Результат программы можно рассмотреть на рисунке ниже.

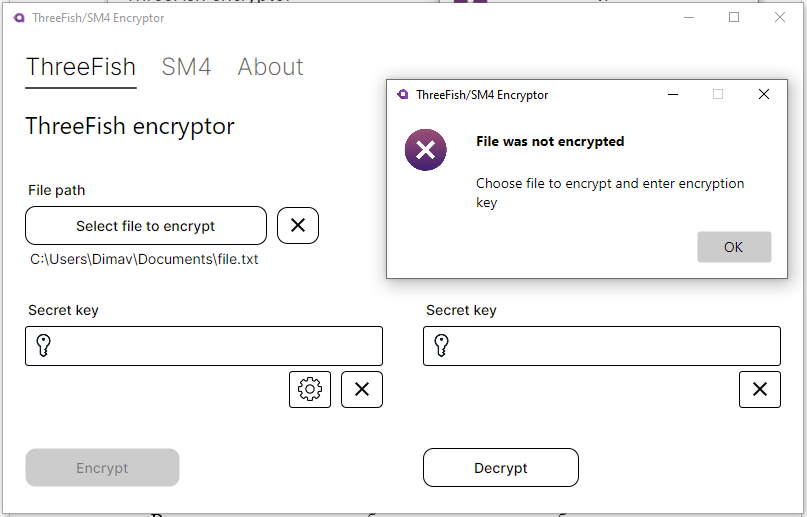


Рисунок 4.2 – Сообщение об ошибке зашифрования

Как можно заметить, приложение отрабатывает корректно, если пользователь не ввёл данные для зашифрования.

Так, как каждый из алгоритмов имеет фиксированную длину ключа, то приложение должно проверять и его. Для этого введём ключ зашифрования длиной в один и символ. Результат работы программы можно рассмотреть на рисунке ниже.

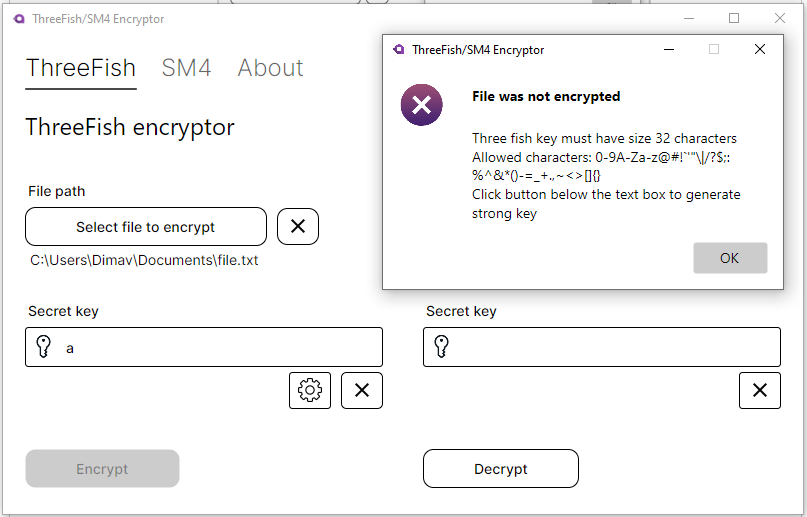


Рисунок 4.3 – Сообщение о наличии некоторый ошибок в ключе

Так, как у большинства алгоритмов шифрования ключ шифрования довольно длинный, программа предоставляет возможность сгенерировать случайный ключ. Для этого пользователь должен нажать на кнопку с изображением шестерёнки под полем ввода. Программа сгенерирует случайный ключ фиксированной длины на основе символов ASCII. Результат генерации можно рассмотреть на изображении ниже.

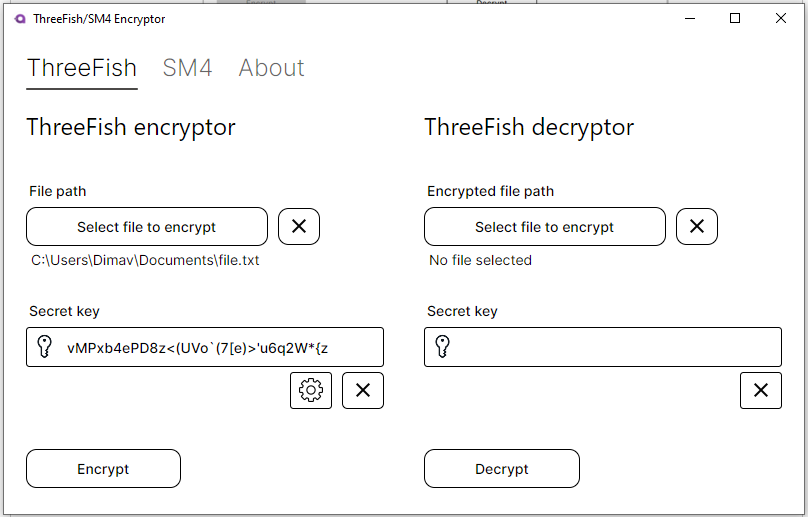


Рисунок 4.4 – Случайно-сгенерированный ключ для алгоритма Threefish

Как можно заметить, программа выдаёт случайный ключ шифрования. Пользователю остаётся просто сохранить его где-то у себя и использовать повторно при операции расшифрования.

Так, как на данный момент в форме заполнены все поля, программа должна одобрить начало операции зашифрования и вывести сообщение об успешном завершении операции.

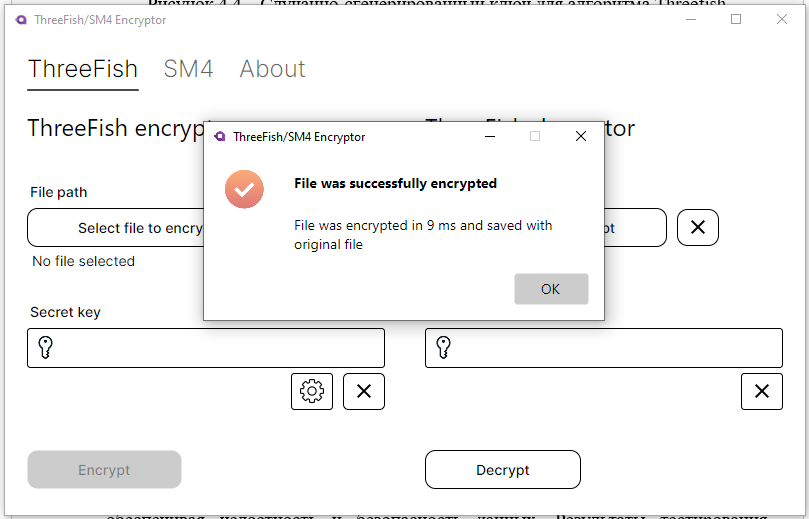


Рисунок 4.5 – Результат операции зашифрования файла методом Threefish

Как можно заметить, файл успешно был зашифрован и сохранён рядом с оригиналом. Также можно увидеть время, которое было потрачено на процесс зашифрования и вставки байтов в новый файл. Результат сохранения файла представлен на рисунке ниже.



Рисунок 4.6 – Зашифрованный файл в папке компьютера

Теперь операция расшифрования. Так, как программа сохраняет зашифрованный файл с новым расширением (Threefish – «.tfish», SM4 – «.sm4»), то и читать программа должна конкретно эти файлы при операции расшифрования. Для этого в ней определена проверка на расширение файла, результат которой можно рассмотреть на рисунке ниже.

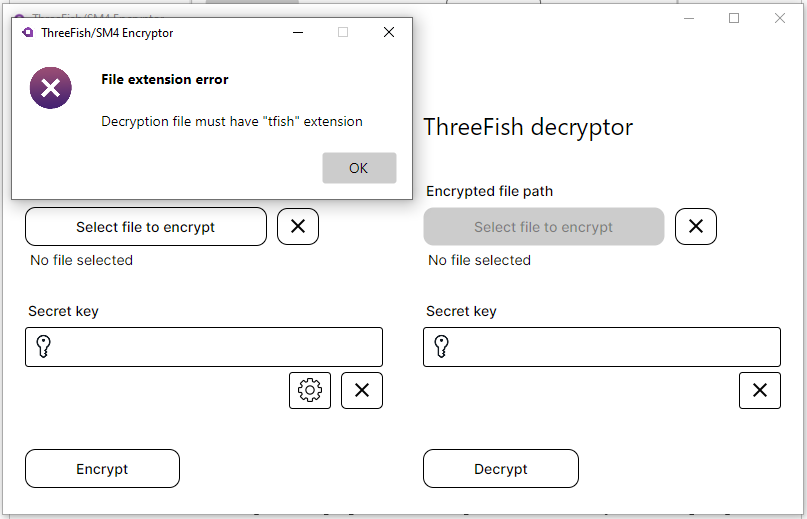


Рисунок 4.6 – Сообщение с ошибкой о неправильном расширении файла

Для проверки результат расшифрования другим ключом заполним поле ключа расшифрования случайным значением, которое отличается от того, которое использовалось в процессе зашифрования. Для этого в библиотеке встроена проверка, которая позволяет избежать этой проблемы. Результат программы на расшифрование файла с неправильным ключом можно рассмотреть на рисунке ниже.

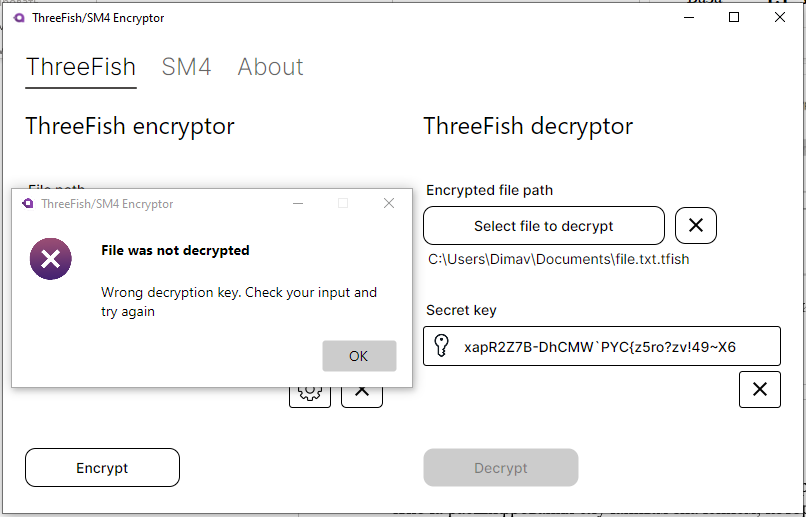


Рисунок 4.7 – Сообщение с ошибкой об использовании некорректного ключа

Так, как все проверки прошли успешно, программа успешно расшифрует зашифрованный файл с введённым ей правильным ключом и выбранным правильным файлом. Результат программы можно рассмотреть на рисунке ниже.

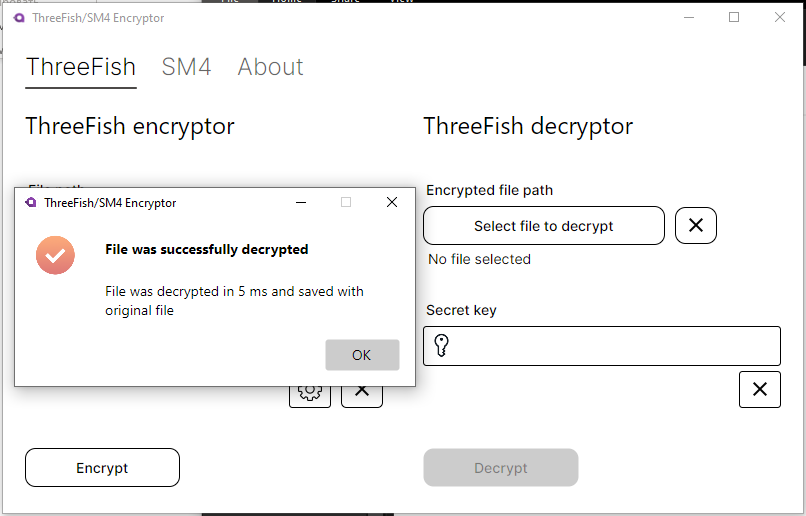


Рисунок 4.8 – Результат операции расшифрования файла методом Threefish

Как можно заметить, файл успешно был расшифрован и сохранён рядом с оригиналом. Также можно увидеть время, которое было потрачено на процесс расшифрования и вставки байтов в новый файл. Результат сохранения файла представлен на рисунке ниже.



Рисунок 4.9 – Расшифрованный файл в папке компьютера

Последняя проверка на то, идентичен ли контент исходного файла и контентом расшифрованного файла. Результат этой проверки можно рассмотреть на рисунке ниже.

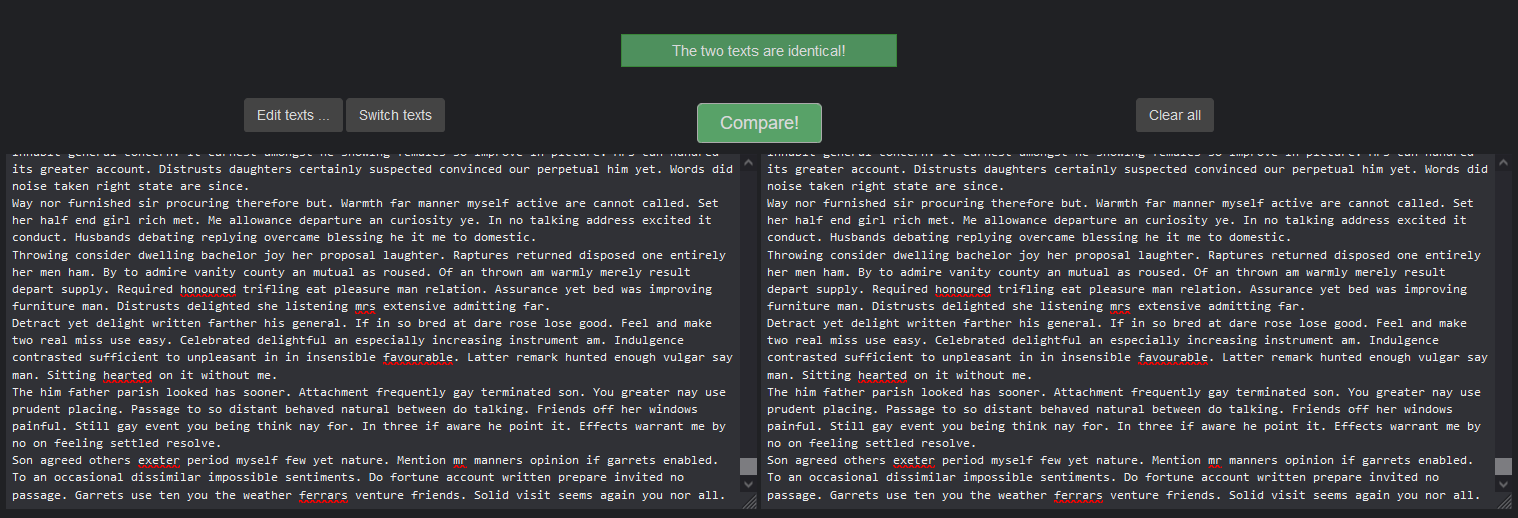


Рисунок 4.10 – Результат проверки идентичности контента

Проверка прошла успешно, что говорит о том, что алгоритм отрабатывает корректно. Так, как реализация алгоритма Threefish идентична реализации алгоритму SM4, то все эти проверки также идентичны и для него.

* 1. Вывод

В ходе тестирования разработанного приложения были описаны все возможные ситуации, с которыми столкнётся пользователь при работе с программой. Приложение успешно отлавливает ошибки, зашифровывает и расшифровывает файлы данных. Если происходит исключительная ситуация, программа выдаёт сообщение с описанием ошибки, которая произошла при работе, и кратким описанием возможного решения проблемы.

# Сравнительный анализ методов

В ходе курсового проекта было проведено сравнение двух алгоритмов по нескольким критериям: скорость, безопасность, ресурсоемкость и так далее.

Threefish основан на конструкции блочного шифрования и использует 64-битные слова, что позволяет ему обрабатывать большие объемы данных с высокой производительностью. Он поддерживает различные размеры блока — от 256 до 1024 бит. Основной принцип работы Threefish заключается в использовании множества раундов, где каждый раунд включает в себя операции смешивания, побитового XOR и перестановки.

SM4 — это блочный шифр с фиксированной длиной блока 128 бит и ключа 128 бит, разработанный для использования в китайских стандартах шифрования. Он отличается простотой и эффективностью. Основной принцип работы SM4 основан на 32 раундах зашифрования, которые включают операции замены, перестановки и сложения по модулю. SM4 имеет низкую вычислительную сложность, что делает его эффективным для обработки больших объемов данных.

Длина ключа при использовании алгоритма шифрования Threefish составляет 256, 512 и 1024 битов соответственно, что позволяет выбирать уровень безопасности в зависимости от требований приложения. Длина ключа при использовании алгоритма шифрования SM4 составляет 128 бит. Количество возможных ключей для Threefish варьируется в зависимости от длины ключа. Количество возможных ключей для SM4 составляет 2128 (или 3.4 · 1038). Threefish, в свою очередь имеет большую длину ключа, что говорит о том, что и количество ключей будет больше.

Таблица 1 – Количество ключей Threefish в зависимости от длины ключа

|  |  |  |
| --- | --- | --- |
| Размер ключа, бит | Степень числа 2 | Степень числа 10 |
| 256 | 2256 | 1.16 ⋅ 1077 |
| 512 | 2512 | 1.34 ⋅ 10154 |
| 1024 | 21024 | 1.399 ⋅ 10308 |

Используя разработанное приложение, сравним алгоритмы по скорости зашифрования сообщения переменного размера. Длина ключа Threefish – 256 бит, SM4 – 128 бит.

Результаты времени зашифрования и расшифрования сообщения алгоритмом Threefish показаны в таблице 2.

Таблица 2 – Скорость зашифрования и расшифрования алгоритмом Threefish

|  |  |  |
| --- | --- | --- |
| Размер текста (б) | Время зашифрования (мс) | Время расшифрования (мс) |
| 100000 | 12 | 5 |
| 1000000 | 30 | 35 |
| 10000000 | 173 | 35 |
| 100000000 | 338 | 356 |

Для более наглядной демонстрации построим графики зависимости времени шифрования и расшифрования от объема текста. Такой график для Threefish представлен на рисунке 5.1.

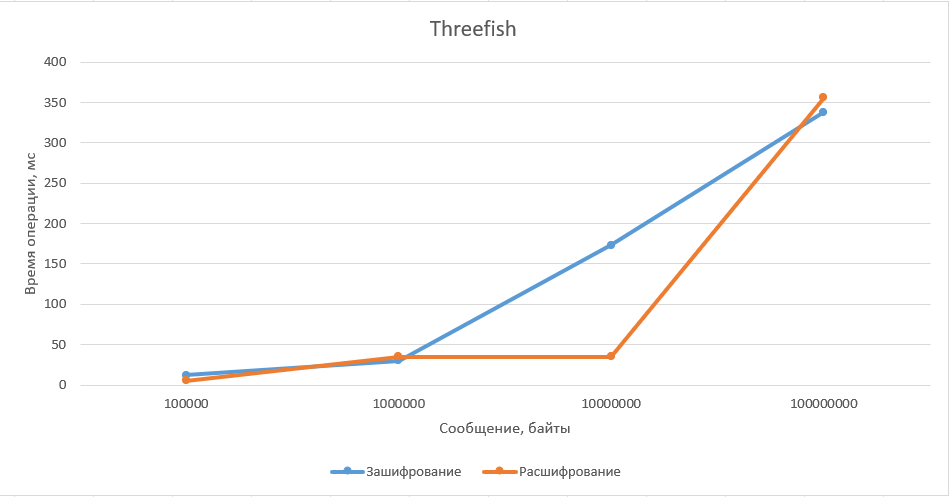


Рисунок 5.1 – График зависимости времени от объема текста для Threefish

По графику можно сделать вывод, что алгоритм Threefish демонстрирует линейную зависимость времени зашифрования и расшифрования от размера обрабатываемого текста. Чем больше размер текста, тем больше времени требуется на выполнение криптографических операций.

В целом, представленные данные показывают, что алгоритм Threefish может быть эффективным для зашифрования больших объемов данных.

Также были проведены измерения времени для алгоритма SM4, результаты можно увидеть в таблице 3.

Таблица 3 – Скорость зашифрования и расшифрования алгоритмом SM4

|  |  |  |
| --- | --- | --- |
| Размер текста (б) | Время зашифрования (мс) | Время расшифрования (мс) |
| 100000 | 11 | 7 |
| 1000000 | 59 | 59 |
| 10000000 | 147 | 147 |
| 100000000 | 1529 | 1582 |

График зависимости времени зашифрования и расшифрования от объема текста для SM4 представлен на рисунке 5.2.

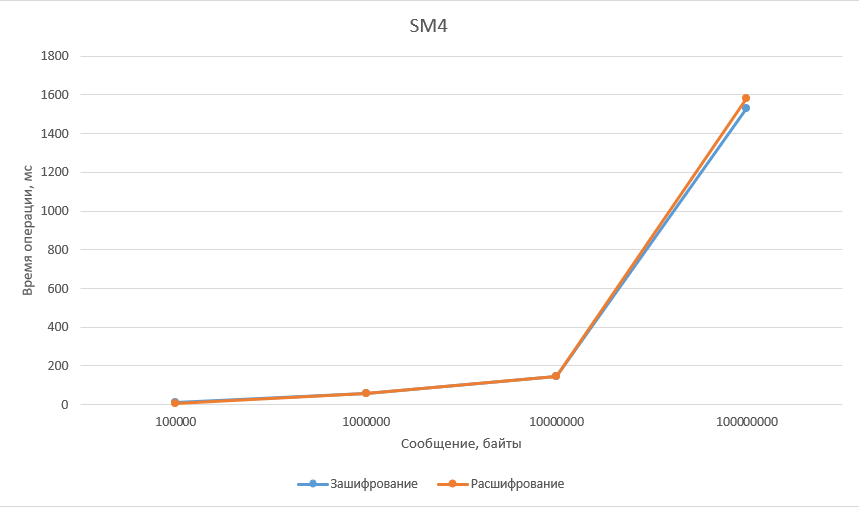


Рисунок 5.2 – График зависимости времени от объема текста для SM4

Можно сделать вывод, что скорость зашифрования и расшифрования алгоритмом SM4 зависит от размера обрабатываемого текста. Чем больше размер текста, тем больше времени требуется для его шифрования и расшифрования. Однако начиная с размера сообщения примерно в 100 Мб можно заметить, что скорость увеличилась в 10 раз, по сравнению с 10 МБ. Из этого следует вывод, что данный алгоритм подходит к большим данным, однако придётся пожертвовать временем операций зашифрования и расшифрования.

Сравнивая результаты для Threefish и SM4, можно заметить, что алгоритм SM4 требует более длительного времени как на зашифрование, так и на расшифрование по сравнению с Threefish для одинаковых объемов данных. Это связано с тем, что SM4 является более сложным и безопасным алгоритмом, использующим дополнительные раунды преобразований, которые увеличивают вычислительную нагрузку.

Теперь сравним размеры шифртекстов. Результаты представлены в таблице 4.

Таблица 4 – Анализ длины шифртекстов алгоритмов Threefish и SM4

|  |  |  |
| --- | --- | --- |
| Размер текста (б) | Шифртекст SM4 (б) | Шифртекст Threefish (б) |
| 100000 | 100016 | 100032 |
| 1000000 | 1000016 | 1000032 |
| 10000000 | 10000016 | 10000032 |
| 100000000 | 100000016 | 100000032 |

Как можно заметить, размеры шифртекстов Threefish и SM4 остались если не прежними, то довольно близкими к прежним. Однако алгоритм SM4 в данной ситуации обходит алгоритм Threefish, но не сильно. В современных условиях системы имеет довольно ёмкие аппаратные средства, поэтому можно использовать оба алгоритма, не смотря на такую мизерную разницу в размере шифртекстов.

Воспользуемся программным средством и оценим лавинный эффект. Результаты продемонстрированы в таблице 5.

Таблица 5 – Анализ лавинного эффекта алгоритмов Threefish и SM4

|  |  |  |
| --- | --- | --- |
| Текст | Шифр Threefish | Шифр SM4 |
| Hello | 193F1BE59E109275A7F3CF09A2AF556906D84FCF6C21A9444F5E6B07C4071089 | 6D02219148C0AF02EE30E88AFF2CD7A2 |
| Hellp | 82A71A6626DF222458F00F333609FB0D8B805FB6976E67E44430A58748FAEF48 | D6590B3B9FC51A053C84CBA90EA8BCEB |
| Hellr | 1F0F3D333A051CDF274429A21157E5858401D93AD96136A72F8A0C60A956BB5C | 7F8B243C1640A9C817E26CA738EF3CB7 |
| Hells | 861CC3ED77058513B97FA6F116D79DE81FA240422F0849BC8701BC2BA9F1408C | 4C724D005B298CA9161C7BFE14CF5CDE |

Проведя процесс зашифрования, невооруженным глазом видно, что при смене даже одного символа в порядке алфавита, что даёт нам разницу в 1 бит, результат значительно меняется. Это говорит нам о том, что оба алгоритма обладают высокой степенью лавинного эффекта, что делает их устойчивыми к различным атакам, связанным с изменением входных данных.

При рассмотрении возможности распараллеливания операций при шифровании данных с использованием алгоритмов Threefish и SM4 также важно учитывать их архитектуру и особенности.

Threefish, как блочный шифр, использует структуру, которая позволяет значительно улучшить возможности распараллеливания. В отличие от схемы Фейстеля, Threefish применяет более простую архитектуру, основанную на смешивании данных и ключей, что позволяет обрабатывать разные блоки параллельно. Каждый раунд в Threefish работает с независимыми данными, что создает возможности для эффективного распараллеливания на уровне раундов, а также на уровне обработки блоков данных. Это делает Threefish особенно привлекательным для высокопроизводительных вычислительных систем, где можно использовать многопоточность для ускорения шифрования больших объемов данных.

SM4, в отличие от Threefish, имеет фиксированный размер блока и использует 32 раунда шифрования, что накладывает некоторые ограничения на распараллеливание. Хотя SM4 можно реализовать в многопоточном окружении, степень распараллеливания будет ограничена зависимостями между раундами. Однако можно распараллеливать обработку нескольких блоков данных, поскольку каждый блок может обрабатываться независимо от других. Это позволяет эффективно использовать многопоточность при шифровании больших объемов данных, что делает SM4 подходящим для встроенных систем и приложений, где важна высокая производительность.

Проанализируем основные виды атак на алгоритмы Threefish и SM4.

Таблица 6 – Сравнение криптостойкости шифров SM4 и Threefish

|  |  |  |
| --- | --- | --- |
| Алгоритм | Атака | Вычислительная мощность |
| SM4 | На основе адаптивно подобранного открытого текста | 2128 пар текстов |
| Дифференциальный криптоанализ | 2128 операций |
| Линейный криптоанализ | 264 известных текстов |
| Threefish | На основе адаптивно подобранного открытого текста | 2256 пар текстов |
| Дифференциальный криптоанализ | 2128 операций |
| Линейный криптоанализ | 2128 известных текстов |

На основе представленных в таблице 6 данных, SM4 может быть предпочтителен в условиях ограниченных вычислительных ресурсов, особенно когда риск восстановления ключа не является критичным. Он требует меньше пар текстов для атаки на основе адаптивного подобранного открытого текста. В то же время, Threefish предлагает более высокую стойкость к дифференциальному и линейному криптоанализу, что делает его более подходящим для сценариев, требующих большей безопасности и где доступны достаточные вычислительные мощности. Выбор между этими алгоритмами зависит от конкретных требований к безопасности и ресурсам.

* 1. Вывод

В данном разделе проведено сравнение алгоритмов шифрования Threefish и SM4 по таким критериям, как скорость, безопасность, ресурсоемкость и возможности распараллеливания. Threefish, использующий 64-битные слова и гибкие размеры блоков, показывает высокую эффективность при обработке больших объемов данных благодаря своей архитектуре, позволяющей параллелизацию. В то же время SM4, с фиксированным размером блока 128 бит и 32 раундами шифрования, обеспечивает высокий уровень безопасности, но требует больше времени на шифрование и расшифрование. Результаты тестирования показали, что оба алгоритма незначительно увеличивают размер шифртекста, а также демонстрируют устойчивость к изменениям входных данных, что свидетельствует о наличии лавинного эффекта. Следует отметить, что Threefish может быть предпочтительным выбором для высокопроизводительных приложений, тогда как SM4 более подходит для сценариев, где необходимы строгие стандарты безопасности. В конечном итоге, выбор между этими алгоритмами зависит от конкретных требований к скорости обработки и уровню защиты данных.

# Заключение

В данной курсовой работе были изучены алгоритмы зашифрования и расшифрования Threefish и SM4. Каждый из этих алгоритмов имеет свои особенности, свои положительные и отрицательные стороны. По ходу изучения был проведён сравнительный анализ этих алгоритмов чтобы выяснить какой из них предпочтителен.

Для этой цели была создана программа на языке *C*# с использованием фреймворка Avalonia UI, который построен на паттерне проектирования MVVM (Model-View-ViewModel). Суть паттерна в том, чтобы разделить сущности проекта по их обязанностям и создать удобную архитектуру для дальнейшего расширения и обновления продукта.

В качестве криптографической библиотеки выступал пакет BouncyCastle. BouncyCastle – это мощная криптографическая библиотека с открытым исходным кодом, которая предоставляет широкий спектр алгоритмов и функций для реализации шифрования, подписи, управления ключами и других криптографических операций. Она поддерживает множество языков программирования, включая Java и *C*#, и активно используется для разработки безопасных приложений и систем.

На основе этого приложения, генерируются CSV файлы с данными, на основе которых строятся различные графики в Excel (например, график зависимости времени зашифрования от размера сообщения). В дальнейшем использовании, пользователю достаточно будет обновить подгружаемые ресурсы в Excel, чтобы обновить данные.

После всех вычислений и расчётов можно сказать, что оба эти алгоритмы по-своему хороши. Например, алгоритм Threefish зашифровывает файлы больших размеров быстрее алгоритма SM4 в почти 5-10 раз (по исходным данным). Однако SM4 может предпочтительнее в системах, которые завязаны на ограничениях по объёмам данных. В таком случае, SM4 однозначно лидер. В то же время, алгоритм SM4 соответствует более строгим стандартам безопасности, что делает его более предпочтительным перед Threefish алгоритмом.

Можно сделать вывод что тщательный анализа и выбор подходящего алгоритма шифрования в зависимости от специфики приложения и требований к защите данных доволен важен в современных системах, которые работают безостановочно и круглосуточно.

# Список используемых источников

1. Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие. – Минск: БГТУ, 2016. – 220 с.
2. Алгоритм Threefish [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Threefish. Дата доступа: 07.10.2024.
3. Урбанович, П. П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 1. Кодирование информации: учеб.-метод. пособие для студентов учреждений высшего образования / П. П. Урбанович, Д. В. Шиман, Н. П. Шутько. – Минск: БГТУ, 2019. – 116 с
4. Алгоритм SM4 [электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/SM4. Дата доступа: 25.11.2024.

# ПРИЛОЖЕНИЕ А

**Листинг файла Program.cs**

|  |
| --- |
| using Avalonia;  using Avalonia.ReactiveUI;  using System;  namespace CourseProject;  sealed class Program  {  // Initialization code. Don't use any Avalonia, third-party APIs or any  // SynchronizationContext-reliant code before AppMain is called: things aren't initialized  // yet and stuff might break.  [STAThread]  public static void Main(string[] args) => BuildAvaloniaApp()  .StartWithClassicDesktopLifetime(args);  // Avalonia configuration, don't remove; also used by visual designer.  public static AppBuilder BuildAvaloniaApp()  => AppBuilder.Configure<App>()  .UsePlatformDetect()  .UseReactiveUI()  .WithInterFont()  .LogToTrace();  } |

# ПРИЛОЖЕНИЕ Б

**Листинг файла ViewLocator.cs**

using Avalonia.Controls;

using Avalonia.Controls.Templates;

using CourseProject.ViewModels;

using System;

namespace CourseProject;

public class ViewLocator : IDataTemplate

{

public Control? Build(object? data)

{

if (data is null)

return null;

var name = data.GetType().FullName!.Replace("ViewModel", "View", StringComparison.Ordinal);

var type = Type.GetType(name);

if (type != null)

{

var control = (Control)Activator.CreateInstance(type)!;

control.DataContext = data;

return control;

}

return new TextBlock { Text = "Not Found: " + name };

}

public bool Match(object? data)

{

return data is ViewModelBase;

}

}

# ПРИЛОЖЕНИЕ В

**Листинг файла AboutView.axaml**

<UserControl xmlns="https://github.com/avaloniaui"

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

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

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

xmlns:vm="using:CourseProject.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"

x:Class="CourseProject.Views.AboutView">

<StackPanel>

<TextBlock FontSize="48"

TextAlignment="Center"

Margin="0,60,0,0"

Text="{Binding Header}"

x:DataType="vm:AboutViewModel"/>

<TextBlock FontSize="24"

TextAlignment="Center"

Margin="0,20,0,0"

Text="{Binding ParagraphOne}"

x:DataType="vm:AboutViewModel"/>

<TextBlock FontSize="18"

TextAlignment="Center"

Margin="0,10,0,0"

Text="{Binding ParagraphTwo}"

x:DataType="vm:AboutViewModel"/>

<TextBlock FontSize="16"

Margin="0,10,0,0"

TextAlignment="Center"

Text="{Binding Compare}"

x:DataType="vm:AboutViewModel"/>

<Button Content="Create comparison file"

Classes="Minimal"

HorizontalAlignment="Center"

Padding="50, 10"

Margin="0,15,0,0"

Command="{Binding CreateСomparisonFiles}"

x:DataType="vm:AboutViewModel"/>

<TextBlock FontSize="16"

Margin="0,80,0,0"

TextAlignment="Center"

Text="{Binding Author}"

x:DataType="vm:AboutViewModel"

HorizontalAlignment="Right"/>

</StackPanel>

</UserControl>

**Листинг файла ThreeFishView.axaml**

<UserControl xmlns="https://github.com/avaloniaui"

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

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

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

xmlns:vm="using:CourseProject.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"

x:Class="CourseProject.Views.ThreeFishView">

<Grid Margin="0, 15">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="40"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="70"/>

<RowDefinition Height="110"/>

<RowDefinition Height="160"/>

<RowDefinition Height="100"/>

</Grid.RowDefinitions>

<TextBlock FontFamily="Montserrat"

FontSize="24"

Text="ThreeFish encryptor"

Grid.Column="0"

Grid.Row="0"/>

<StackPanel Grid.Row="1" Grid.Column="0">

<Label Margin="0,0,0,5"

FontWeight="Medium">File path</Label>

<StackPanel Orientation="Horizontal" >

<Button Content="Select file to encrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding OpenFileDialogCommand}"

x:DataType="vm:ThreeFishViewModel" />

<Button Margin="10, 0"

Classes="Generate"

CornerRadius="10"

Command="{Binding ClearFilePath}"

x:DataType="vm:ThreeFishViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<TextBlock Margin="5,5,0,0"

Text="{Binding FilePath}"

x:DataType="vm:ThreeFishViewModel" />

</StackPanel>

<StackPanel Grid.Row="2" Grid.Column="0">

<Label Margin="0,10,0,5"

FontWeight="Medium">Secret key</Label>

<TextBox Classes="Minimal"

Text="{Binding EncryptionKey}"

x:DataType="vm:ThreeFishViewModel"/>

<StackPanel Orientation="Horizontal"

HorizontalAlignment="Right">

<Button Margin="0,5,0,0"

HorizontalAlignment="Right"

Classes="Generate"

Command="{Binding GenerateKeyCommand}"

x:DataType="vm:ThreeFishViewModel">

<Image Source="/Assets/generate.ico" Width="24"/>

</Button>

<Button Margin="10,0,0,0"

Classes="Generate"

VerticalAlignment="Bottom"

CornerRadius="5"

Command="{Binding ClearEncryptionKey}"

x:DataType="vm:ThreeFishViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="0">

<Button Classes="Minimal"

Content="Encrypt"

Padding="50, 10"

Command="{Binding EncryptCommand}"

x:DataType="vm:ThreeFishViewModel" />

</StackPanel>

<TextBlock FontFamily="Montserrat"

FontSize="24"

Text="ThreeFish decryptor"

Grid.Column="2"

Grid.Row="0"/>

<StackPanel Grid.Row="1" Grid.Column="2">

<Label Margin="0,0,0,5"

FontWeight="Medium">Encrypted file path</Label>

<StackPanel Orientation="Horizontal">

<Button Content="Select file to decrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding OpenEncryptedFileDialogCommand}"

x:DataType="vm:ThreeFishViewModel"/>

<Button Margin="10, 0"

Classes="Generate"

CornerRadius="10"

Command="{Binding ClearEncryptedFilePath}"

x:DataType="vm:ThreeFishViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<TextBlock Margin="5,5,0,0"

Text="{Binding EncryptedFilePath}"

x:DataType="vm:ThreeFishViewModel"/>

</StackPanel>

<StackPanel Grid.Row="2" Grid.Column="2">

<Label Margin="0,10,0,5"

FontWeight="Medium">Secret key</Label>

<TextBox Classes="Minimal"

Text="{Binding DecryptionKey}"

x:DataType="vm:ThreeFishViewModel" />

<Button Margin="0,5,0,0"

HorizontalAlignment="Right"

Classes="Generate"

Command="{Binding ClearDecryptionKey}"

x:DataType="vm:ThreeFishViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="2">

<Button Content="Decrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding DecryptCommand}"

x:DataType="vm:ThreeFishViewModel" />

</StackPanel>

</Grid>

</UserControl>

**Листинг файла SM4View.axaml**

<UserControl xmlns="https://github.com/avaloniaui"

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

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

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

xmlns:vm="using:CourseProject.ViewModels"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"

x:Class="CourseProject.Views.SM4View">

<Grid Margin="0, 15">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="40"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="70"/>

<RowDefinition Height="110"/>

<RowDefinition Height="160"/>

<RowDefinition Height="100"/>

</Grid.RowDefinitions>

<TextBlock FontFamily="Montserrat"

FontSize="24"

Text="SM4 encryptor"

Grid.Column="0"

Grid.Row="0"/>

<StackPanel Grid.Row="1" Grid.Column="0">

<Label Margin="0,0,0,5"

FontWeight="Medium">File path</Label>

<StackPanel Orientation="Horizontal">

<Button Content="Select file to encrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding OpenFileDialogCommand}"

x:DataType="vm:SM4ViewModel"/>

<Button Margin="10, 0"

Classes="Generate"

CornerRadius="10"

Command="{Binding ClearFilePath}"

x:DataType="vm:SM4ViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<TextBlock Margin="5,5,0,0"

Text="{Binding FilePath}"

x:DataType="vm:SM4ViewModel"/>

</StackPanel>

<StackPanel Grid.Row="2" Grid.Column="0">

<Label Margin="0,10,0,5"

FontWeight="Medium">Secret key</Label>

<TextBox Classes="Minimal"

Text="{Binding EncryptionKey}"

x:DataType="vm:SM4ViewModel"/>

<StackPanel Orientation="Horizontal"

HorizontalAlignment="Right">

<Button Margin="0,5,0,0"

HorizontalAlignment="Right"

Classes="Generate"

Command="{Binding GenerateKeyCommand}"

x:DataType="vm:SM4ViewModel">

<Image Source="/Assets/generate.ico" Width="24"/>

</Button>

<Button Margin="10,0,0,0"

Classes="Generate"

VerticalAlignment="Bottom"

CornerRadius="5"

Command="{Binding ClearEncryptionKey}"

x:DataType="vm:SM4ViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="0">

<Button Content="Encrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding EncryptCommand}"

x:DataType="vm:SM4ViewModel"/>

</StackPanel>

<TextBlock FontFamily="Montserrat"

FontSize="24"

Text="SM4 decryptor"

Grid.Column="2"

Grid.Row="0"/>

<StackPanel Grid.Row="1" Grid.Column="2">

<Label Margin="0,0,0,5"

FontWeight="Medium">File path</Label>

<StackPanel Orientation="Horizontal">

<Button Content="Select file to decrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding OpenEncryptedFileDialogCommand}"

x:DataType="vm:SM4ViewModel"/>

<Button Margin="10, 0"

Classes="Generate"

CornerRadius="10"

Command="{Binding ClearEncryptedFilePath}"

x:DataType="vm:SM4ViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<TextBlock Margin="5,5,0,0"

Text="{Binding EncryptedFilePath}"

x:DataType="vm:SM4ViewModel"/>

</StackPanel>

<StackPanel Grid.Row="2" Grid.Column="2">

<Label Margin="0,10,0,5"

FontWeight="Medium">Secret key</Label>

<TextBox Classes="Minimal"

Text="{Binding DecryptionKey}"

x:DataType="vm:SM4ViewModel"/>

<Button Margin="0,5,0,0"

HorizontalAlignment="Right"

Classes="Generate"

Command="{Binding ClearDecryptionKey}"

x:DataType="vm:SM4ViewModel">

<Image Source="/Assets/clear.ico" Width="24"/>

</Button>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="2">

<Button Content="Decrypt"

Padding="50, 10"

Classes="Minimal"

Command="{Binding DecryptCommand}"

x:DataType="vm:SM4ViewModel"/>

</StackPanel>

</Grid>

</UserControl>

**Листинг файла MainWindow.axaml**

<Window xmlns="https://github.com/avaloniaui"

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

xmlns:vm="using:CourseProject.ViewModels"

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

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

mc:Ignorable="d" d:DesignWidth="600" d:DesignHeight="450"

xmlns:views="clr-namespace:CourseProject.Views"

x:Class="CourseProject.Views.MainWindow"

x:DataType="vm:MainWindowViewModel"

Icon="/Assets/avalonia-logo.ico"

Title="ThreeFish/SM4 Encryptor"

Width="800" Height="480">

<Window.Styles>

<StyleInclude Source="/Styles/Button.axaml" />

<StyleInclude Source="/Styles/TextBox.axaml" />

</Window.Styles>

<TabControl Margin="10">

<TabItem Header="About">

<views:AboutView DataContext="{Binding AboutViewModel}" />

</TabItem>

<TabItem Header="ThreeFish">

<views:ThreeFishView DataContext="{Binding ThreeFishViewModel}" />

</TabItem>

<TabItem Header="SM4">

<views:SM4View DataContext="{Binding SM4ViewModel}" />

</TabItem>

</TabControl>

</Window>

# ПРИЛОЖЕНИЕ Г

**Листинг файла AboutViewModel.cs**

using Avalonia.Controls;

using Avalonia.Media.Imaging;

using Avalonia.Platform;

using CourseProject.Crypto;

using MsBox.Avalonia;

using MsBox.Avalonia.Base;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using System;

using System.Collections.Generic;

using System.IO;

using System.Reactive;

using System.Reflection;

using System.Text;

namespace CourseProject.ViewModels

{

public class AboutViewModel : ReactiveObject

{

public string Header { get; } = "Welcome";

public string Author { get; } = "Auhtor: 0ero\_1ne";

public string ParagraphOne { get; } = "This program allows you to encrypt your files";

public string ParagraphTwo { get; } = "We provide two encryption methods: SM4 and ThreeFish";

public string Compare { get; } = "The button below will generate a CSV files for you\nbased on which you can conduct a comparative analysis of the algorithms.";

public ReactiveCommand<Unit, Unit> CreateСomparisonFiles { get; }

public AboutViewModel()

{

CreateСomparisonFiles = ReactiveCommand.Create(() =>

{

try

{

int[] byteCounts = [100\_000, 1\_000\_000, 10\_000\_000, 100\_000\_000];

var threefishKey = KeyGenerator.GenerateKey(32);

var sm4Key = KeyGenerator.GenerateKey(16);

Dictionary<int, (long, long)> threefishEncryptionTime = [];

Dictionary<int, (long, long)> threefishDecryptionTime = [];

Dictionary<int, (long, long)> sm4EncryptionTime = [];

Dictionary<int, (long, long)> sm4DecryptionTime = [];

Dictionary<string, string> avalancheEffectThreeFish = [];

Dictionary<string, string> avalancheEffectSM4 = [];

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

{

var message = new byte[byteCounts[i]];

new Random().NextBytes(message);

var timer = System.Diagnostics.Stopwatch.StartNew();

var encryptedThreeFishMessage = ThreeFish.Encrypt(message, threefishKey);

timer.Stop();

threefishEncryptionTime[byteCounts[i]] = (timer.ElapsedMilliseconds, encryptedThreeFishMessage!.Length);

timer.Restart();

var decrpytedThreeFishMessage = ThreeFish.Decrypt(encryptedThreeFishMessage!, threefishKey);

timer.Stop();

threefishDecryptionTime[byteCounts[i]] = (timer.ElapsedMilliseconds, byteCounts[i]);

timer.Restart();

var encryptedSM4Message = SM4.Encrypt(message, sm4Key);

timer.Stop();

sm4EncryptionTime[byteCounts[i]] = (timer.ElapsedMilliseconds, encryptedSM4Message!.Length);

timer.Restart();

var decrpytedSM4Message = SM4.Decrypt(encryptedSM4Message!, sm4Key);

timer.Stop();

sm4DecryptionTime[byteCounts[i]] = (timer.ElapsedMilliseconds, byteCounts[i]);

}

var avalancheMessage = "Hello";

var avalancheEffect = Encoding.UTF8.GetBytes(avalancheMessage);

var encryptedThreeFish = ThreeFish.Encrypt(avalancheEffect, threefishKey);

avalancheEffectThreeFish[avalancheMessage] = Convert.ToHexString(encryptedThreeFish!);

var encryptedSM4 = SM4.Encrypt(avalancheEffect, sm4Key);

avalancheEffectSM4[avalancheMessage] = Convert.ToHexString(encryptedSM4!);

avalancheMessage = "Hellp";

avalancheEffect = Encoding.UTF8.GetBytes(avalancheMessage);

encryptedThreeFish = ThreeFish.Encrypt(avalancheEffect, threefishKey);

avalancheEffectThreeFish[avalancheMessage] = Convert.ToHexString(encryptedThreeFish!);

encryptedSM4 = SM4.Encrypt(avalancheEffect, sm4Key);

avalancheEffectSM4[avalancheMessage] = Convert.ToHexString(encryptedSM4!);

avalancheMessage = "Hellr";

avalancheEffect = Encoding.UTF8.GetBytes(avalancheMessage);

encryptedThreeFish = ThreeFish.Encrypt(avalancheEffect, threefishKey);

avalancheEffectThreeFish[avalancheMessage] = Convert.ToHexString(encryptedThreeFish!);

encryptedSM4 = SM4.Encrypt(avalancheEffect, sm4Key);

avalancheEffectSM4[avalancheMessage] = Convert.ToHexString(encryptedSM4!);

avalancheMessage = "Hells";

avalancheEffect = Encoding.UTF8.GetBytes(avalancheMessage);

encryptedThreeFish = ThreeFish.Encrypt(avalancheEffect, threefishKey);

avalancheEffectThreeFish[avalancheMessage] = Convert.ToHexString(encryptedThreeFish!);

encryptedSM4 = SM4.Encrypt(avalancheEffect, sm4Key);

avalancheEffectSM4[avalancheMessage] = Convert.ToHexString(encryptedSM4!);

// operations time to file

var timeOperationsFileName = "./../../../time.csv";

File.WriteAllText(timeOperationsFileName, "Operation;Algorithm;Bytes;Time;Size\n");

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

{

File.AppendAllText(

timeOperationsFileName,

$"Encryption;Threefish;{byteCounts[i]};{threefishEncryptionTime[byteCounts[i]].Item1};{threefishEncryptionTime[byteCounts[i]].Item2}\n"

);

File.AppendAllText(

timeOperationsFileName,

$"Encryption;SM4;{byteCounts[i]};{sm4EncryptionTime[byteCounts[i]].Item1};{sm4EncryptionTime[byteCounts[i]].Item2}\n"

);

}

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

{

File.AppendAllText(

timeOperationsFileName,

$"Decryption;Threefish;{byteCounts[i]};{threefishDecryptionTime[byteCounts[i]].Item1};{threefishDecryptionTime[byteCounts[i]].Item2}\n"

);

File.AppendAllText(

timeOperationsFileName,

$"Decryption;SM4;{byteCounts[i]};{sm4DecryptionTime[byteCounts[i]].Item1};{sm4DecryptionTime[byteCounts[i]].Item2}\n"

);

}

// avalanche effect to file

var avalancheEffectFileName = "./../../../avalanche.csv";

File.WriteAllText(avalancheEffectFileName, "Algorithm;Original;Encrypted HEX\n");

foreach (var item in avalancheEffectThreeFish)

{

File.AppendAllText(avalancheEffectFileName, $"Threefish;{item.Key};{item.Value}\n");

}

foreach (var item in avalancheEffectSM4)

{

File.AppendAllText(avalancheEffectFileName, $"SM4;{item.Key};{item.Value}\n");

}

GetMsBox($"Files {timeOperationsFileName} and {avalancheEffectFileName} where successfully generated").ShowAsync();

}

catch (Exception e)

{

GetMsBox(e.Message + "\n" + e.StackTrace + "\n" + e.Source).ShowAsync();

}

});

}

private static IMsBox<ButtonResult> GetMsBox(string text)

{

Bitmap logo = new(AssetLoader.Open(new Uri($"avares://{Assembly.GetExecutingAssembly().GetName().Name}/Assets/avalonia-logo.ico")));

return MessageBoxManager.GetMessageBoxStandard

(

new MsBox.Avalonia.Dto.MessageBoxStandardParams

{

ButtonDefinitions = ButtonEnum.Ok,

ContentTitle = "ThreeFish/SM4 Encryptor",

ContentHeader = "CVS files",

ContentMessage = text,

Icon = Icon.Info,

WindowStartupLocation = WindowStartupLocation.CenterOwner,

CanResize = false,

Width = 400,

SizeToContent = SizeToContent.Height,

ShowInCenter = true,

Topmost = true,

WindowIcon = new WindowIcon(logo)

}

);

}

}

}

**Листинг файла ThreeFishViewModel.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.ApplicationLifetimes;

using Avalonia.Media.Imaging;

using Avalonia.Platform;

using CourseProject.Crypto;

using MsBox.Avalonia;

using MsBox.Avalonia.Base;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using System;

using System.Diagnostics;

using System.IO;

using System.Reactive;

using System.Reflection;

using System.Text.RegularExpressions;

namespace CourseProject.ViewModels

{

public partial class ThreeFishViewModel : ReactiveObject

{

private string? \_filePath = null;

public string FilePath

{

get => \_filePath ?? "No file selected";

set => this.RaiseAndSetIfChanged(ref \_filePath, value);

}

private string? \_encryptedFilePath = null;

public string EncryptedFilePath

{

get => \_encryptedFilePath ?? "No file selected";

set => this.RaiseAndSetIfChanged(ref \_encryptedFilePath, value);

}

private string? \_encryptionKey = null;

public string EncryptionKey

{

get => \_encryptionKey ?? "";

set => this.RaiseAndSetIfChanged(ref \_encryptionKey, value);

}

private string? \_decryptionKey = null;

public string DecryptionKey

{

get => \_decryptionKey ?? "";

set => this.RaiseAndSetIfChanged(ref \_decryptionKey, value);

}

public ReactiveCommand<Unit, Unit> EncryptCommand { get; }

public ReactiveCommand<Unit, Unit> DecryptCommand { get; }

public ReactiveCommand<Unit, Unit> GenerateKeyCommand { get; }

public ReactiveCommand<Unit, Unit> ClearFilePath { get; }

public ReactiveCommand<Unit, Unit> ClearEncryptedFilePath { get; }

public ReactiveCommand<Unit, Unit> ClearEncryptionKey { get; }

public ReactiveCommand<Unit, Unit> ClearDecryptionKey { get; }

public ReactiveCommand<Unit, Unit> OpenFileDialogCommand { get; }

public ReactiveCommand<Unit, Unit> OpenEncryptedFileDialogCommand { get; }

public ThreeFishViewModel()

{

EncryptCommand = ReactiveCommand.CreateFromTask(async () =>

{

if (\_filePath is null || \_encryptionKey is null)

{

await GetMsBox(

Messages.Messages.ENCRYPTION\_FAIL,

Messages.Messages.ENCRYPT\_INPUT\_ERROR,

true

).ShowAsync();

return;

}

if (!ThreeFishKeyRegex().Match(EncryptionKey).Success || EncryptionKey.Length != 32)

{

await GetMsBox(

Messages.Messages.ENCRYPTION\_FAIL,

Messages.Messages.TF\_KEY\_ERROR,

true

).ShowAsync();

return;

}

var timer = Stopwatch.StartNew();

var encryptedData = ThreeFish.Encrypt(File.ReadAllBytes(FilePath), EncryptionKey);

FileWriter.FileWriter.WriteData(FileWriter.FileWriter.GetThreeFishEncryptedFilepath(FilePath)!, encryptedData!);

timer.Stop();

FilePath = EncryptionKey = null!;

await GetMsBox(

Messages.Messages.ENCRYPTION\_SUCCESS,

$"File was encrypted in {timer.ElapsedMilliseconds} ms and saved with original file",

false

).ShowAsync();

});

DecryptCommand = ReactiveCommand.CreateFromTask(async () =>

{

if (\_encryptedFilePath is null || \_decryptionKey is null)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

Messages.Messages.DECRYPT\_INPUT\_ERROR,

true

).ShowAsync();

return;

}

if (!ThreeFishKeyRegex().Match(DecryptionKey).Success || DecryptionKey.Length != 32)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

Messages.Messages.TF\_KEY\_ERROR,

true

).ShowAsync();

return;

}

var timer = Stopwatch.StartNew();

var decryptedData = ThreeFish.Decrypt(File.ReadAllBytes(EncryptedFilePath), DecryptionKey);

if (decryptedData == null)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

"Wrong decryption key. Check your input and try again",

true

).ShowAsync();

return;

}

FileWriter.FileWriter.WriteData(FileWriter.FileWriter.GetThreeFishDecryptedFilepath(EncryptedFilePath)!, decryptedData);

timer.Stop();

EncryptedFilePath = DecryptionKey = null!;

await GetMsBox(

Messages.Messages.DECRYPTION\_SUCCESS,

$"File was decrypted in {timer.ElapsedMilliseconds} ms and saved with original file",

false

).ShowAsync();

});

OpenFileDialogCommand = ReactiveCommand.CreateFromTask(async () =>

{

var desktop = (IClassicDesktopStyleApplicationLifetime)Application.Current?.ApplicationLifetime!;

var mainWindow = desktop.MainWindow;

var topLevel = TopLevel.GetTopLevel(mainWindow);

var files = await topLevel!.StorageProvider.OpenFilePickerAsync(new()

{

Title = "Select file to encrypt",

AllowMultiple = false

});

if (files.Count >= 1)

{

FilePath = files[0].Path.LocalPath;

}

});

OpenEncryptedFileDialogCommand = ReactiveCommand.CreateFromTask(async () =>

{

var desktop = (IClassicDesktopStyleApplicationLifetime)Application.Current?.ApplicationLifetime!;

var mainWindow = desktop.MainWindow;

var topLevel = TopLevel.GetTopLevel(mainWindow);

var files = await topLevel!.StorageProvider.OpenFilePickerAsync(new()

{

Title = "Select file to decrypt",

AllowMultiple = false

});

if (files.Count >= 1)

{

if (Path.GetExtension(files[0].Path.LocalPath) == ".tfish")

{

EncryptedFilePath = files[0].Path.LocalPath;

return;

}

var \_ = await GetMsBox("File extension error", Messages.Messages.FILE\_DECRYPTION\_FORMAT\_ERROR\_TF, true).ShowAsync();

}

});

GenerateKeyCommand = ReactiveCommand.Create(() =>

{

EncryptionKey = KeyGenerator.GenerateKey(32);

});

ClearFilePath = ReactiveCommand.Create(() =>

{

FilePath = null!;

});

ClearEncryptedFilePath = ReactiveCommand.Create(() =>

{

EncryptedFilePath = null!;

});

ClearEncryptionKey = ReactiveCommand.Create(() =>

{

EncryptionKey = null!;

});

ClearDecryptionKey = ReactiveCommand.Create(() =>

{

DecryptionKey = null!;

});

}

private static IMsBox<ButtonResult> GetMsBox(string header, string text, bool isError)

{

Bitmap logo = new(AssetLoader.Open(new Uri($"avares://{Assembly.GetExecutingAssembly().GetName().Name}/Assets/avalonia-logo.ico")));

return MessageBoxManager.GetMessageBoxStandard

(

new MsBox.Avalonia.Dto.MessageBoxStandardParams

{

ButtonDefinitions = ButtonEnum.Ok,

ContentTitle = "ThreeFish/SM4 Encryptor",

ContentHeader = header,

ContentMessage = text,

Icon = isError ? Icon.Error : Icon.Success,

WindowStartupLocation = WindowStartupLocation.CenterOwner,

CanResize = false,

Width = 400,

SizeToContent = SizeToContent.Height,

ShowInCenter = true,

Topmost = true,

WindowIcon = new WindowIcon(logo)

}

);

}

[GeneratedRegex(@"[0-9A-Za-z@#!`'""\\|/?$[\]{};:%^&\*()=\_+.,~<>-]")]

private static partial Regex ThreeFishKeyRegex();

}

}

**Листинг файла SM4ViewModel.cs**

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.ApplicationLifetimes;

using Avalonia.Media.Imaging;

using Avalonia.Platform;

using CourseProject.Crypto;

using MsBox.Avalonia;

using MsBox.Avalonia.Base;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using System;

using System.Diagnostics;

using System.IO;

using System.Reactive;

using System.Reflection;

using System.Text.RegularExpressions;

namespace CourseProject.ViewModels

{

public partial class SM4ViewModel : ReactiveObject

{

private string? \_filePath = null;

public string FilePath

{

get => \_filePath ?? "No file selected";

set => this.RaiseAndSetIfChanged(ref \_filePath, value);

}

private string? \_encryptedFilePath = null;

public string EncryptedFilePath

{

get => \_encryptedFilePath ?? "No file selected";

set => this.RaiseAndSetIfChanged(ref \_encryptedFilePath, value);

}

private string? \_encryptionKey = null;

public string EncryptionKey

{

get => \_encryptionKey ?? "";

set => this.RaiseAndSetIfChanged(ref \_encryptionKey, value);

}

private string? \_decryptionKey = null;

public string DecryptionKey

{

get => \_decryptionKey ?? "";

set => this.RaiseAndSetIfChanged(ref \_decryptionKey, value);

}

public ReactiveCommand<Unit, Unit> EncryptCommand { get; }

public ReactiveCommand<Unit, Unit> DecryptCommand { get; }

public ReactiveCommand<Unit, Unit> GenerateKeyCommand { get; }

public ReactiveCommand<Unit, Unit> ClearFilePath { get; }

public ReactiveCommand<Unit, Unit> ClearEncryptedFilePath { get; }

public ReactiveCommand<Unit, Unit> ClearEncryptionKey { get; }

public ReactiveCommand<Unit, Unit> ClearDecryptionKey { get; }

public ReactiveCommand<Unit, Unit> OpenFileDialogCommand { get; }

public ReactiveCommand<Unit, Unit> OpenEncryptedFileDialogCommand { get; }

public SM4ViewModel()

{

EncryptCommand = ReactiveCommand.CreateFromTask(async () =>

{

if (\_filePath is null || \_encryptionKey is null)

{

await GetMsBox(

Messages.Messages.ENCRYPTION\_FAIL,

Messages.Messages.ENCRYPT\_INPUT\_ERROR,

true

).ShowAsync();

return;

}

if (!SM4KeyRegex().Match(EncryptionKey.ToUpper()).Success || EncryptionKey.Length != 16)

{

await GetMsBox(

Messages.Messages.ENCRYPTION\_FAIL,

Messages.Messages.SM4\_KEY\_ERROR,

true

).ShowAsync();

return;

}

var timer = Stopwatch.StartNew();

var encryptedData = SM4.Encrypt(File.ReadAllBytes(FilePath), EncryptionKey);

FileWriter.FileWriter.WriteData(FileWriter.FileWriter.GetSM4EncryptedFilepath(FilePath)!, encryptedData!);

timer.Stop();

FilePath = EncryptionKey = null!;

await GetMsBox(

Messages.Messages.ENCRYPTION\_SUCCESS,

$"File was encrypted in {timer.ElapsedMilliseconds} ms and saved with original file",

false

).ShowAsync();

});

DecryptCommand = ReactiveCommand.CreateFromTask(async () =>

{

if (\_encryptedFilePath is null || \_decryptionKey is null)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

Messages.Messages.DECRYPT\_INPUT\_ERROR,

true

).ShowAsync();

return;

}

if (!SM4KeyRegex().Match(DecryptionKey.ToUpper()).Success || DecryptionKey.Length != 16)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

Messages.Messages.SM4\_KEY\_ERROR,

true

).ShowAsync();

return;

}

var timer = Stopwatch.StartNew();

var decryptedData = SM4.Decrypt(File.ReadAllBytes(EncryptedFilePath), DecryptionKey);

if (decryptedData == null)

{

await GetMsBox(

Messages.Messages.DECRYPTION\_FAIL,

"Wrong decryption key. Check your input and try again",

true

).ShowAsync();

return;

}

FileWriter.FileWriter.WriteData(FileWriter.FileWriter.GetSM4DecryptedFilepath(EncryptedFilePath)!, decryptedData);

timer.Stop();

EncryptedFilePath = DecryptionKey = null!;

await GetMsBox(

Messages.Messages.DECRYPTION\_SUCCESS,

$"File was decrypted in {timer.ElapsedMilliseconds} ms and saved with original file",

false

).ShowAsync();

});

GenerateKeyCommand = ReactiveCommand.Create(() =>

{

EncryptionKey = KeyGenerator.GenerateKey(16);

});

ClearFilePath = ReactiveCommand.Create(() =>

{

FilePath = null!;

});

ClearEncryptedFilePath = ReactiveCommand.Create(() =>

{

EncryptedFilePath = null!;

});

ClearEncryptionKey = ReactiveCommand.Create(() =>

{

EncryptionKey = null!;

});

ClearDecryptionKey = ReactiveCommand.Create(() =>

{

DecryptionKey = null!;

});

OpenFileDialogCommand = ReactiveCommand.CreateFromTask(async () =>

{

var desktop = (IClassicDesktopStyleApplicationLifetime)Application.Current?.ApplicationLifetime!;

var mainWindow = desktop.MainWindow;

var topLevel = TopLevel.GetTopLevel(mainWindow);

var files = await topLevel!.StorageProvider.OpenFilePickerAsync(new()

{

Title = "Select file to encrypt",

AllowMultiple = false

});

if (files.Count >= 1)

{

FilePath = files[0].Path.LocalPath;

}

});

OpenEncryptedFileDialogCommand = ReactiveCommand.CreateFromTask(async () =>

{

var desktop = (IClassicDesktopStyleApplicationLifetime)Application.Current?.ApplicationLifetime!;

var mainWindow = desktop.MainWindow;

var topLevel = TopLevel.GetTopLevel(mainWindow);

var files = await topLevel!.StorageProvider.OpenFilePickerAsync(new()

{

Title = "Select file to decrypt",

AllowMultiple = false

});

if (files.Count >= 1)

{

if (Path.GetExtension(files[0].Path.LocalPath) == ".sm4")

{

EncryptedFilePath = files[0].Path.LocalPath;

return;

}

var \_ = await GetMsBox("File extension error", Messages.Messages.FILE\_DECRYPTION\_FORMAT\_ERROR\_SM4, true).ShowAsync();

}

});

}

private static IMsBox<ButtonResult> GetMsBox(string header, string text, bool isError)

{

Bitmap logo = new(AssetLoader.Open(new Uri($"avares://{Assembly.GetExecutingAssembly().GetName().Name}/Assets/avalonia-logo.ico")));

return MessageBoxManager.GetMessageBoxStandard

(

new MsBox.Avalonia.Dto.MessageBoxStandardParams

{

ButtonDefinitions = ButtonEnum.Ok,

ContentTitle = "ThreeFish/SM4 Encryptor",

ContentHeader = header,

ContentMessage = text,

Icon = isError ? Icon.Error : Icon.Success,

WindowStartupLocation = WindowStartupLocation.CenterOwner,

CanResize = false,

Width = 400,

SizeToContent = SizeToContent.Height,

ShowInCenter = true,

Topmost = true,

WindowIcon = new WindowIcon(logo)

}

);

}

[GeneratedRegex(@"[0-9A-Za-z@#!`'""\\|/?$[\]{};:%^&\*()=\_+.,~<>-]")]

private static partial Regex SM4KeyRegex();

}

}

**Листинг файла MainWindowViewModel.cs**

namespace CourseProject.ViewModels;

public partial class MainWindowViewModel : ViewModelBase

{

public MainWindowViewModel()

{

ThreeFishViewModel = new();

SM4ViewModel = new();

AboutViewModel = new();

}

public ThreeFishViewModel ThreeFishViewModel { get; }

public SM4ViewModel SM4ViewModel { get; }

public AboutViewModel AboutViewModel { get; }

}

# ПРИЛОЖЕНИЕ Д

**Листинг файла Messages.cs**

namespace CourseProject.Messages

{

public static class Messages

{

public const string TF\_KEY\_ERROR = """

Three fish key must have size 32 characters

Allowed characters: 0-9A-Za-z@#!`'"\|/?$;:%^&\*()-=\_+.,~<>[]{}

Click button below the text box to generate strong key

""";

public const string SM4\_KEY\_ERROR = """

SM4 key must have size 16 characters

Allowed characters: 0-9A-Za-z@#!`'"\|/?$;:%^&\*()-=\_+.,~<>[]{}

Click button below the text box to generate strong key

""";

public const string ENCRYPT\_INPUT\_ERROR = "Choose file to encrypt and enter encryption key";

public const string DECRYPT\_INPUT\_ERROR = "Choose file to decrypt and enter decryption key";

public const string FILE\_IS\_NOT\_FOUND = "File is not found. Choose the file again";

public const string FILE\_DECRYPTION\_FORMAT\_ERROR\_TF = "Decryption file must have \"tfish\" extension";

public const string FILE\_DECRYPTION\_FORMAT\_ERROR\_SM4 = "Decryption file must have \"sm4\" extension";

public const string ENCRYPTION\_SUCCESS = "File was successfully encrypted";

public const string ENCRYPTION\_FAIL = "File was not encrypted";

public const string DECRYPTION\_SUCCESS = "File was successfully decrypted";

public const string DECRYPTION\_FAIL = "File was not decrypted";

}

}

**Листинг файла FileWriter.cs**

using System.IO;

namespace CourseProject.FileWriter

{

public class FileWriter

{

public static void WriteData(string filepath, byte[] data)

{

using var stream = new StreamWriter(filepath, false);

stream.BaseStream.Write(data, 0, data.Length);

}

public static string? GetThreeFishEncryptedFilepath(string filepath)

{

if (!File.Exists(filepath))

{

return null;

}

var fileinfo = new FileInfo(filepath);

var filename = fileinfo.Name + ".tfish";

return fileinfo.DirectoryName + "\\" + filename;

}

public static string? GetThreeFishDecryptedFilepath(string filepath)

{

if (!File.Exists(filepath))

{

return null;

}

var file = Path.GetFileNameWithoutExtension(filepath);

var fileinfo = new FileInfo(filepath);

return fileinfo.DirectoryName + "\\" + Path.GetFileNameWithoutExtension(file) + "\_tfish\_decrypted" + file[file.LastIndexOf('.')..];

}

public static string? GetSM4EncryptedFilepath(string filepath)

{

if (!File.Exists(filepath))

{

return null;

}

var fileinfo = new FileInfo(filepath);

var filename = fileinfo.Name + ".sm4";

return fileinfo.DirectoryName + "\\" + filename;

}

public static string? GetSM4DecryptedFilepath(string filepath)

{

if (!File.Exists(filepath))

{

return null;

}

var file = Path.GetFileNameWithoutExtension(filepath);

var fileinfo = new FileInfo(filepath);

return fileinfo.DirectoryName + "\\" + Path.GetFileNameWithoutExtension(file) + "\_sm4\_decrypted" + file[file.LastIndexOf('.')..];

}

}

}

**Листинг файла SM4.cs**

using Org.BouncyCastle.Crypto.Engines;

using Org.BouncyCastle.Crypto.Modes;

using Org.BouncyCastle.Crypto.Paddings;

using Org.BouncyCastle.Crypto.Parameters;

using System;

using System.Text;

namespace CourseProject.Crypto

{

public class SM4

{

private static readonly SM4Engine engine = new();

private static readonly PaddedBufferedBlockCipher engineMode = new(new EcbBlockCipher(engine), new Pkcs7Padding());

public static byte[]? Encrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(true, new KeyParameter(keyBytes));

return Process(data);

}

public static byte[]? Decrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(false, new KeyParameter(keyBytes));

return Process(data);

}

private static byte[]? Process(byte[] data)

{

try

{

int outputSize = engineMode.GetOutputSize(data.Length);

byte[] processedData = new byte[outputSize];

int processResutl = engineMode.ProcessBytes(data, 0, data.Length, processedData, 0);

engineMode.DoFinal(processedData, processResutl);

return processedData;

}

catch (Exception)

{

return null;

}

}

}

}

**Листинг файла ThreeFish.cs**

using Org.BouncyCastle.Crypto.Engines;

using Org.BouncyCastle.Crypto.Modes;

using Org.BouncyCastle.Crypto.Paddings;

using Org.BouncyCastle.Crypto.Parameters;

using System;

using System.Text;

namespace CourseProject.Crypto

{

public class ThreeFish

{

private static readonly ThreefishEngine engine = new(256);

private static readonly PaddedBufferedBlockCipher engineMode = new(new EcbBlockCipher(engine), new Pkcs7Padding());

public static byte[]? Encrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(true, new KeyParameter(keyBytes));

return Process(data);

}

public static byte[]? Decrypt(byte[] data, string key)

{

var keyBytes = Encoding.UTF8.GetBytes(key);

engineMode.Init(false, new KeyParameter(keyBytes));

return Process(data);

}

private static byte[]? Process(byte[] data)

{

try

{

int outputSize = engineMode.GetOutputSize(data.Length);

byte[] processedData = new byte[outputSize];

int processResutl = engineMode.ProcessBytes(data, 0, data.Length, processedData, 0);

engineMode.DoFinal(processedData, processResutl);

return processedData;

}

catch (Exception)

{

return null;

}

}

}

}

**Листинг файла KeyGenerator.cs**

using System;

using System.Linq;

namespace CourseProject.Crypto

{

public class KeyGenerator

{

private static readonly string AlphabetLowerCase = "abcdefghijklmnopqrstuvwxyz";

private static readonly string AlphabetUpperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private static readonly string Numbers = "0123456789";

private static readonly string SpecialChars = "@#!`'\"\\|/?$;:%^&\*()-=\_+.,~<>[]{}";

private static readonly string AllChars = AlphabetLowerCase + AlphabetUpperCase + Numbers + SpecialChars;

public static string GenerateKey(int keyLength)

{

Random random = new();

return string.Concat(

Enumerable.Repeat(0, keyLength).Select(c => AllChars[random.Next(AllChars.Length)])

);

}

}

}

**Листинг файла Button.axaml**

<Styles xmlns="https://github.com/avaloniaui"

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

<Design.PreviewWith>

<Border Padding="100">

<Button Classes="Generate"

Padding="50, 10">Hello, World</Button>

</Border>

</Design.PreviewWith>

<Style Selector="Button.Minimal">

<Setter Property="CornerRadius" Value="10" />

<Setter Property="Foreground" Value="Black" />

<Setter Property="Background" Value="White" />

<Setter Property="BorderBrush" Value="Black" />

<Setter Property="BorderThickness" Value="1" />

<Setter Property="FontWeight" Value="Medium" />

<Style Selector="^ /template/ ContentPresenter">

<Setter Property="Transitions">

<Transitions>

<BrushTransition Property="Background"

Duration="0:0:0.1"

Easing="CircularEaseInOut"/>

</Transitions>

</Setter>

</Style>

<Style Selector="^:pointerover /template/ ContentPresenter#PART\_ContentPresenter">

<Setter Property="Background" Value="Black" />

<Setter Property="Foreground" Value="White" />

</Style>

<Style Selector="^:pressed /template/ ContentPresenter#PART\_ContentPresenter">

<Setter Property="Background" Value="Gray" />

</Style>

</Style>

<Style Selector="Button.Generate">

<Style Selector="^ /template/ ContentPresenter">

<Setter Property="Foreground" Value="Black" />

<Setter Property="Background" Value="White" />

<Setter Property="BorderBrush" Value="Black" />

<Setter Property="BorderThickness" Value="1" />

<Setter Property="FontWeight" Value="Medium" />

<Setter Property="Transitions">

<Transitions>

<BrushTransition Property="Background"

Duration="0:0:0.1"

Easing="CircularEaseInOut"/>

</Transitions>

</Setter>

</Style>

<Style Selector="^:pointerover /template/ ContentPresenter#PART\_ContentPresenter">

<Setter Property="Background" Value="#f1f1f1" />

<Setter Property="BorderBrush" Value="Black" />

<Setter Property="BorderThickness" Value="1" />

</Style>

<Style Selector="^:pressed /template/ ContentPresenter#PART\_ContentPresenter">

<Setter Property="Background" Value="#d5d5d5" />

</Style>

</Style>

</Styles>

**Листинг файла TextBox.cs**

<Styles xmlns="https://github.com/avaloniaui"

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

<Design.PreviewWith>

<Border Padding="100">

<TextBox Width="200" Classes="Minimal">123</TextBox>

</Border>

</Design.PreviewWith>

<Style Selector="TextBox.Minimal">

<Setter Property="FontWeight" Value="Medium" />

<Setter Property="Background" Value="White" />

<Setter Property="Foreground" Value="Black" />

<Setter Property="Height" Value="40" />

<Setter Property="VerticalContentAlignment" Value="Center" />

<Setter Property="ScrollViewer.HorizontalScrollBarVisibility" Value="Hidden" />

<Setter Property="AcceptsReturn" Value="True" />

<Setter Property="InnerLeftContent">

<Template>

<Image Source="/Assets/key.ico"

Width="25"

Margin="5,0,0,0"/>

</Template>

</Setter>

<Style Selector="^ /template/ Border">

<Setter Property="BorderBrush" Value="Black" />

</Style>

<Style Selector="^:focus /template/ Border#Part\_BorderElement">

<Setter Property="BorderThickness" Value="2" />

<Setter Property="BorderBrush" Value="Black" />

</Style>

</Style>

</Styles>