МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 01 02 03 Информационные системы

и технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Защита информации и надежность информационных систем»

Тема: «Сравнительный анализ криптостойкости блочных алгоритмов шифрования Blowfish, Twofish и Threefish»

Исполнитель

Студент 3 курса группы 1 Потапейко П.А.

(Ф.И.О.)

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

Курсовой проект защищен с оценкой

Председатель Берников В.О.

(подпись)

Минск 2022

**Оглавление**

[Введение 3](#_Toc103018757)

[1. Аналитический обзор литературы 4](#_Toc103018758)

[1.1. Описание алгоритма Blowfish 4](#_Toc103018759)

[1.2. Функция F(x) 5](#_Toc103018760)

[1.3. Описание алгоритма Twofish 6](#_Toc103018761)

[1.3.1. Генерация подключей 6](#_Toc103018762)

[1.3.2. Функция h 6](#_Toc103018763)

[1.3.3. Функция g и S-блоки 7](#_Toc103018764)

[1.3.4. Криптопреобразование Адамара 9](#_Toc103018765)

[1.3.5. Отбеливание 9](#_Toc103018766)

[1.4. Описание алгоритма Threefish 9](#_Toc103018767)

[3. Сравнительный анализ алгоритмов шифрования Blowfish, Twofish, Threefish 11](#_Toc103018768)

[4. Проектирование программного средства 13](#_Toc103018769)

[4.1. Blowfish 14](#_Toc103018770)

[4.2. Twofish 19](#_Toc103018771)

[4.3. Threefish 21](#_Toc103018772)

[5. Тестирование программного средства 25](#_Toc103018773)

[6. Руководство пользователя 27](#_Toc103018774)

[Заключение 32](#_Toc103018775)

[Список источников 33](#_Toc103018776)

[Приложение А 34](#_Toc103018777)

[Приложение Б 53](#_Toc103018778)

[Приложение B 56](#_Toc103018779)

# **Введение**

В настоящее время все существующие криптосистемы принято разделять на два класса: симметричные и асимметричные. Соответственно, говорят о симметричной и асимметричной криптографии.

В симметричных криптосистемах одним и тем же секретным ключом осуществляется и шифрование, и расшифрование.

Существует также разделение всех симметричных шифров на блочные (обрабатывают группы битов (блоки) открытого текста) и потоковые (обрабатывают открытый текст побитово).

Наиболее широко применяемыми на практике симметричными шифрами долгое время являлись шифры DES, IDEA, ГОСТ и их модификации.

Со временем были выявлены недостатки данных алгоритмов, на смену им пришли новые, более криптостойкие, призванные заменить используемые алгоритмы, среди которых блочный алгоритм шифрования Blowfish, а также его модификации Twofish и Threefish, созданные Брюсом Шнайером – известным специалистом по криптографии.

Алгоритм Blowfish был создан в 1993 году, превосходит DES по скорости и стойкости, особенностью которого является также то, что он непатентованный и бесплатный, используется во многих коммерческих приложениях.

Модификацией Blowfish стал алгоритм Twofish, разработанный в 1998 году группой специалистов во главе со Шнайером. Но высокая сложность структуры алгоритма и сложность его анализа, а также достаточно медленное время выполнения на большинстве платформ не привело к широкому распространению алгоритма, хотя все еще многие программы шифрования его используют. Как и алгоритм Blowfish является незапатентованным и бесплатным.

В 2008 году был представлен переработанный и усовершенствованный алгоритм Threefish, призванный стать универсальной заменой существующим блочным шифрам. Главными достоинствами названы минимальное использование памяти, устойчивость к атакам, простота реализации и оптимизация под 64-разрядные процессоры.

Целью курсовой работы является анализ надежности блочных алгоритмов шифрования Blowfish, Twofish и Threefish.

1. **Аналитический обзор литературы**

До появления Blowfish существовавшие алгоритмы были либо запатентованными, либо ненадёжными, а некоторые и вовсе держались в секрете. Алгоритм был разработан в 1993 году Брюсом Шнайером в качестве быстрой и свободной альтернативы устаревшему DES и запатентованному IDEA. По заявлению автору он хотел создать быстрый, простой, компактный и настраиваемый алгоритм шифрования.

Скорость шифрования получилась очень хорошая: всего 26 тактов на 32-битном процессоре. Такая скорость шифрования была достигнута благодаря простоте: использовались только простейшие математические операции.

Алгоритм получился крайне компактным: наименьшая реализация данного алгоритма могла поместиться в 5кб памяти.

В зависимости от имеющихся в распоряжении мощностей можно использовать ключ переменной длины от 32 до 448 бит.

Алгоритм Twofish является логическим развитием Blowfish, разработанный группой специалистов во главе со Шнайером.

Размер блока 128 бит, а размер ключа – до 256 бит.

Отличительными особенностями алгоритма являются использование предварительно вычисляемых и зависящих от ключа S-блоков и сложная схема развёртки подключей шифрования. Половина n-битного ключа шифрования используется как собственно ключ шифрования, другая — для модификации алгоритма (от неё зависят S-блоки).

Алгоритм Threefish задумывался как усовершенствованная замена всех существующих блочных алгоритмов шифрования.

**1.1. Описание алгоритма Blowfish**

Алгоритм Blowfish условно можно разделить на две части: подготовительный (расширение ключа) и само шифрование.

Подготовительный этап представляет собой формирование ключей шифрования по секретному ключу. Для работы алгоритма необходимо сгенерировать 18 32-битных чисел Pi и четыре массива Si 32-битных чисел размерностью 256 элементов.

Изначально P1..18 состоит из мантиссы числа пи. Обозначим ключ буквой K, а K1 – первые 32 бита ключа, K2 – вторые 32 бита ключа. Если, к примеру ключ длиной 64 бита, то K3 будет равен K4, то есть если ключ недостаточной длины, то он циклически повторяется.

Далее в Pi записывается результат операции XOR над Pi и Ki. Затем используя сгенерированные P зашифровывается 64-битный нулевой блок. P1 приравнивается к левой половине зашифрованного блока, а P2 – к правой половине. Далее зашифровывается предыдущий зашифрованный блок и P3 приравнивается к левой половине зашифрованного блока, а P4 – к правой. Данная операция происходит 9 раз, т.е. до полной замены P1...18, а затем по такому же принципу заполняются четыре таблицы замен Si.

После этих операций можно считать, что подготовительный этап завершен. Подготовительный этап достаточно продолжительный по сравнению с самим шифрованием и расшифрованием.

Процесс зашифрования состоит из 16 раундов. Данный алгоритм шифрует 64-битные блоки. Входной блок делится на две 32-битных половины L0 и R0. Примем i – номер текущего ранда, тогда Li = Li-1 XOR Pi, а Ri = Ri-1 XOR F(Li), где F – функция Фейстеля. После этого Li и Ri меняются местами, и данная операция повторяется 16 раз. После 16 раунда L17 = L16 XOR P18 и R17 = R16 XOR P17. L17 и R17 и являются зашифрованным блоком. Схема зашифрования представлена на рисунке 1.1.

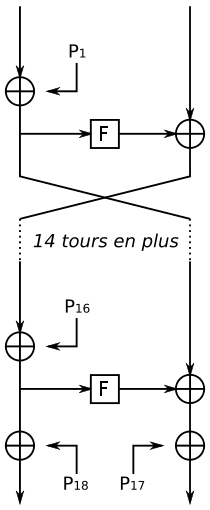


Рисунок 1.1 – Схема зашифрования Blowfish

Процесс дешифрования происходит аналогичным образом, только P1 – P18 применяются в обратном порядке.

В качестве начального состояния числа P можно использовать не только мантиссу числа пи, но и любое другое случайное число. Данный выбор заключается в инициализации последовательности, не связанной с алгоритмом, которая могла бы быть сохранена как часть алгоритма или получена при необходимости.

**1.2. Функция F(x)**

Функция F(x) принимает на вход блок размером в 32 бита, где делит данный блок на четыре 8-битных блока X1, X2, X3, X4, каждый из которых является индексом массива таблицы замен Si. Далее S1[X1] складывается с S2[X2] по модулю 232, затем результат складывается с S3[X3] по модулю 2, и затем результат складывается с S4[X4] по модулю 232. Результатом этих операций является значение F(x). Схема функции представлена на рисунке 1.2.

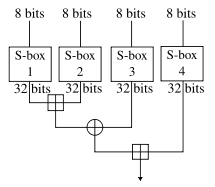


Рисунок 1.2 – Схема функции F(x)

**1.3. Описание алгоритма Twofish**

### **1.3.1. Генерация подключей**

Twofish рассчитан на работу с ключами длиной 128, 192 и 256 бит. Из исходного ключа генерируется 40 32-битных подключей, первые восемь из которых используются только в операциях входного и выходного отбеливания, а остальные 32 – в раундах шифрования, по два подключа на раунд. Особенностью Twofish является то, что исходный ключ используется также и для изменения самого алгоритма шифрования, так как используемые в функции g S-блоки не фиксированы, а зависят от ключа.

Для формирования раундовых подключей исходный ключ M разбивается с перестановкой байт на два одинаковых блока Mo и Me. Затем с помощью блока Mo и функции h шифруется значение 2\*i, а с помощью блока Me шифруется значение 2\*i+1, где i — номер текущего раунда (0 – 15). Полученные зашифрованные блоки смешиваются криптопреобразованием Адамара, и затем используются в качестве раундовых подключей.

### **1.3.2. Функция h**

Как для формирования подключей, так и для формирования функции g в Twofish используется одна основная функция: h(X, L0, L1, … , Lk). Здесь X, L0, L1, …, Lk — 32 битные слова, а k = N / 64, где N — длина исходного ключа в битах. Результатом функции является одно 32-битное слово.

Функция выполняется в k этапов. То есть для длины ключа 256 бит будет 4 этапа, для ключа в 192 бита — 3 этапа, для 128 бит — 2 этапа.

На каждом этапе входное 32-битное слово разбивается на 4 байта, и каждый байт подвергается фиксированной перестановке битов q0 или q1. Входной байт x разбивается на две 4-битные половины a0 и b0, над которыми проводятся следующие преобразования:

* a0 = x/16; b0 = x mod 16
* a1 = a0 XOR b0; b1 = a0 XOR ROR4(b0, 1) XOR 8a0 mod 16
* a2 = t0[a1]; b2 = t1[b1]
* a3 = a2 XOR b2; b3 = a2 XOR ROR4(b2, 1) XOR 8a2 mod 16
* a4 = t2[a3]; b4 = t3[b3]
* y = 16b4 + a4

Здесь ROR4 ­­­­­­­­­­– 4-битный циклический сдвиг вправо, а t0, t1, t2, t3 – табличные замены четырехбитных чисел.

На рисунке 1.3 представлены схемы функции h для различных длин ключа.

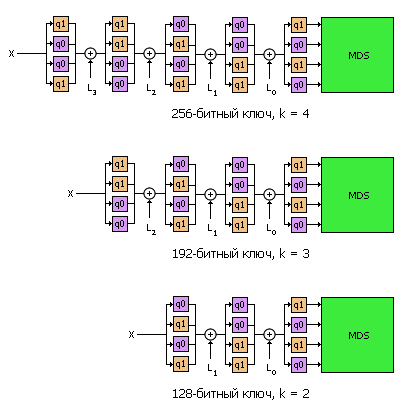


Рисунок 1.3 – Функция h для разных длин ключа

### **1.3.3. Функция g и S-блоки**

Функция g – основа алгоритма Twofish. На вход функции подается 32-битное число X, которое затем разбивается на четыре байта x0, x1, x2, x3. Каждый из получившихся байтов пропускается через свой S-блок.

Получившиеся 4 байта на выходах S-блоков интерпретируются как вектор с четырьмя компонентами. Результат умножается на фиксированную MDS матрицу размером 4x4, причем вычисления проводятся в [поле Галуа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) GF(28) по модулю неприводимого многочлена x8+x6+x5+x3+1.

На рисунке 1.4 представлена схема одного раунда шифрования алгоритма Twofish для 128-битного ключа.

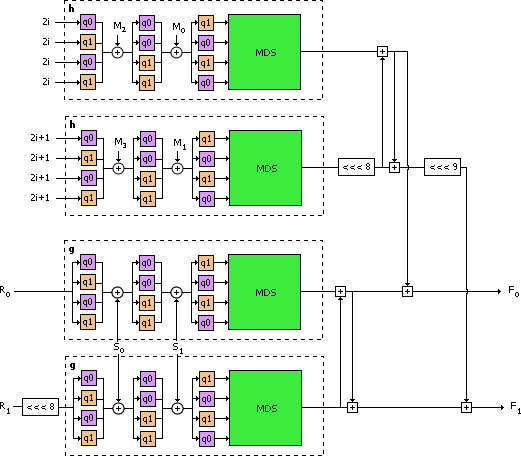


Рисунок 1.4 – Схема одного раунда шифрования Twofish

MDS матрица – это такая матрица над конечным полем K, что если взять её в качестве матрицы линейного преобразования f(x) = (MDS)x из пространства Kn в пространство Km, то любые два вектора из пространства Kn+m вида (x, f(x)) будут иметь как минимум m+1 различий в компонентах. То есть набор векторов вида (x, f(x)) образует код, обладающий свойством максимальной разнесённости. То есть, для алгоритма Twofish это значит, что общее количество меняющихся байт вектора a и вектора b = (MDS)a не меньше пяти. Любое изменение только одного байта в a приводит к изменению всех четырех байтов в b.

Вид матрицы MDS представлен на рисунке 1.5.

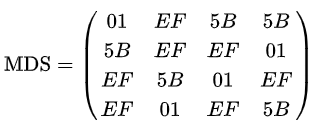


Рисунок 1.5 – Матрица MDS

Вектор S, как и векторы Me и Mo, тоже формируется из исходного ключа и состоит из k 32-битных слов. Исходные байты ключа разбиваются на k групп по восемь байтов. Каждая такая группа рассматривается как вектор с 8 компонентами и умножается на фиксированную RS-матрицу размером 4×8 байтов. В результате умножения получается вектор, состоящий из четырёх байтов.

Вид матрицы RS представлен на рисунке 1.6.

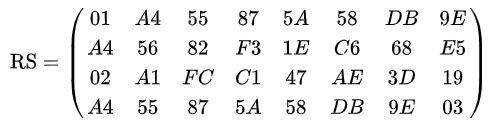


Рисунок 1.6 – Матрица RS

### **1.3.4. Криптопреобразование Адамара**

Криптопреобразование Адамара –  обратимое преобразование битовой строки длиной 2n. Строка разбивается на две части a и b одинаковой длины в n бит. Преобразование вычисляется следующим образом: a’ = a + b; b’ = a + 2b.

Эта операция часто используется для «рассеивания» кода.

Данное преобразование в алгоритме Twofish используется при смешивании результатов двух g-функций (n = 32).

### **1.3.5. Отбеливание**

Отбеливание – это процедура сложения по модулю два данных с подключами перед первым раундом и после последнего раунда.

Исследования применения отбеливания показали, что данная процедура усложняет задачу поиска ключа полным перебором. Разработчики Twofish утверждают, что в Twofish отбеливание также существенно усложняет задачу подбора ключа, поскольку криптоаналитик не может узнать, какие данные попадают на вход функции F на первом раунде.

**1.4. Описание алгоритма Threefish**

Алгоритм Threefish имеет очень простую структуру, является быстрым и гибким шифром, работающим в произвольном режиме шифрования.

Алгоритм не использует S-блоки, основан на комбинировании инструкций исключающего или, сложения и циклического сдвига.

Предусмотрено 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.

Алгоритм имеет нетрадиционно большое число раундов – 72 или 80 при ключе 1024 бит. Размер блока данных – 256, 512 или 1024 бит. Длина ключа равна выбранному размеру блока. Размер tweak-значения для любого из размеров блока – 128 бит.

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

Функция шифрования E(K, T, P), где:

* K – ключ шифрования, представляющий собой строку 256, 512 или 1024 бит;
* T – tweak-значение, строка длиной 128 бит;
* P – открытый текст для шифрования, строка длиной, равной размеру блока.

Количество раундов Nr определяется следующим образом:

* для длины ключа 256 бит Nr равно 72;
* для длины ключа 512 бит Nr равно 72;
* для длины ключа 1024 бит Nr равно 80.

Алгоритм использует Nr/4+1 раундовых ключей.

Функция MIX является нелинейной функцией смешивания и принимает на вход два аргумента (x0, x1) и возвращает (y0, y1):

* y0 = (x0 + x1) mod 264
* y1 = (x1 <<< R(d mod 8), j) XOR y0, где <<< – оператор побитового циклического сдвига влево, а константа Rd,j определяется из специальной таблицы

Каждый раунд зашифрования состоит из нескольких этапов. На первом этапе раунда к внутреннему состоянию добавляется раундовый ключ ks,i, на следующем этапе используется нелинейная функция MIXd,j. После всех раундов выходом алгоритма является шифротекст.

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.

# **3. Сравнительный анализ алгоритмов шифрования Blowfish, Twofish, Threefish**

Для сравнительного анализа криптостойкости указанных алгоритмов шифрования определим следующие критерии, по которым и будет производиться сравнение:

* криптостойкость;
* переменность размера ключа;
* наличие S-блоков, зависимых от ключа;
* наличие tweak-значения;
* переменность размера блока данных;
* отсутствие слабых ключей;
* простота структуры алгоритма.

Если критерий не может быть реализован в определенном алгоритме, то значение критерия приравнивается к 0, если критерий реализуется в полной мере, то значение принимается равным 1, если критерий реализуется с ограничениями, то значение критерия принимается равным 0,5.

В таблице 3.1 приведен результат сравнительного анализа указанных алгоритмов шифрования.

Таблица 3.1 – Сравнительный анализ алгоритмов шифрования Blowfish, Twofish, Threefish

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Blowfish | Twofish | Threefish |
| Криптостойкость | 0,5 | 1 | 1 |
| Переменность размера ключа | 1 | 1 | 1 |
| Наличие S-блоков, зависимых от ключа | 1 | 1 | 0 |
| Наличие tweak-значения | 0 | 0 | 1 |
| Переменность размера блока данных | 0 | 0 | 1 |
| Отсутствие слабых ключей | 0 | 1 | 1 |
| Простота структуры алгоритма | 1 | 0 | 1 |
| Результат | 3,5 | 4 | 6 |

На основе анализа информационных ресурсов был сделан вывод, что криптостойкость алгоритма Blowfish в значительной степени зависит от вероятности появления слабых ключей, количества раундов шифрования, количества используемых S-блоков. Малый размер блока данных также делает алгоритм уязвимым для атаки дней рождения, а вариант Blowfish с уменьшенным количеством раундов уязвим для атак на основе открытых текстов на слабых ключах. Таким образом значение криптостойкости принимается равным 0,5.

Для алгоритма Twofish было проведено исследование на основе усеченного дифференциального криптоанализа, однако исследования носили сугубо теоретический характер, так как самой атаки не было проведено, также сам создатель алгоритма опроверг саму возможность проведения такой атаки в реальности. Таким образом, значение криптостойкости алгоритма Twofish принимается равным 1.

Алгоритм Threefish имеет большее количество раундов шифрования, что сильно повышает его криптостойкость по сравнению с другими алгоритмами, также большой размер ключа делает алгоритм неуязвимым к атаке грубой силой на современном оборудовании. Таким образом, значение криптостойкости алгоритма Threefish принимается равным 1.

Размеры ключей всех трех алгоритмов могут варьироваться, значение соответствующего критерия для всех алгоритмов принимается равным 1.

Алгоритмы Blowfish и Twofish используют S-блоки, инициализируемые исходным ключом, в то время как алгоритм Threefish S-блоки вообще не использует. Для Blowfish и Twofish значение соответствующего критерия принимается равным 1, для Threefish – 0.

Tweak-значение, используемое только в алгоритме Threefish среди рассматриваемых алгоритмов позволяет изменять значение выхода, без изменения ключа, что значительно повышает криптостойкость алгоритма. Значение критерия наличия tweak-значения принимается равным 1. Алгоритмы Blowfish и Twofish tweak-значение не используют. Значение критерия наличия tweak-значения принимается равным 0.

У алгоритма Blowfish размер блока данных является относительно малым для современных блочных алгоритмов – 64 бита, у алгоритма Twofish – 128 бит. У алгоритма Threefish размер блока данных равен выбранному размеру ключа (256, 512 или 1024 бит). Для алгоритма Blowfish и Twofish значение критерия переменности размера блока данных принимается равным 0, для Threefish – 1.

На основе проанализированных информационных ресурсов было выявлено, что только у алгоритма Blowfish существует проблема слабых ключей, поэтому значение критерия отсутствия слабых ключей для алгоритмов Twofish и Threefish принимается равным 1, для Blowfish – 0.

Структура алгоритма Twofish является очень сложной за счет усложнения процедуры генерации подключей, что в большой степени усложняет процесс криптоанализа, поэтому значение соответствующего критерия устанавливается равным 0. Алгоритм Blowfish является сетью Фейстеля с использованием простых операций сложения, подстановки и XOR. Структура алгоритма Threefish является очень простой, за счет функции MIX, являющейся ядром шифра. Значения критерия простоты структуры алгоритмов для Blowfish и Threefish принимается равным 1.

По результатам сложения всех значений критериев получаем для Blowfish 3,5, для Twofish 4 и для Threefish 6. Как можно видеть, алгоритм Threefish является наиболее стойким ко взлому среди рассмотренных алгоритмов.

# **4. Проектирование программного средства**

Программное средство для визуализации работы блочных алгоритмов шифрования Blowfish, Twofish и Threefish, реализовано на объектно-ориентированном языке программирования C# на платформе Microsoft .NET Framework версии 4.8 в среде разработки Microsoft Visual Studio 2022.

Для реализации программного средства использован API-интерфейс Windows Forms для создания настольных графических программ, имеющих понятный и интерактивный интерфейс.

Визуализация алгоритмов подразумевает просмотр последовательной работы алгоритмов шифрования и расшифрования, также оценивается время проведения данных операций.

Само приложение должно:

* принимать сообщение, введенное пользователем;
* шифровать и расшифровывать сообщение, введенное пользователем;
* оценивать время проведения операций щифрования и расшифрования.

Рассмотрим решение проекта «ZINISСourseProject», имеющее структуру, представленную на рисунке 4.1. Описание структуры проекта описано в таблице 4.1.

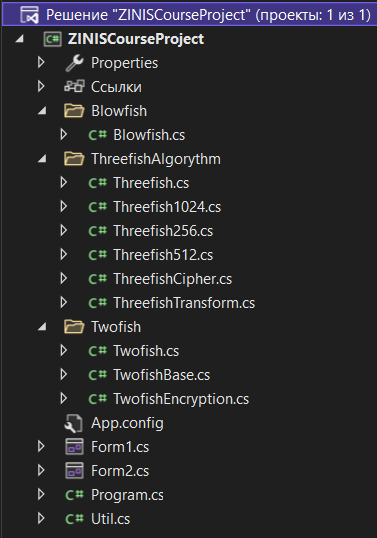


Рисунок 4.1 – Структура проекта

Таблица 4.1 – Описание структуры проекта

|  |  |
| --- | --- |
| Properties | Свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках |
| Ссылки | Ссылки на классы, используемые в проекте |
| Blowfish | Директория с классами, реализующими алгоритм Blowfish |
| ThreefishAlgorythm | Директория с классами, реализующими алгоритм Threefish |
| Twofish | Директория с классами, реализующими алгоритм Twofish |

Программная реализация всех функций представлена в приложении А.

**4.1. Blowfish**

На рисунке 4.2 показаны глобальные переменные и константы класса, реализующего алгоритм Blowfish.

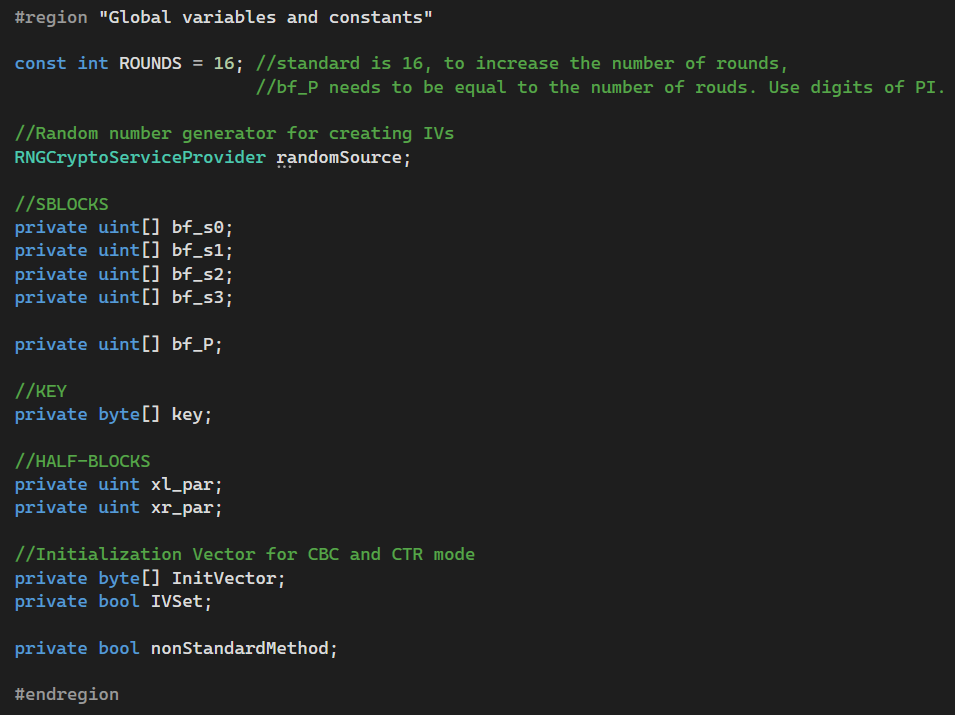


Рисунок 4.2 – Константы класса Blowfish

Константа ROUNDS представляет собой количество раундов, используемых в алгоритме.

Константа randomSouce представляет объект класса RNGCryptoServiceProvider, используется для генератора случайных чисел для создания вектора инициализации. Переменные bf\_s0, bf\_s1, bf\_s2, bf\_s3 реализуют S-блоки, переменная bf\_P реализует P-блок, переменная key хранит секретный ключ, переменные xl-par и xr-par используются для хранения левой и правой половин блока данных, переменные InitVector и IVSet нужны для хранения вектора инициализации, переменная nonStandardMethod нужна для хранения настройки нестандартного метода шифрования/расшифрования.

На рисунке 4.3 показаны конструкторы класса Blowfish.

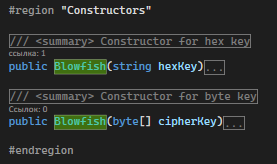


Рисунок 4.3 – Конструкторы класса Blowfish

Первый конструктор принимает в качестве параметра строку ключа в 16-ной системе исчисления, второй конструктор в качестве параметра принимает ключ в виде массива байтов.

На рисунке 4.4 показаны методы инициализации S-блоков и P-блока, нужные для работы алгоритма.

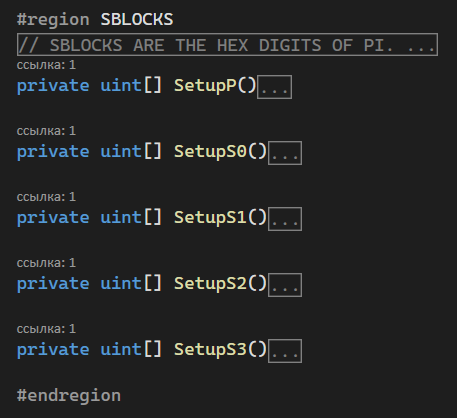


Рисунок 4.4 – Методы инициализации блоков

Метод SetupP заполняет массив P-блока, методы SetupS0, SetupS1, SetupS2, SetupS3 заполняют массивы четырех S-блоков.

На рисунке 4.5 показаны публичные методы задания режима шифрования алгоритмом Blowfish.

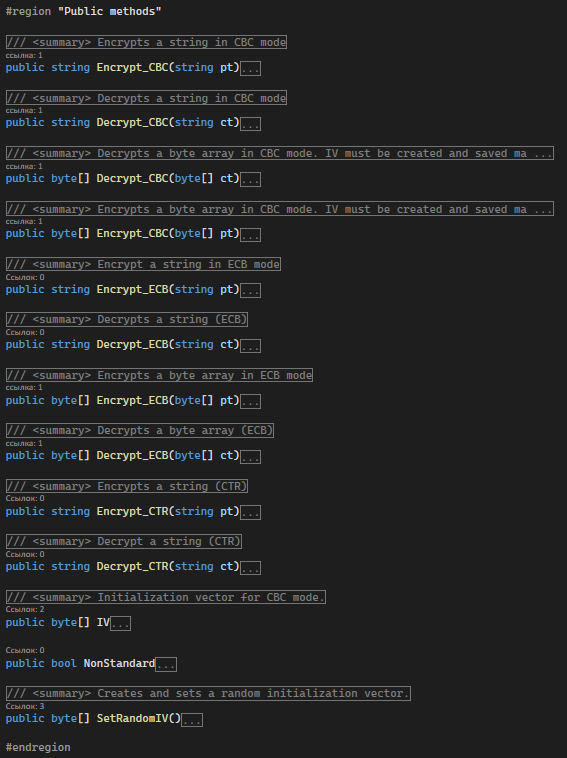


Рисунок 4.5 – Методы задания режима шифрования

Для режимов CBC и ECB определено по два метода установки режима, которые принимают в качестве параметра соответственно строку или массив байтов. Это методы Encrypt\_CBC, Decrypt\_CBC, Encrypt\_ECB, Decrypt\_ECB, для режима шифрования CTR определено два метода Encrypt\_CTR и Decrypt\_CTR, которые принимают в качестве параметра строку.

Свойство IV устанавливает и возвращает вектор инициализации, свойство NonStandard устанавливает и возвращает значение использования нестандартного метода шифрования/расшифрования. Метод SetRandomIV создает и устанавливает случайное значение вектора инициализации.

На рисунке 4.6 показаны основные методы, реализующие алгоритм Blowfish.

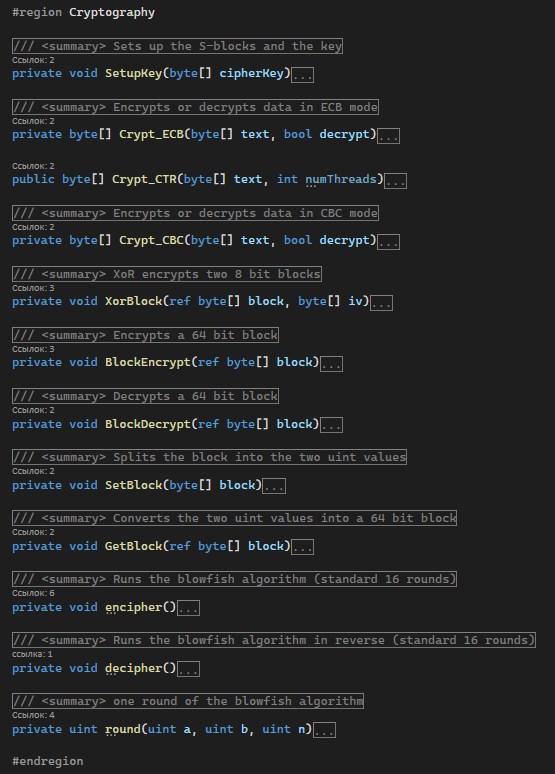


Рисунок 4.6 – Основные методы алгоритма Blowfish

Метод SetupKey заполняет S-блоки, методы Crypt\_ECB, Crypt\_CTR, Crypt\_CBC реализуют операцию зашифрования различными режимами работы алгоритма. Метод XorBlock реализует операцию XOR двух блоков данных.

Метод BlockEncrypt реализует операцию зашифрования 64-битного блока, метод BlockDecrypt в свою очередь реализует операцию расшифрования 64-битного блока данных. Метод SetBlock разделяет блок на два значения типа uint, метод GetBlock конвертирует две переменных типа uint в один 64-битный блок данных. Метод encipher запускает операции зашифрования, метод decipher запускает операцию расшифрования. Метод round представляет собой реализацию одного раунда зашифрования.

На рисунке 4.7 показаны вспомогательные методы конвертирования данных.

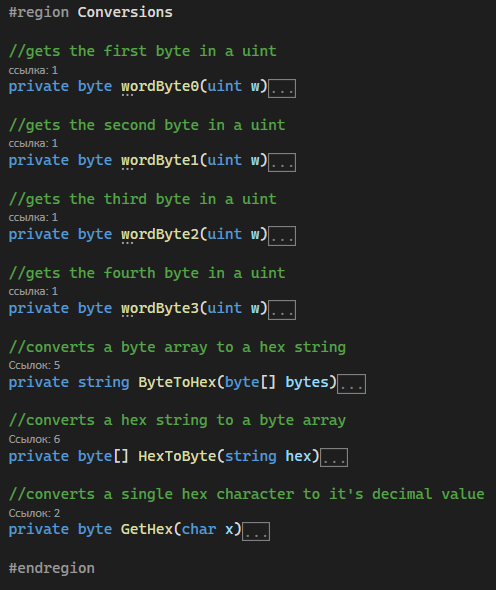


Рисунок 4.6 – Вспомогательные методы конвертирования

Методы wordByte0, wordByte1, wordByte2, wordByte3 нужны для получения первого, второго, третьего и четвертого байта соответственно в формате uint. Методы ByteToHex и HexToByte переводят массив байтов в формат строки в 16-ной системе исчисления и наоборот. Метод GetHex конвертирует символ ы 16-ной системе исчисления в формат decimal.

**4.2. Twofish**

Реализация алгоритма Twofish состоит из трех классов: TwofishBase, TwofishEncryption и Twofish. Рассмотрим структуру каждого класса в отдельности.

На рисунке 4.7 показаны функции класса TwofishBase.

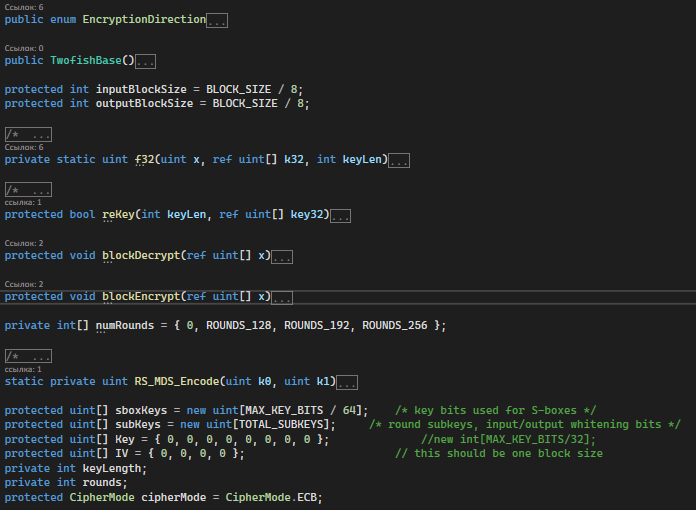


Рисунок 4.7 – Функции класса TwofishBase

Конструктор класса инициализирует объект класса, поля inputBlockSize и outputBlockSize хранят длину входного и выходного блока соответственно. Метод f32 реализует F-функцию. Метод reKey реализует генерацию раундовых ключей. Метод blockDecrypt и blockEncrypt реализуют функции расшифрования и шифрования блока данных соответственно, функция RS\_MDS\_Encode реализует инициализацию матрицы MDS. Переменная sboxKeys хранит биты ключа, нужные для заполнения S-блоков, переменная subKeys хранит раундовые ключи, а также биты этапов входного и выходного отбеливания. Переменная Key хранит значение секретного ключа, переменная IV хранит значение вектора инициализации, переменная keyLength хранит значение длины ключа, переменная rounds хранит значение количества раундов, переменная cipherMode хранит значение установленного режима шифрования, по умолчанию – ECB.

На рисунке 4.8 показаны методы класса TwofishEncryption.

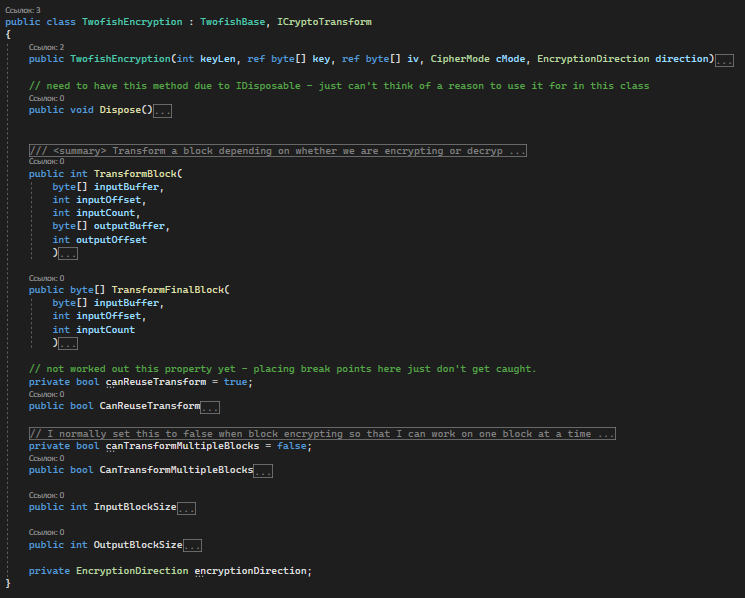


Рисунок 4.8 – Функции класса TwofishEncryption

Конструктор класса инициализирует объект класса, метод Dispose высвобождает ресурсы, метод TransformBlock выполняет трансформацию блока данных, метод TransformFinalBlock выполняет трансформацию последнего блока данных. Свойства CanReuseTransform и CanTransformMultipleBlocks задают и возвращают значения переменных настройки того, можно ли переиспользовать трансформацию и можно ли трансформировать несколько блоков одновременно соответственно. Свойства InputBlockSize и OutputBlockSize устанавливают и возвращают переменные, отвечающие за длину входного и выходного блока соответственно.

На рисунке 4.9 показаны методы класса TwofishEncryption.

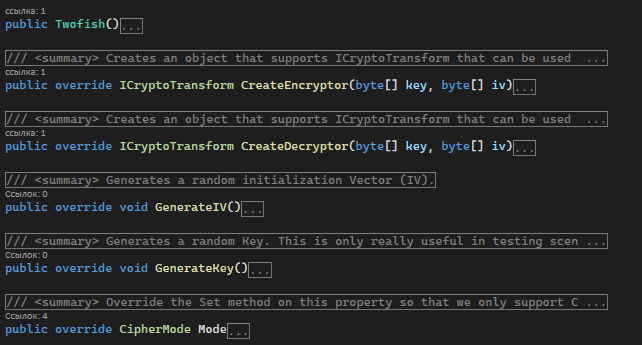


Рисунок 4.9 – Методы класса Twofish

Конструктор класса инициализирует объект класса, метод CreateEncryptor создает экземпляр класса TwofishEncryption и устанавливает режим шифрования. Метод CreateDecryptor также создает экземпляр класса TwofishEncryption и устанавливает режим расшифрования. Метод GenerateIV предназначен для генерации случайного вектора инициализации, метод GenerateKey реализует генерацию случайного секретного ключа, свойство Mode устанавливает режим шифрования.

**4.3. Threefish**

Реализация алгоритма Threefish состоит из шести классов: Threefish, ThreefishCipher, ThreefishTransform и классов, реализующих варианты алгоритма Threefish с различным размером ключа – Threefish256, Threefish512, Threefish1024. Рассмотрим структуру каждого класса в отдельности.

На рисунке 4.10 показаны переменные класса ThreefishCipher.

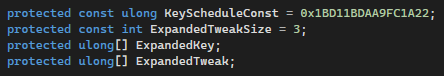


Рисунок 4.10 – Переменные класса ThreefishCipher

Константа KeyScheduleConst представляет расписание ключей, константа ExpandedTweakSize хранит размер tweak-значения, переменная ExpandedKey представляет массив байт расширенного ключа, переменная ExpandedTweak хранит tweak-значение.

На рисунке 4.11 показаны методы класса ThreefishCipher.

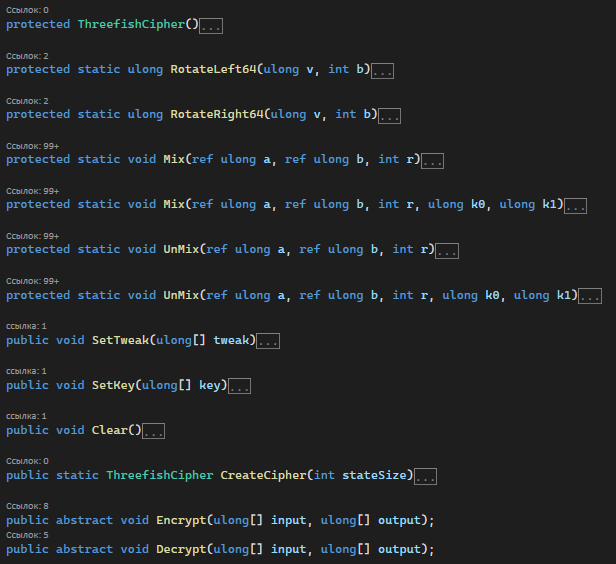


Рисунок 4.11 – Методы класса ThreefishCipher

Методы RotateLeft64 и RotateRight64 реализуют операцию сдвига влево и вправо соответственно. метод Mix имеет две реализации, отличающиеся параметрами, оба метода реализуют функцию Mix, метод UnMix также имеет две реализации и используется как реализация функции Mix для расшифрования, метод SetTweak устанавливает tweak-значение, метод SetKey устанавливает секретный ключ, метод Clear сбрасывает значения секретного ключа и tweak-значения, метод CreateCipher возвращает объект класса выбранного варианта алгоритма Threefish с длиной ключа 256, 512 или 1024 бит, методы Encrypt и Decrypt являются абстрактными и определяют методы шифрования и расшифрования соответственно.

На рисунке 4.12 показаны методы класса ThreefishTransform.

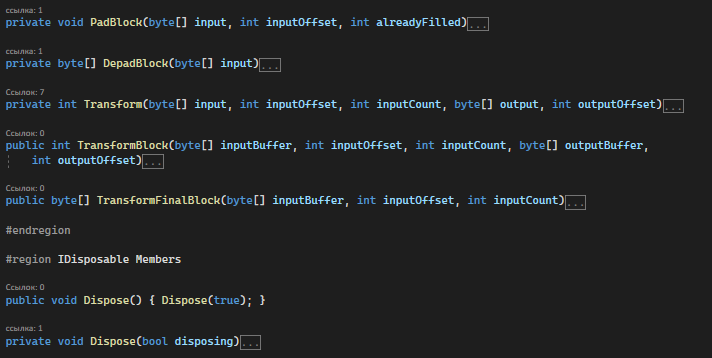


Рисунок 4.12 – Методы класса ThreefishTransform

Метод PadBlock определяет вид заполнения пустых битов в операции расширения блока, метод DepadBlock реализует обратную методу PadBlock процедуру, методы Transform, TransformBlock, TransformFinalBlock реализуют трансформации блоков.

На рисунке 4.13 показаны методы класса Threefish.

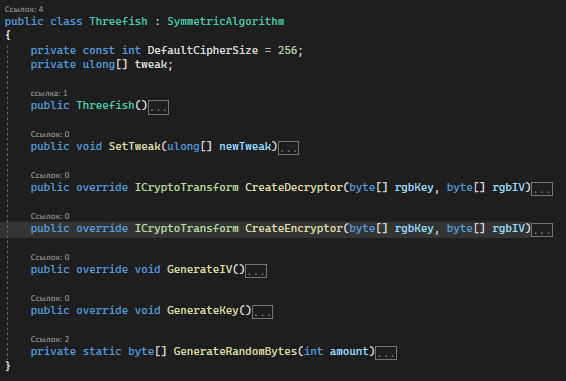


Рисунок 4.13 – Методы класса Threefish

Конструктор инициализирует объект класса, метод SetTweak устанавливает значение tweak-значения, метод CreateDecryptor создает экземпляр класса ThreefishTransform для расшифрования, метод CreateEncryptor создает экземпляр класса ThreefishTransform для шифрования, метод GenerateIV используется для генерации вектора инициализации, метод GenerateKey генерирует секретный ключ, метод GenerateRandomBytes генерирует случайную последовательность байтов.

На рисунке 4.14 показана реализация класса Threefish256.

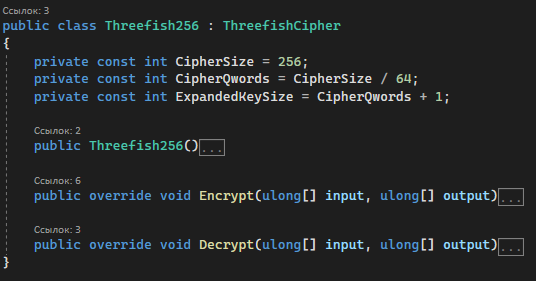


Рисунок 4.14 – Реализация класса Threefish256

Константа CipherSize хранит значение длины блока данных, константа CipherQwords хранит число слов для шифрования, константа ExpandedKeySize хранит значение длины расширенного ключа.

Метод Encrypt реализует операцию зашифрования, метод Decrypt реализует операцию расшифрования.

Классы Threefish512 и Threefish1024 реализованы аналогичным образом, отличие состоит в количестве операций функции Mix.

# **5. Тестирование программного средства**

Для корректной работы программы необходимо обеспечить защиту работы пользователя от ошибок и сбоев. Для этого используются конструкции типа «if-else» и «try-catch». Они служат для отлавливания исключений с последующей их обработкой. Это необходимо, чтобы при вводе пользователем некорректных значений, а также при попытке шифрования пустых значений появлялось сообщение об ошибке.

После запуска приложения можно проследить за потреблением ресурсов компьютера (рисунок 5.1).

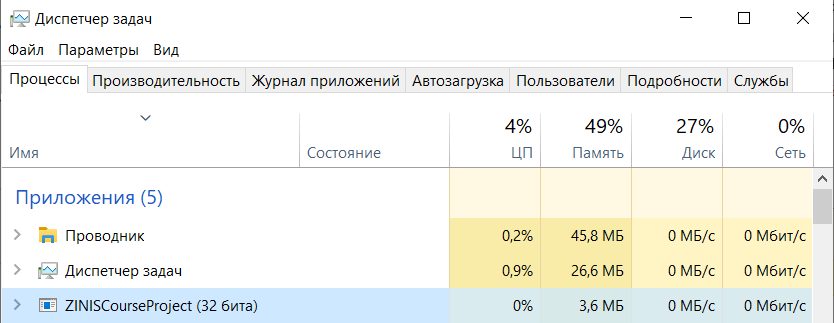


Рисунок 5.1 – Потребляемые ресурсы компьютера

Ниже приведены основные исключительные ситуации.

Для шифрования нужно ввести исходное сообщение. Для этого необходимо в поле «Ведите сообщение» ввести текст либо кликнув по кнопке выбора файла для зашифрования выбрать текстовый файл. При попытке нажатия кнопки зашифрования, когда поле сообщения пустое, появится окно сообщения об ошибке (рисунок 5.2).

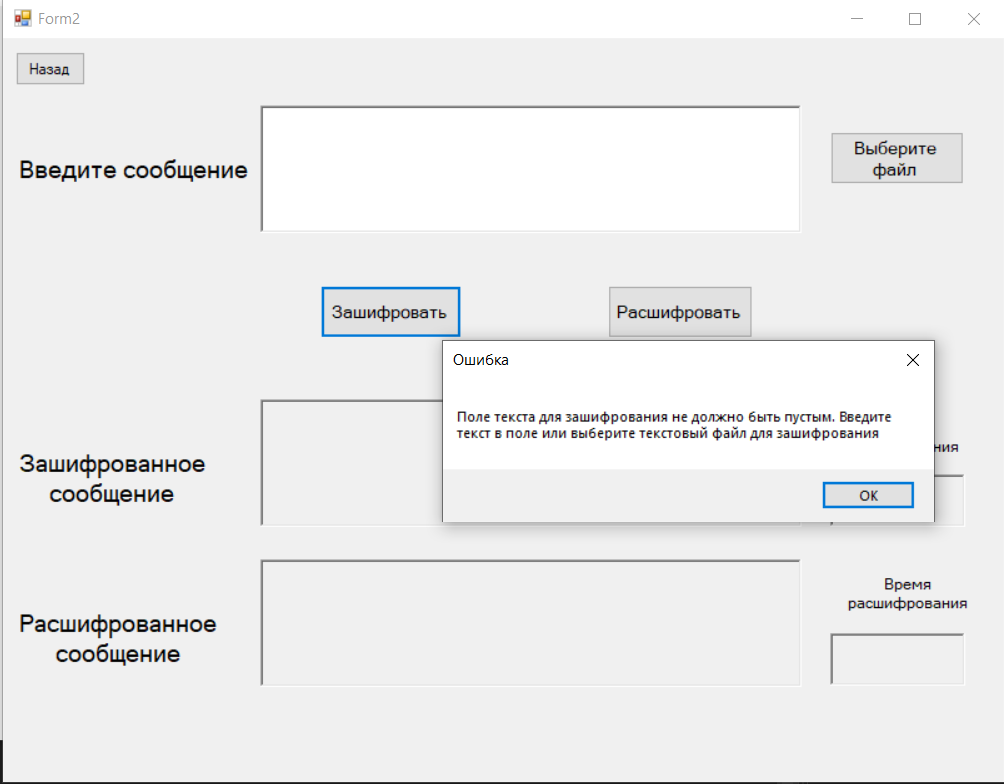


Рисунок 5.2 – Сообщение об ошибке пустого текста для зашифрования

Таким же образом, если пользователь попытается нажать кнопку расшифрования при пустом поле зашифрованного сообщения, появится сообщение об ошибке (рисунок 5.3).

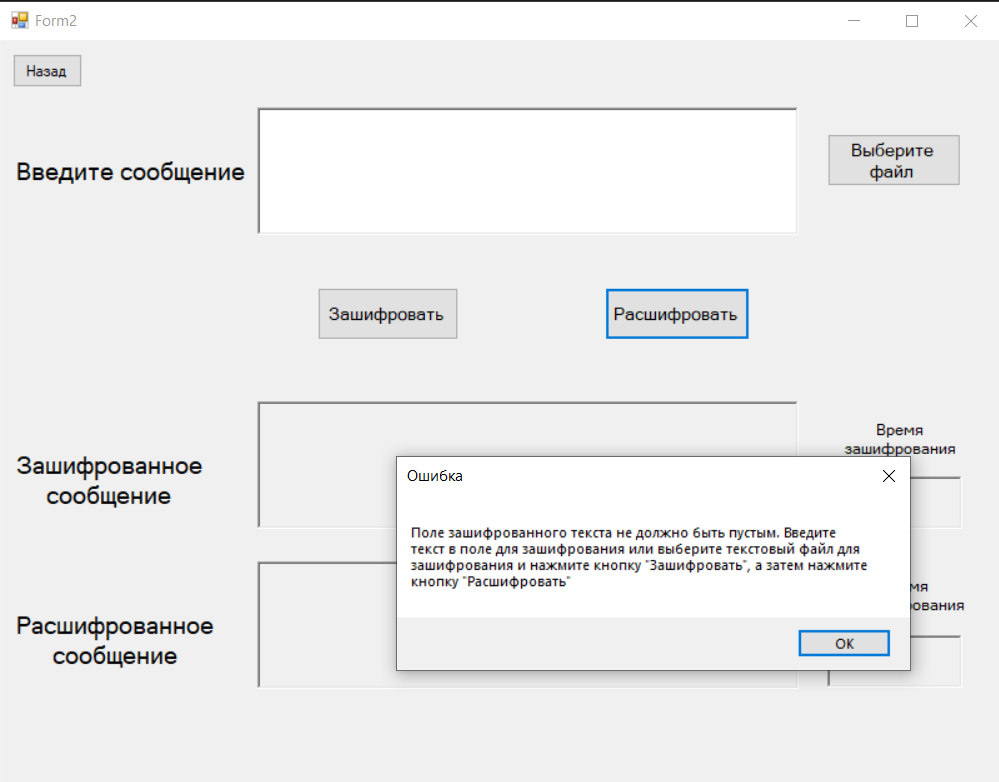


Рисунок 5.3 – Сообщение об ошибке при нажатии кнопки расшифрования, когда поле зашифрованного текста пустое

**6. Руководство пользователя**

Чтобы запустить приложение нужно запустить exe-файл – ZINISCourseProject.exe. При запуске открывается главное окно приложения (рисунок 6.1).

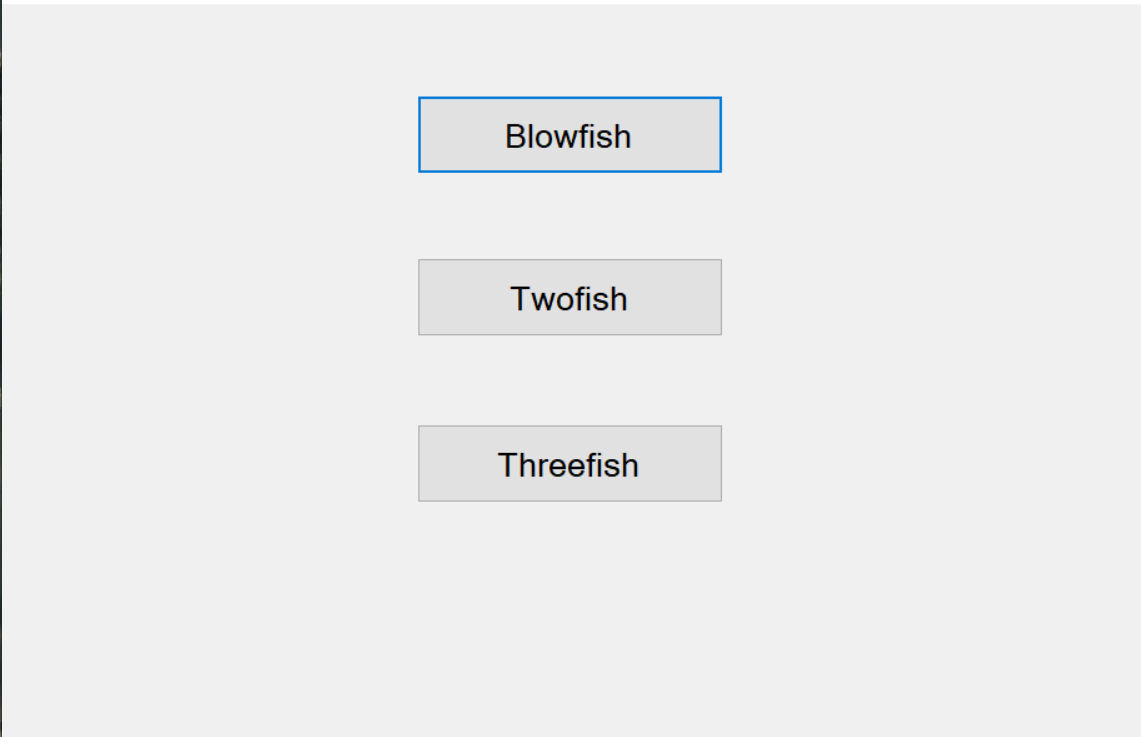


Рисунок 6.1 – Главное окно приложения

Окно содержит кнопки выбора алгоритмов, исследуемых в рамках курсового проекта.

После выбора алгоритма нажатием соответствующей кнопки, появится окно с вводом сообщения и кнопками зашифрования и расшифрования (рисунок 6.2).



Рисунок 6.2 – Окно выполнения процедур зашифрования и расшифрования

В поле «Введите сообщение» нужно ввести сообщение, а затем нажать кнопку «Зашифровать». В поле «Зашифрованное сообщение» будет выведено зашифрованное сообщение, а в поле «Время зашифрования» выведется время, потраченное на зашифрование сообщения, измеренное в тактах (рисунок 6.3).

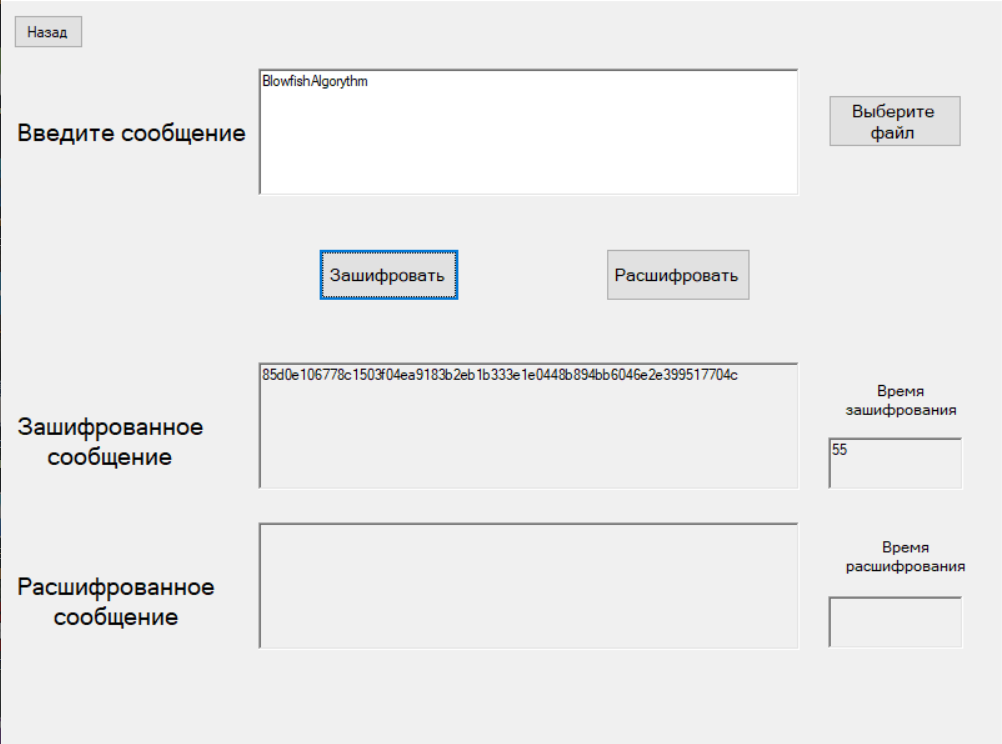


Рисунок 6.3 – Зашифрованное сообщение

В разработанном приложении пользователь может не только самостоятельно вводить сообщение в поле ввода сообщения, но также и выбирать текстовый файл формата .txt, для чего нужно нажать кнопку «Выберите файл» справа от поля ввода сообщения, затем с помощью файлового менеджера выбрать нужный файл для зашифрования и нажать кнопку «Открыть» (рисунок 6.4).

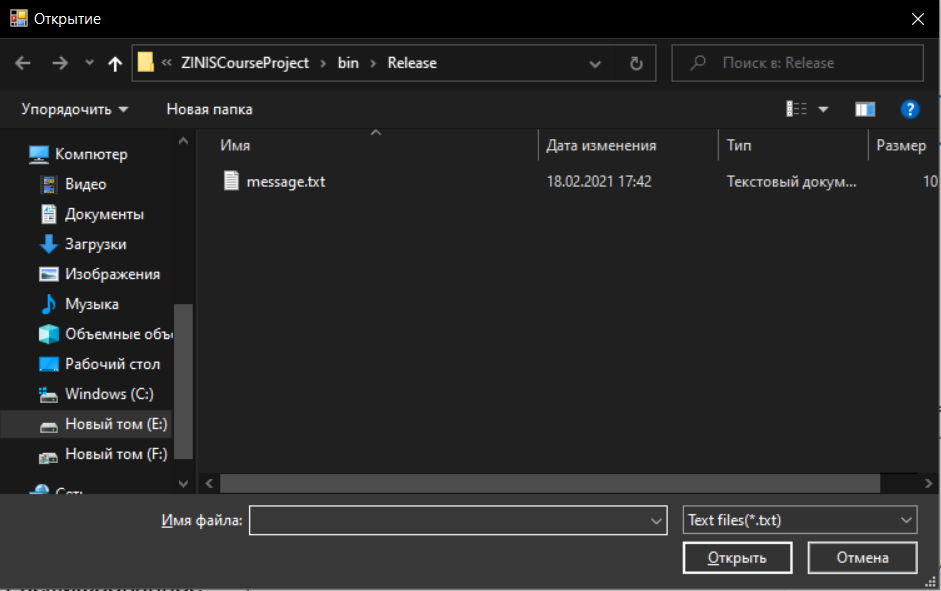


Рисунок 6.4 – Выбор файла для зашифрования

После выбора файла, в поле сообщения выведется содержимое этого файла (рисунок 6.5).

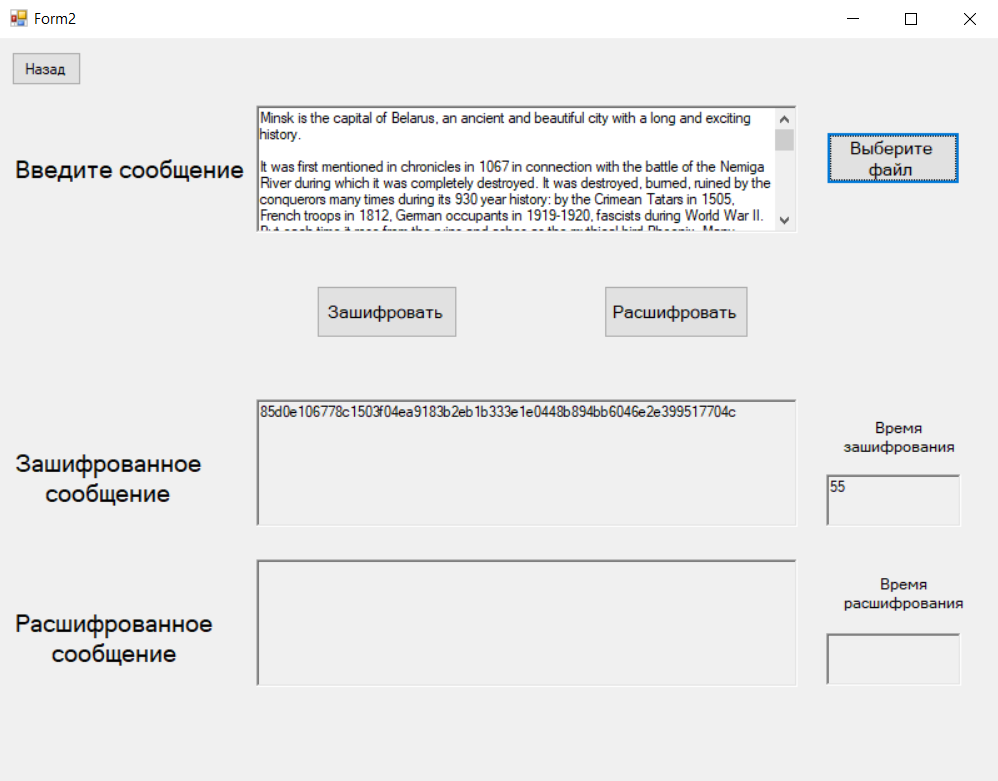


Рисунок 6.5 – Вывод содержимого файла в поле сообщения

Затем нужно нажать кнопку «Зашифровать», после чего в поле «Зашифрованное сообщение» будет выведен шифртекст, а в поле «Время зашифрования» будет выведено потраченное на зашифрование время (рисунок 6.6).

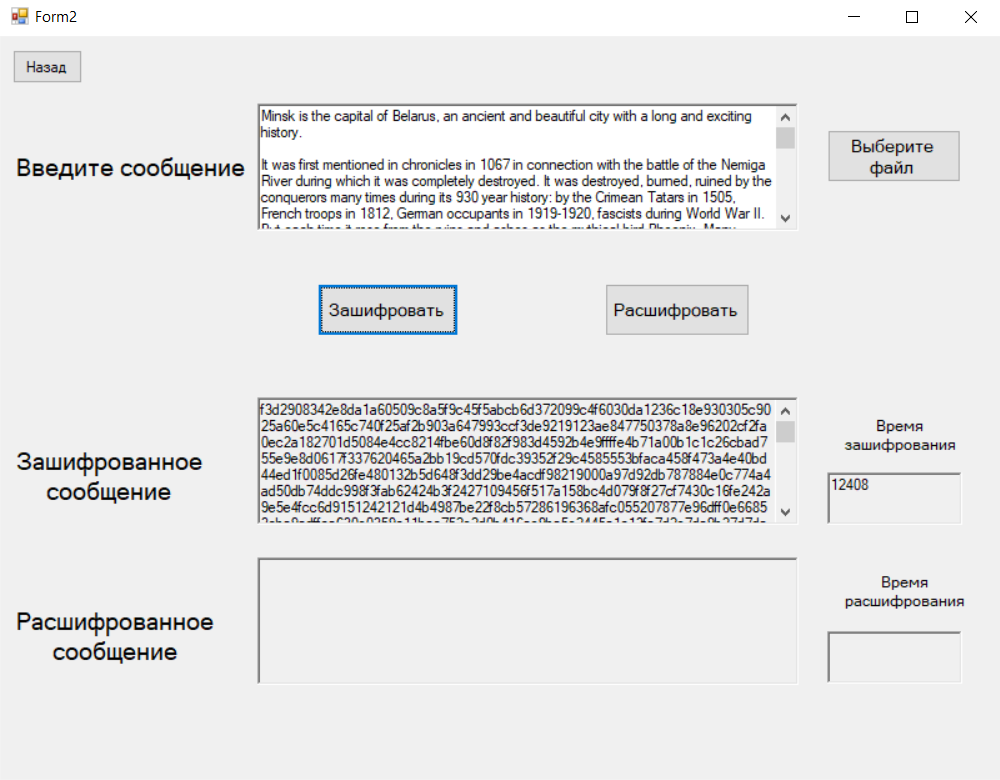


Рисунок 6.6 – Вывод зашифрованного сообщения

После получения зашифрованного сообщения, можно расшифровать его, нажав на кнопку «Расшифровать». Расшифрованное сообщение будет выведено в поле «Расшифрованное сообщение», а время потраченное на расшифрование будет выведено в поле «Время расшифрования» (рисунок 6.7).

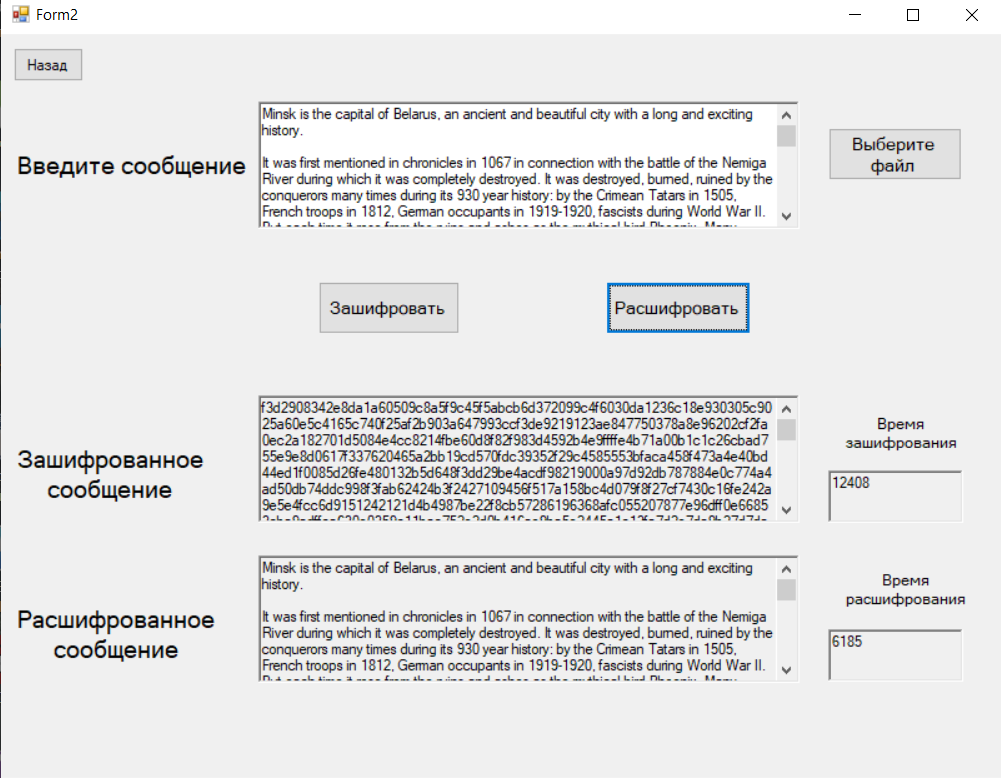


Рисунок 6.7 – Вывод расшифрованного сообщения

В данном руководстве был представлен пример использования алгоритма Blowfish. Работа остальных алгоритмов проводится аналогичным образом.

**Заключение**

Данный курсовой проект был разработан для сравнительного анализа работы блочных алгоритмов шифрования Blowfish, Twofish и Threefish по заданным критериям. Главной задачей было разработать простое и функциональное приложение. для выполнения поставленной цели был использован API-интерфейс Windows Forms и среда разработки Visual Studio 2022. Также был проработан ряд исключительных ситуаций с выводом сообщений в случае некорректной работы пользователя.

В ходе экспериментов было выяснено, что алгоритм Blowfish является быстрым как при шифровании, так и при расшифровании, а алгоритмы Twofish и Threefish проявили медлительность, это обусловлено тем, что в 32-разрядных средах эти алгоритмы не являются эффективными в плане скорости, однако в 64-разрядной среде эти алгоритмы показывают очень хорошую скорость

Все три алгоритма проявили себя как надежные алгоритмы шифрования

Приложение получило удобный интерфейс и является понятной даже для человека, который мало знаком с компьютером или мало использует его в повседневной жизни.

# **Список источников**

1. Сравнительный анализ криптостойкости симметричных алгоритмов шифрования [Электронный ресурс] – <https://elib.belstu.by/bitstream/123456789/33420/1/Bernikov_sravnitel%27nyj%20analiz.pdf> – Дата доступа: 10.04.2022.
2. Blowfish [Электронный ресурс] – <https://ru.wikipedia.org/wiki/Blowfish> – Дата доступа: 27.02.2022.
3. На пути к Skein: просто и понятно про Blowfish [Электронный ресурс] – <https://habr.com/ru/post/140394/> – Дата доступа: 27.02.2022.
4. Алгоритм Blowfish [Электронный ресурс] – <http://crypto.pp.ua/2010/04/66/> – Дата доступа: 27.02.2022.
5. Twofish [Электронный ресурс] – <https://ru.wikipedia.org/wiki/Twofish> – Дата доступа: 01.03.2022.
6. Алгоритмы Twofish и Twofish-FK [Электронный ресурс] – <http://crypto.pp.ua/2010/12/algoritmy-twofish-i-twofish-fk/> – Дата доступа: 02.03.2022.
7. Threefish [Электронный ресурс] – <https://ru.wikipedia.org/wiki/Threefish> – Дата доступа: 03.03.2022.
8. Threefish [Электронный ресурс] – <https://www.schneier.com/academic/skein/threefish/> – Дата доступа: 03.03.2022.
9. Знакомство со Skein [Электронный ресурс] – <https://habr.com/ru/post/531140/> – Дата доступа: 03.03.2022.

# **Приложение А**

**Алгоритм Blowfish**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ZINISCourseProject

{

public class Blowfish

{

#region "Global variables and constants"

const int ROUNDS = 16; //standard is 16, to increase the number of rounds,

//bf\_P needs to be equal to the number of rouds. Use digits of PI.

//Random number generator for creating IVs

RNGCryptoServiceProvider randomSource;

//SBLOCKS

private uint[] bf\_s0;

private uint[] bf\_s1;

private uint[] bf\_s2;

private uint[] bf\_s3;

private uint[] bf\_P;

//KEY

private byte[] key;

//HALF-BLOCKS

private uint xl\_par;

private uint xr\_par;

//Initialization Vector for CBC and CTR mode

private byte[] InitVector;

private bool IVSet;

private bool nonStandardMethod;

#endregion

#region "Constructors"

/// <summary>

/// Constructor for hex key

/// </summary>

/// <param name="hexKey">Cipher key as a hex string</param>

public Blowfish(string hexKey)

{

randomSource = new RNGCryptoServiceProvider();

SetupKey(HexToByte(hexKey));

}

/// <summary>

/// Constructor for byte key

/// </summary>

/// <param name="cipherKey">Cipher key as a byte array</param>

public Blowfish(byte[] cipherKey)

{

randomSource = new RNGCryptoServiceProvider();

SetupKey(cipherKey);

}

#endregion

#region "Public methods"

/// <summary>

/// Encrypts a string in CBC mode

/// </summary>

/// <param name="pt">Plaintext data to encrypt</param>

/// <returns>Ciphertext with IV appended to front</returns>

public string Encrypt\_CBC(string pt)

{

if (!IVSet)

SetRandomIV();

return ByteToHex(InitVector) + ByteToHex(Encrypt\_CBC(Encoding.ASCII.GetBytes(pt)));

}

/// <summary>

/// Decrypts a string in CBC mode

/// </summary>

/// <param name="ct">Ciphertext with IV appended to front</param>

/// <returns>Plaintext</returns>

public string Decrypt\_CBC(string ct)

{

IV = HexToByte(ct.Substring(0, 16));

return Encoding.ASCII.GetString(Decrypt\_CBC(HexToByte(ct.Substring(16)))).Replace("\0", "");

}

/// <summary>

/// Decrypts a byte array in CBC mode.

/// IV must be created and saved manually.

/// </summary>

/// <param name="ct">Ciphertext data to decrypt</param>

/// <returns>Plaintext</returns>

public byte[] Decrypt\_CBC(byte[] ct)

{

return Crypt\_CBC(ct, true);

}

/// <summary>

/// Encrypts a byte array in CBC mode.

/// IV must be created and saved manually.

/// </summary>

/// <param name="pt">Plaintext data to encrypt</param>

/// <returns>Ciphertext</returns>

public byte[] Encrypt\_CBC(byte[] pt)

{

return Crypt\_CBC(pt, false);

}

/// <summary>

/// Encrypt a string in ECB mode

/// </summary>

/// <param name="pt">Plaintext to encrypt as ascii string</param>

/// <returns>hex value of encrypted data</returns>

public string Encrypt\_ECB(string pt)

{

return ByteToHex(Encrypt\_ECB(Encoding.ASCII.GetBytes(pt)));

}

/// <summary>

/// Decrypts a string (ECB)

/// </summary>

/// <param name="ct">hHex string of the ciphertext</param>

/// <returns>Plaintext ascii string</returns>

public string Decrypt\_ECB(string ct)

{

return Encoding.ASCII.GetString(Decrypt\_ECB(HexToByte(ct))).Replace("\0", "");

}

/// <summary>

/// Encrypts a byte array in ECB mode

/// </summary>

/// <param name="pt">Plaintext data</param>

/// <returns>Ciphertext bytes</returns>

public byte[] Encrypt\_ECB(byte[] pt)

{

return Crypt\_ECB(pt, false);

}

/// <summary>

/// Decrypts a byte array (ECB)

/// </summary>

/// <param name="ct">Ciphertext byte array</param>

/// <returns>Plaintext</returns>

public byte[] Decrypt\_ECB(byte[] ct)

{

return Crypt\_ECB(ct, true);

}

/// <summary>

/// Encrypts a string (CTR)

/// </summary>

/// <param name="pt">The plaintext to encrypt</param>

/// <returns>The ciphertext</returns>

public string Encrypt\_CTR(string pt)

{

if (!IVSet)

SetRandomIV();

return ByteToHex(InitVector) + ByteToHex(Crypt\_CTR(Encoding.ASCII.GetBytes(pt), 2));

}

/// <summary>

/// Decrypt a string (CTR)

/// </summary>

/// <param name="ct">The ciphertext to decrypt</param>

/// <returns>The plaintext</returns>

public string Decrypt\_CTR(string ct)

{

IV = HexToByte(ct.Substring(0, 16));

return Encoding.ASCII.GetString(Crypt\_CTR(HexToByte(ct.Substring(16)), 2)).Replace("\0", "");

}

/// <summary>

/// Initialization vector for CBC mode.

/// </summary>

public byte[] IV

{

get { return InitVector; }

set

{

if (value.Length == 8)

{

InitVector = value;

IVSet = true;

}

else

{

throw new Exception("Invalid IV size.");

}

}

}

public bool NonStandard

{

get { return nonStandardMethod; }

set { nonStandardMethod = value; }

}

/// <summary>

/// Creates and sets a random initialization vector.

/// </summary>

/// <returns>The random IV</returns>

public byte[] SetRandomIV()

{

InitVector = new byte[8];

randomSource.GetBytes(InitVector);

IVSet = true;

return InitVector;

}

#endregion

#region Cryptography

/// <summary>

/// Sets up the S-blocks and the key

/// </summary>

/// <param name="cipherKey">Block cipher key (1-448 bits)</param>

private void SetupKey(byte[] cipherKey)

{

bf\_P = SetupP();

//set up the S blocks

bf\_s0 = SetupS0();

bf\_s1 = SetupS1();

bf\_s2 = SetupS2();

bf\_s3 = SetupS3();

key = new byte[cipherKey.Length]; // 448 bits

if (cipherKey.Length > 56)

{

throw new Exception("Key too long. 56 bytes required.");

}

Buffer.BlockCopy(cipherKey, 0, key, 0, cipherKey.Length);

int j = 0;

for (int i = 0; i < 18; i++)

{

uint d = (uint)(((key[j % cipherKey.Length] \* 256 + key[(j + 1) % cipherKey.Length]) \* 256 + key[(j + 2) % cipherKey.Length]) \* 256 + key[(j + 3) % cipherKey.Length]);

bf\_P[i] ^= d;

j = (j + 4) % cipherKey.Length;

}

xl\_par = 0;

xr\_par = 0;

for (int i = 0; i < 18; i += 2)

{

encipher();

bf\_P[i] = xl\_par;

bf\_P[i + 1] = xr\_par;

}

for (int i = 0; i < 256; i += 2)

{

encipher();

bf\_s0[i] = xl\_par;

bf\_s0[i + 1] = xr\_par;

}

for (int i = 0; i < 256; i += 2)

{

encipher();

bf\_s1[i] = xl\_par;

bf\_s1[i + 1] = xr\_par;

}

for (int i = 0; i < 256; i += 2)

{

encipher();

bf\_s2[i] = xl\_par;

bf\_s2[i + 1] = xr\_par;

}

for (int i = 0; i < 256; i += 2)

{

encipher();

bf\_s3[i] = xl\_par;

bf\_s3[i + 1] = xr\_par;

}

}

/// <summary>

/// Encrypts or decrypts data in ECB mode

/// </summary>

/// <param name="text">plain/ciphertext</param>

/// <param name="decrypt">true to decrypt, false to encrypt</param>

/// <returns>(En/De)crypted data</returns>

private byte[] Crypt\_ECB(byte[] text, bool decrypt)

{

int paddedLen = (text.Length % 8 == 0 ? text.Length : text.Length + 8 - (text.Length % 8));

byte[] plainText = new byte[paddedLen];

Buffer.BlockCopy(text, 0, plainText, 0, text.Length);

byte[] block = new byte[8];

for (int i = 0; i < plainText.Length; i += 8)

{

Buffer.BlockCopy(plainText, i, block, 0, 8);

if (decrypt)

{

BlockDecrypt(ref block);

}

else

{

BlockEncrypt(ref block);

}

Buffer.BlockCopy(block, 0, plainText, i, 8);

}

return plainText;

}

public byte[] Crypt\_CTR(byte[] text, int numThreads)

{

if (!IVSet)

{

throw new Exception("IV not set.");

}

byte[] input = new byte[8];

byte[] counter = new byte[8];

int paddedLen = (text.Length % 8 == 0 ? text.Length : text.Length + 8 - (text.Length % 8));

byte[] plainText = new byte[paddedLen];

Buffer.BlockCopy(text, 0, plainText, 0, text.Length);

byte[] block = new byte[8];

for (int i = 0; i < plainText.Length; i += 8)

{

for (int x = 0; x < 8; x++)

{

input[x] = (byte)(counter[x] ^ InitVector[x]);

}

Buffer.BlockCopy(plainText, i, block, 0, 8);

BlockEncrypt(ref input);

XorBlock(ref block, input);

Buffer.BlockCopy(block, 0, plainText, i, 8);

}

return plainText;

}

/// <summary>

/// Encrypts or decrypts data in CBC mode

/// </summary>

/// <param name="text">plain/ciphertext</param>

/// <param name="decrypt">true to decrypt, false to encrypt</param>

/// <returns>(En/De)crypted data</returns>

private byte[] Crypt\_CBC(byte[] text, bool decrypt)

{

if (!IVSet)

{

throw new Exception("IV not set.");

}

int paddedLen = (text.Length % 8 == 0 ? text.Length : text.Length + 8 - (text.Length % 8));

byte[] plainText = new byte[paddedLen];

Buffer.BlockCopy(text, 0, plainText, 0, text.Length);

byte[] block = new byte[8];

byte[] preblock = new byte[8];

byte[] iv = new byte[8];

Buffer.BlockCopy(InitVector, 0, iv, 0, 8);

if (!decrypt)

{

for (int i = 0; i < plainText.Length; i += 8)

{

Buffer.BlockCopy(plainText, i, block, 0, 8);

XorBlock(ref block, iv);

BlockEncrypt(ref block);

Buffer.BlockCopy(block, 0, iv, 0, 8);

Buffer.BlockCopy(block, 0, plainText, i, 8);

}

}

else

{

for (int i = 0; i < plainText.Length; i += 8)

{

Buffer.BlockCopy(plainText, i, block, 0, 8);

Buffer.BlockCopy(block, 0, preblock, 0, 8);

BlockDecrypt(ref block);

XorBlock(ref block, iv);

Buffer.BlockCopy(preblock, 0, iv, 0, 8);

Buffer.BlockCopy(block, 0, plainText, i, 8);

}

}

return plainText;

}

/// <summary>

/// XoR encrypts two 8 bit blocks

/// </summary>

/// <param name="block">8 bit block 1</param>

/// <param name="iv">8 bit block 2</param>

private void XorBlock(ref byte[] block, byte[] iv)

{

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

{

block[i] ^= iv[i];

}

}

/// <summary>

/// Encrypts a 64 bit block

/// </summary>

/// <param name="block">The 64 bit block to encrypt</param>

private void BlockEncrypt(ref byte[] block)

{

SetBlock(block);

encipher();

GetBlock(ref block);

}

/// <summary>

/// Decrypts a 64 bit block

/// </summary>

/// <param name="block">The 64 bit block to decrypt</param>

private void BlockDecrypt(ref byte[] block)

{

SetBlock(block);

decipher();

GetBlock(ref block);

}

/// <summary>

/// Splits the block into the two uint values

/// </summary>

/// <param name="block">the 64 bit block to setup</param>

private void SetBlock(byte[] block)

{

byte[] block1 = new byte[4];

byte[] block2 = new byte[4];

Buffer.BlockCopy(block, 0, block1, 0, 4);

Buffer.BlockCopy(block, 4, block2, 0, 4);

//split the block

if (nonStandardMethod)

{

xr\_par = BitConverter.ToUInt32(block1, 0);

xl\_par = BitConverter.ToUInt32(block2, 0);

}

else

{

//ToUInt32 requires the bytes in reverse order

Array.Reverse(block1);

Array.Reverse(block2);

xl\_par = BitConverter.ToUInt32(block1, 0);

xr\_par = BitConverter.ToUInt32(block2, 0);

}

}

/// <summary>

/// Converts the two uint values into a 64 bit block

/// </summary>

/// <param name="block">64 bit buffer to receive the block</param>

private void GetBlock(ref byte[] block)

{

byte[] block1 = new byte[4];

byte[] block2 = new byte[4];

if (nonStandardMethod)

{

block1 = BitConverter.GetBytes(xr\_par);

block2 = BitConverter.GetBytes(xl\_par);

}

else

{

block1 = BitConverter.GetBytes(xl\_par);

block2 = BitConverter.GetBytes(xr\_par);

//GetBytes returns the bytes in reverse order

Array.Reverse(block1);

Array.Reverse(block2);

}

//join the block

Buffer.BlockCopy(block1, 0, block, 0, 4);

Buffer.BlockCopy(block2, 0, block, 4, 4);

}

/// <summary>

/// Runs the blowfish algorithm (standard 16 rounds)

/// </summary>

private void encipher()

{

xl\_par ^= bf\_P[0];

for (uint i = 0; i < ROUNDS; i += 2)

{

xr\_par = round(xr\_par, xl\_par, i + 1);

xl\_par = round(xl\_par, xr\_par, i + 2);

}

xr\_par = xr\_par ^ bf\_P[17];

//swap the blocks

uint swap = xl\_par;

xl\_par = xr\_par;

xr\_par = swap;

}

/// <summary>

/// Runs the blowfish algorithm in reverse (standard 16 rounds)

/// </summary>

private void decipher()

{

xl\_par ^= bf\_P[17];

for (uint i = 16; i > 0; i -= 2)

{

xr\_par = round(xr\_par, xl\_par, i);

xl\_par = round(xl\_par, xr\_par, i - 1);

}

xr\_par = xr\_par ^ bf\_P[0];

//swap the blocks

uint swap = xl\_par;

xl\_par = xr\_par;

xr\_par = swap;

}

/// <summary>

/// one round of the blowfish algorithm

/// </summary>

/// <param name="a">See spec</param>

/// <param name="b">See spec</param>

/// <param name="n">See spec</param>

/// <returns></returns>

private uint round(uint a, uint b, uint n)

{

uint x1 = (bf\_s0[wordByte0(b)] + bf\_s1[wordByte1(b)]) ^ bf\_s2[wordByte2(b)];

uint x2 = x1 + bf\_s3[this.wordByte3(b)];

uint x3 = x2 ^ bf\_P[n];

return x3 ^ a;

}

#endregion

#region SBLOCKS

//SBLOCKS ARE THE HEX DIGITS OF PI.

//The amount of hex digits can be increased if you want to experiment with more rounds and longer key lengths

//Increase the size of this array when increasing the number of rounds

private uint[] SetupP()

{

return new uint[] {

0x243f6a88,0x85a308d3,0x13198a2e,0x03707344,0xa4093822,0x299f31d0,

0x082efa98,0xec4e6c89,0x452821e6,0x38d01377,0xbe5466cf,0x34e90c6c,

0xc0ac29b7,0xc97c50dd,0x3f84d5b5,0xb5470917,0x9216d5d9,0x8979fb1b

};

}

private uint[] SetupS0()

{

return new uint[] {

0xd1310ba6,0x98dfb5ac,0x2ffd72db,0xd01adfb7,0xb8e1afed,0x6a267e96,

0xba7c9045,0xf12c7f99,0x24a19947,0xb3916cf7,0x0801f2e2,0x858efc16,

0x636920d8,0x71574e69,0xa458fea3,0xf4933d7e,0x0d95748f,0x728eb658,

0x718bcd58,0x82154aee,0x7b54a41d,0xc25a59b5,0x9c30d539,0x2af26013,

0xc5d1b023,0x286085f0,0xca417918,0xb8db38ef,0x8e79dcb0,0x603a180e,

0x6c9e0e8b,0xb01e8a3e,0xd71577c1,0xbd314b27,0x78af2fda,0x55605c60,

0xe65525f3,0xaa55ab94,0x57489862,0x63e81440,0x55ca396a,0x2aab10b6,

0xb4cc5c34,0x1141e8ce,0xa15486af,0x7c72e993,0xb3ee1411,0x636fbc2a,

0x2ba9c55d,0x741831f6,0xce5c3e16,0x9b87931e,0xafd6ba33,0x6c24cf5c,

0x7a325381,0x28958677,0x3b8f4898,0x6b4bb9af,0xc4bfe81b,0x66282193,

0x61d809cc,0xfb21a991,0x487cac60,0x5dec8032,0xef845d5d,0xe98575b1,

0xdc262302,0xeb651b88,0x23893e81,0xd396acc5,0x0f6d6ff3,0x83f44239,

0x2e0b4482,0xa4842004,0x69c8f04a,0x9e1f9b5e,0x21c66842,0xf6e96c9a,

0x670c9c61,0xabd388f0,0x6a51a0d2,0xd8542f68,0x960fa728,0xab5133a3,

0x6eef0b6c,0x137a3be4,0xba3bf050,0x7efb2a98,0xa1f1651d,0x39af0176,

0x66ca593e,0x82430e88,0x8cee8619,0x456f9fb4,0x7d84a5c3,0x3b8b5ebe,

0xe06f75d8,0x85c12073,0x401a449f,0x56c16aa6,0x4ed3aa62,0x363f7706,

0x1bfedf72,0x429b023d,0x37d0d724,0xd00a1248,0xdb0fead3,0x49f1c09b,

0x075372c9,0x80991b7b,0x25d479d8,0xf6e8def7,0xe3fe501a,0xb6794c3b,

0x976ce0bd,0x04c006ba,0xc1a94fb6,0x409f60c4,0x5e5c9ec2,0x196a2463,

0x68fb6faf,0x3e6c53b5,0x1339b2eb,0x3b52ec6f,0x6dfc511f,0x9b30952c,

0xcc814544,0xaf5ebd09,0xbee3d004,0xde334afd,0x660f2807,0x192e4bb3,

0xc0cba857,0x45c8740f,0xd20b5f39,0xb9d3fbdb,0x5579c0bd,0x1a60320a,

0xd6a100c6,0x402c7279,0x679f25fe,0xfb1fa3cc,0x8ea5e9f8,0xdb3222f8,

0x3c7516df,0xfd616b15,0x2f501ec8,0xad0552ab,0x323db5fa,0xfd238760,

0x53317b48,0x3e00df82,0x9e5c57bb,0xca6f8ca0,0x1a87562e,0xdf1769db,

0xd542a8f6,0x287effc3,0xac6732c6,0x8c4f5573,0x695b27b0,0xbbca58c8,

0xe1ffa35d,0xb8f011a0,0x10fa3d98,0xfd2183b8,0x4afcb56c,0x2dd1d35b,

0x9a53e479,0xb6f84565,0xd28e49bc,0x4bfb9790,0xe1ddf2da,0xa4cb7e33,

0x62fb1341,0xcee4c6e8,0xef20cada,0x36774c01,0xd07e9efe,0x2bf11fb4,

0x95dbda4d,0xae909198,0xeaad8e71,0x6b93d5a0,0xd08ed1d0,0xafc725e0,

0x8e3c5b2f,0x8e7594b7,0x8ff6e2fb,0xf2122b64,0x8888b812,0x900df01c,

0x4fad5ea0,0x688fc31c,0xd1cff191,0xb3a8c1ad,0x2f2f2218,0xbe0e1777,

0xea752dfe,0x8b021fa1,0xe5a0cc0f,0xb56f74e8,0x18acf3d6,0xce89e299,

0xb4a84fe0,0xfd13e0b7,0x7cc43b81,0xd2ada8d9,0x165fa266,0x80957705,

0x93cc7314,0x211a1477,0xe6ad2065,0x77b5fa86,0xc75442f5,0xfb9d35cf,

0xebcdaf0c,0x7b3e89a0,0xd6411bd3,0xae1e7e49,0x00250e2d,0x2071b35e,

0x226800bb,0x57b8e0af,0x2464369b,0xf009b91e,0x5563911d,0x59dfa6aa,

0x78c14389,0xd95a537f,0x207d5ba2,0x02e5b9c5,0x83260376,0x6295cfa9,

0x11c81968,0x4e734a41,0xb3472dca,0x7b14a94a,0x1b510052,0x9a532915,

0xd60f573f,0xbc9bc6e4,0x2b60a476,0x81e67400,0x08ba6fb5,0x571be91f,

0xf296ec6b,0x2a0dd915,0xb6636521,0xe7b9f9b6,0xff34052e,0xc5855664,

0x53b02d5d,0xa99f8fa1,0x08ba4799,0x6e85076a

};

}

private uint[] SetupS1()

{

return new uint[] {

0x4b7a70e9,0xb5b32944,0xdb75092e,0xc4192623,0xad6ea6b0,0x49a7df7d,

0x9cee60b8,0x8fedb266,0xecaa8c71,0x699a17ff,0x5664526c,0xc2b19ee1,

0x193602a5,0x75094c29,0xa0591340,0xe4183a3e,0x3f54989a,0x5b429d65,

0x6b8fe4d6,0x99f73fd6,0xa1d29c07,0xefe830f5,0x4d2d38e6,0xf0255dc1,

0x4cdd2086,0x8470eb26,0x6382e9c6,0x021ecc5e,0x09686b3f,0x3ebaefc9,

0x3c971814,0x6b6a70a1,0x687f3584,0x52a0e286,0xb79c5305,0xaa500737,

0x3e07841c,0x7fdeae5c,0x8e7d44ec,0x5716f2b8,0xb03ada37,0xf0500c0d,

0xf01c1f04,0x0200b3ff,0xae0cf51a,0x3cb574b2,0x25837a58,0xdc0921bd,

0xd19113f9,0x7ca92ff6,0x94324773,0x22f54701,0x3ae5e581,0x37c2dadc,

0xc8b57634,0x9af3dda7,0xa9446146,0x0fd0030e,0xecc8c73e,0xa4751e41,

0xe238cd99,0x3bea0e2f,0x3280bba1,0x183eb331,0x4e548b38,0x4f6db908,

0x6f420d03,0xf60a04bf,0x2cb81290,0x24977c79,0x5679b072,0xbcaf89af,

0xde9a771f,0xd9930810,0xb38bae12,0xdccf3f2e,0x5512721f,0x2e6b7124,

0x501adde6,0x9f84cd87,0x7a584718,0x7408da17,0xbc9f9abc,0xe94b7d8c,

0xec7aec3a,0xdb851dfa,0x63094366,0xc464c3d2,0xef1c1847,0x3215d908,

0xdd433b37,0x24c2ba16,0x12a14d43,0x2a65c451,0x50940002,0x133ae4dd,

0x71dff89e,0x10314e55,0x81ac77d6,0x5f11199b,0x043556f1,0xd7a3c76b,

0x3c11183b,0x5924a509,0xf28fe6ed,0x97f1fbfa,0x9ebabf2c,0x1e153c6e,

0x86e34570,0xeae96fb1,0x860e5e0a,0x5a3e2ab3,0x771fe71c,0x4e3d06fa,

0x2965dcb9,0x99e71d0f,0x803e89d6,0x5266c825,0x2e4cc978,0x9c10b36a,

0xc6150eba,0x94e2ea78,0xa5fc3c53,0x1e0a2df4,0xf2f74ea7,0x361d2b3d,

0x1939260f,0x19c27960,0x5223a708,0xf71312b6,0xebadfe6e,0xeac31f66,

0xe3bc4595,0xa67bc883,0xb17f37d1,0x018cff28,0xc332ddef,0xbe6c5aa5,

0x65582185,0x68ab9802,0xeecea50f,0xdb2f953b,0x2aef7dad,0x5b6e2f84,

0x1521b628,0x29076170,0xecdd4775,0x619f1510,0x13cca830,0xeb61bd96,

0x0334fe1e,0xaa0363cf,0xb5735c90,0x4c70a239,0xd59e9e0b,0xcbaade14,

0xeecc86bc,0x60622ca7,0x9cab5cab,0xb2f3846e,0x648b1eaf,0x19bdf0ca,

0xa02369b9,0x655abb50,0x40685a32,0x3c2ab4b3,0x319ee9d5,0xc021b8f7,

0x9b540b19,0x875fa099,0x95f7997e,0x623d7da8,0xf837889a,0x97e32d77,

0x11ed935f,0x16681281,0x0e358829,0xc7e61fd6,0x96dedfa1,0x7858ba99,

0x57f584a5,0x1b227263,0x9b83c3ff,0x1ac24696,0xcdb30aeb,0x532e3054,

0x8fd948e4,0x6dbc3128,0x58ebf2ef,0x34c6ffea,0xfe28ed61,0xee7c3c73,

0x5d4a14d9,0xe864b7e3,0x42105d14,0x203e13e0,0x45eee2b6,0xa3aaabea,

0xdb6c4f15,0xfacb4fd0,0xc742f442,0xef6abbb5,0x654f3b1d,0x41cd2105,

0xd81e799e,0x86854dc7,0xe44b476a,0x3d816250,0xcf62a1f2,0x5b8d2646,

0xfc8883a0,0xc1c7b6a3,0x7f1524c3,0x69cb7492,0x47848a0b,0x5692b285,

0x095bbf00,0xad19489d,0x1462b174,0x23820e00,0x58428d2a,0x0c55f5ea,

0x1dadf43e,0x233f7061,0x3372f092,0x8d937e41,0xd65fecf1,0x6c223bdb,

0x7cde3759,0xcbee7460,0x4085f2a7,0xce77326e,0xa6078084,0x19f8509e,

0xe8efd855,0x61d99735,0xa969a7aa,0xc50c06c2,0x5a04abfc,0x800bcadc,

0x9e447a2e,0xc3453484,0xfdd56705,0x0e1e9ec9,0xdb73dbd3,0x105588cd,

0x675fda79,0xe3674340,0xc5c43465,0x713e38d8,0x3d28f89e,0xf16dff20,

0x153e21e7,0x8fb03d4a,0xe6e39f2b,0xdb83adf7

};

}

private uint[] SetupS2()

{

return new uint[] {

0xe93d5a68,0x948140f7,0xf64c261c,0x94692934,0x411520f7,0x7602d4f7,

0xbcf46b2e,0xd4a20068,0xd4082471,0x3320f46a,0x43b7d4b7,0x500061af,

0x1e39f62e,0x97244546,0x14214f74,0xbf8b8840,0x4d95fc1d,0x96b591af,

0x70f4ddd3,0x66a02f45,0xbfbc09ec,0x03bd9785,0x7fac6dd0,0x31cb8504,

0x96eb27b3,0x55fd3941,0xda2547e6,0xabca0a9a,0x28507825,0x530429f4,

0x0a2c86da,0xe9b66dfb,0x68dc1462,0xd7486900,0x680ec0a4,0x27a18dee,

0x4f3ffea2,0xe887ad8c,0xb58ce006,0x7af4d6b6,0xaace1e7c,0xd3375fec,

0xce78a399,0x406b2a42,0x20fe9e35,0xd9f385b9,0xee39d7ab,0x3b124e8b,

0x1dc9faf7,0x4b6d1856,0x26a36631,0xeae397b2,0x3a6efa74,0xdd5b4332,

0x6841e7f7,0xca7820fb,0xfb0af54e,0xd8feb397,0x454056ac,0xba489527,

0x55533a3a,0x20838d87,0xfe6ba9b7,0xd096954b,0x55a867bc,0xa1159a58,

0xcca92963,0x99e1db33,0xa62a4a56,0x3f3125f9,0x5ef47e1c,0x9029317c,

0xfdf8e802,0x04272f70,0x80bb155c,0x05282ce3,0x95c11548,0xe4c66d22,

0x48c1133f,0xc70f86dc,0x07f9c9ee,0x41041f0f,0x404779a4,0x5d886e17,

0x325f51eb,0xd59bc0d1,0xf2bcc18f,0x41113564,0x257b7834,0x602a9c60,

0xdff8e8a3,0x1f636c1b,0x0e12b4c2,0x02e1329e,0xaf664fd1,0xcad18115,

0x6b2395e0,0x333e92e1,0x3b240b62,0xeebeb922,0x85b2a20e,0xe6ba0d99,

0xde720c8c,0x2da2f728,0xd0127845,0x95b794fd,0x647d0862,0xe7ccf5f0,

0x5449a36f,0x877d48fa,0xc39dfd27,0xf33e8d1e,0x0a476341,0x992eff74,

0x3a6f6eab,0xf4f8fd37,0xa812dc60,0xa1ebddf8,0x991be14c,0xdb6e6b0d,

0xc67b5510,0x6d672c37,0x2765d43b,0xdcd0e804,0xf1290dc7,0xcc00ffa3,

0xb5390f92,0x690fed0b,0x667b9ffb,0xcedb7d9c,0xa091cf0b,0xd9155ea3,

0xbb132f88,0x515bad24,0x7b9479bf,0x763bd6eb,0x37392eb3,0xcc115979,

0x8026e297,0xf42e312d,0x6842ada7,0xc66a2b3b,0x12754ccc,0x782ef11c,

0x6a124237,0xb79251e7,0x06a1bbe6,0x4bfb6350,0x1a6b1018,0x11caedfa,

0x3d25bdd8,0xe2e1c3c9,0x44421659,0x0a121386,0xd90cec6e,0xd5abea2a,

0x64af674e,0xda86a85f,0xbebfe988,0x64e4c3fe,0x9dbc8057,0xf0f7c086,

0x60787bf8,0x6003604d,0xd1fd8346,0xf6381fb0,0x7745ae04,0xd736fccc,

0x83426b33,0xf01eab71,0xb0804187,0x3c005e5f,0x77a057be,0xbde8ae24,

0x55464299,0xbf582e61,0x4e58f48f,0xf2ddfda2,0xf474ef38,0x8789bdc2,

0x5366f9c3,0xc8b38e74,0xb475f255,0x46fcd9b9,0x7aeb2661,0x8b1ddf84,

0x846a0e79,0x915f95e2,0x466e598e,0x20b45770,0x8cd55591,0xc902de4c,

0xb90bace1,0xbb8205d0,0x11a86248,0x7574a99e,0xb77f19b6,0xe0a9dc09,

0x662d09a1,0xc4324633,0xe85a1f02,0x09f0be8c,0x4a99a025,0x1d6efe10,

0x1ab93d1d,0x0ba5a4df,0xa186f20f,0x2868f169,0xdcb7da83,0x573906fe,

0xa1e2ce9b,0x4fcd7f52,0x50115e01,0xa70683fa,0xa002b5c4,0x0de6d027,

0x9af88c27,0x773f8641,0xc3604c06,0x61a806b5,0xf0177a28,0xc0f586e0,

0x006058aa,0x30dc7d62,0x11e69ed7,0x2338ea63,0x53c2dd94,0xc2c21634,

0xbbcbee56,0x90bcb6de,0xebfc7da1,0xce591d76,0x6f05e409,0x4b7c0188,

0x39720a3d,0x7c927c24,0x86e3725f,0x724d9db9,0x1ac15bb4,0xd39eb8fc,

0xed545578,0x08fca5b5,0xd83d7cd3,0x4dad0fc4,0x1e50ef5e,0xb161e6f8,

0xa28514d9,0x6c51133c,0x6fd5c7e7,0x56e14ec4,0x362abfce,0xddc6c837,

0xd79a3234,0x92638212,0x670efa8e,0x406000e0

};

}

private uint[] SetupS3()

{

return new uint[] {

0x3a39ce37,0xd3faf5cf,0xabc27737,0x5ac52d1b,0x5cb0679e,0x4fa33742,

0xd3822740,0x99bc9bbe,0xd5118e9d,0xbf0f7315,0xd62d1c7e,0xc700c47b,

0xb78c1b6b,0x21a19045,0xb26eb1be,0x6a366eb4,0x5748ab2f,0xbc946e79,

0xc6a376d2,0x6549c2c8,0x530ff8ee,0x468dde7d,0xd5730a1d,0x4cd04dc6,

0x2939bbdb,0xa9ba4650,0xac9526e8,0xbe5ee304,0xa1fad5f0,0x6a2d519a,

0x63ef8ce2,0x9a86ee22,0xc089c2b8,0x43242ef6,0xa51e03aa,0x9cf2d0a4,

0x83c061ba,0x9be96a4d,0x8fe51550,0xba645bd6,0x2826a2f9,0xa73a3ae1,

0x4ba99586,0xef5562e9,0xc72fefd3,0xf752f7da,0x3f046f69,0x77fa0a59,

0x80e4a915,0x87b08601,0x9b09e6ad,0x3b3ee593,0xe990fd5a,0x9e34d797,

0x2cf0b7d9,0x022b8b51,0x96d5ac3a,0x017da67d,0xd1cf3ed6,0x7c7d2d28,

0x1f9f25cf,0xadf2b89b,0x5ad6b472,0x5a88f54c,0xe029ac71,0xe019a5e6,

0x47b0acfd,0xed93fa9b,0xe8d3c48d,0x283b57cc,0xf8d56629,0x79132e28,

0x785f0191,0xed756055,0xf7960e44,0xe3d35e8c,0x15056dd4,0x88f46dba,

0x03a16125,0x0564f0bd,0xc3eb9e15,0x3c9057a2,0x97271aec,0xa93a072a,

0x1b3f6d9b,0x1e6321f5,0xf59c66fb,0x26dcf319,0x7533d928,0xb155fdf5,

0x03563482,0x8aba3cbb,0x28517711,0xc20ad9f8,0xabcc5167,0xccad925f,

0x4de81751,0x3830dc8e,0x379d5862,0x9320f991,0xea7a90c2,0xfb3e7bce,

0x5121ce64,0x774fbe32,0xa8b6e37e,0xc3293d46,0x48de5369,0x6413e680,

0xa2ae0810,0xdd6db224,0x69852dfd,0x09072166,0xb39a460a,0x6445c0dd,

0x586cdecf,0x1c20c8ae,0x5bbef7dd,0x1b588d40,0xccd2017f,0x6bb4e3bb,

0xdda26a7e,0x3a59ff45,0x3e350a44,0xbcb4cdd5,0x72eacea8,0xfa6484bb,

0x8d6612ae,0xbf3c6f47,0xd29be463,0x542f5d9e,0xaec2771b,0xf64e6370,

0x740e0d8d,0xe75b1357,0xf8721671,0xaf537d5d,0x4040cb08,0x4eb4e2cc,

0x34d2466a,0x0115af84,0xe1b00428,0x95983a1d,0x06b89fb4,0xce6ea048,

0x6f3f3b82,0x3520ab82,0x011a1d4b,0x277227f8,0x611560b1,0xe7933fdc,

0xbb3a792b,0x344525bd,0xa08839e1,0x51ce794b,0x2f32c9b7,0xa01fbac9,

0xe01cc87e,0xbcc7d1f6,0xcf0111c3,0xa1e8aac7,0x1a908749,0xd44fbd9a,

0xd0dadecb,0xd50ada38,0x0339c32a,0xc6913667,0x8df9317c,0xe0b12b4f,

0xf79e59b7,0x43f5bb3a,0xf2d519ff,0x27d9459c,0xbf97222c,0x15e6fc2a,

0x0f91fc71,0x9b941525,0xfae59361,0xceb69ceb,0xc2a86459,0x12baa8d1,

0xb6c1075e,0xe3056a0c,0x10d25065,0xcb03a442,0xe0ec6e0e,0x1698db3b,

0x4c98a0be,0x3278e964,0x9f1f9532,0xe0d392df,0xd3a0342b,0x8971f21e,

0x1b0a7441,0x4ba3348c,0xc5be7120,0xc37632d8,0xdf359f8d,0x9b992f2e,

0xe60b6f47,0x0fe3f11d,0xe54cda54,0x1edad891,0xce6279cf,0xcd3e7e6f,

0x1618b166,0xfd2c1d05,0x848fd2c5,0xf6fb2299,0xf523f357,0xa6327623,

0x93a83531,0x56cccd02,0xacf08162,0x5a75ebb5,0x6e163697,0x88d273cc,

0xde966292,0x81b949d0,0x4c50901b,0x71c65614,0xe6c6c7bd,0x327a140a,

0x45e1d006,0xc3f27b9a,0xc9aa53fd,0x62a80f00,0xbb25bfe2,0x35bdd2f6,

0x71126905,0xb2040222,0xb6cbcf7c,0xcd769c2b,0x53113ec0,0x1640e3d3,

0x38abbd60,0x2547adf0,0xba38209c,0xf746ce76,0x77afa1c5,0x20756060,

0x85cbfe4e,0x8ae88dd8,0x7aaaf9b0,0x4cf9aa7e,0x1948c25c,0x02fb8a8c,

0x01c36ae4,0xd6ebe1f9,0x90d4f869,0xa65cdea0,0x3f09252d,0xc208e69f,

0xb74e6132,0xce77e25b,0x578fdfe3,0x3ac372e6

};

}

#endregion

#region Conversions

//gets the first byte in a uint

private byte wordByte0(uint w)

{

return (byte)(w / 256 / 256 / 256 % 256);

}

//gets the second byte in a uint

private byte wordByte1(uint w)

{

return (byte)(w / 256 / 256 % 256);

}

//gets the third byte in a uint

private byte wordByte2(uint w)

{

return (byte)(w / 256 % 256);

}

//gets the fourth byte in a uint

private byte wordByte3(uint w)

{

return (byte)(w % 256);

}

//converts a byte array to a hex string

private string ByteToHex(byte[] bytes)

{

StringBuilder s = new StringBuilder();

foreach (byte b in bytes)

s.Append(b.ToString("x2"));

return s.ToString();

}

//converts a hex string to a byte array

private byte[] HexToByte(string hex)

{

byte[] r = new byte[hex.Length / 2];

for (int i = 0; i < hex.Length - 1; i += 2)

{

byte a = GetHex(hex[i]);

byte b = GetHex(hex[i + 1]);

r[i / 2] = (byte)(a \* 16 + b);

}

return r;

}

//converts a single hex character to it's decimal value

private byte GetHex(char x)

{

if (x <= '9' && x >= '0')

{

return (byte)(x - '0');

}

else if (x <= 'z' && x >= 'a')

{

return (byte)(x - 'a' + 10);

}

else if (x <= 'Z' && x >= 'A')

{

return (byte)(x - 'A' + 10);

}

return 0;

}

#endregion

}

}

# **Приложение Б**

**Алгоритм Twofish**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ZINISCourseProject

{

public class Twofish : SymmetricAlgorithm

{

public Twofish()

{

this.LegalKeySizesValue = new KeySizes[] { new KeySizes(128, 256, 64) }; // this allows us to have 128,192,256 key sizes

this.LegalBlockSizesValue = new KeySizes[] { new KeySizes(128, 128, 0) }; // this is in bits - typical of MS - always 16 bytes

this.BlockSize = 128; // set this to 16 bytes we cannot have any other value

this.KeySize = 128; // in bits - this can be changed to 128,192,256

this.Padding = PaddingMode.Zeros;

this.Mode = CipherMode.ECB;

}

/// <summary>

/// Creates an object that supports ICryptoTransform that can be used to encrypt data using the Twofish encryption algorithm.

/// </summary>

/// <param name="key">A byte array that contains a key. The length of this key should be equal to the KeySize property</param>

/// <param name="iv">A byte array that contains an initialization vector. The length of this IV should be equal to the BlockSize property</param>

public override ICryptoTransform CreateEncryptor(byte[] key, byte[] iv)

{

Key = key; // this appears to make a new copy

if (Mode == CipherMode.CBC)

IV = iv;

return new TwofishEncryption(KeySize, ref KeyValue, ref IVValue, ModeValue, TwofishBase.EncryptionDirection.Encrypting);

}

/// <summary>

/// Creates an object that supports ICryptoTransform that can be used to decrypt data using the Twofish encryption algorithm.

/// </summary>

/// <param name="key">A byte array that contains a key. The length of this key should be equal to the KeySize property</param>

/// <param name="iv">A byte array that contains an initialization vector. The length of this IV should be equal to the BlockSize property</param>

public override ICryptoTransform CreateDecryptor(byte[] key, byte[] iv)

{

Key = key;

if (Mode == CipherMode.CBC)

IV = iv;

return new TwofishEncryption(KeySize, ref KeyValue, ref IVValue, ModeValue, TwofishBase.EncryptionDirection.Decrypting);

}

/// <summary>

/// Generates a random initialization Vector (IV).

/// </summary>

public override void GenerateIV()

{

IV = new byte[16] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

}

/// <summary>

/// Generates a random Key. This is only really useful in testing scenarios.

/// </summary>

public override void GenerateKey()

{

Key = new byte[KeySize / 8];

// set the array to all 0 - implement a random key generation mechanism later probably based on PRNG

for (int i = Key.GetLowerBound(0); i < Key.GetUpperBound(0); i++)

{

Key[i] = 0;

}

}

/// <summary>

/// Override the Set method on this property so that we only support CBC and EBC

/// </summary>

public override CipherMode Mode

{

set

{

switch (value)

{

case CipherMode.CBC:

break;

case CipherMode.ECB:

break;

default:

throw (new CryptographicException("Specified CipherMode is not supported."));

}

this.ModeValue = value;

}

}

}

}

# **Приложение B**

**Алгоритм Threefish**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ZINISCourseProject

{

public class Threefish : SymmetricAlgorithm

{

private const int DefaultCipherSize = 256;

private ulong[] tweak;

public Threefish()

{

// Set up supported key and block sizes for Threefish

KeySizes[] supportedSizes =

{

new KeySizes(256, 512, 256),

new KeySizes(1024, 1024, 0)

};

LegalBlockSizesValue = supportedSizes;

LegalKeySizesValue = supportedSizes;

// Set up default sizes

KeySizeValue = DefaultCipherSize;

BlockSizeValue = DefaultCipherSize;

FeedbackSizeValue = DefaultCipherSize / 2;

// CBC is the default for the other symmetric ciphers in the standard library.

ModeValue = CipherMode.CBC;

}

public void SetTweak(ulong[] newTweak)

{

if (newTweak.Length != 2)

throw new ArgumentException("Tweak must be an array of two unsigned 64-bit integers.");

tweak = newTweak;

}

public override ICryptoTransform CreateDecryptor(byte[] rgbKey, byte[] rgbIV)

{

var tsm = new ThreefishTransform(rgbKey, rgbIV, FeedbackSize,

ThreefishTransformMode.Decrypt, ModeValue, PaddingValue);

if (tweak != null)

tsm.InternalSetTweak(tweak);

return tsm;

}

public override ICryptoTransform CreateEncryptor(byte[] rgbKey, byte[] rgbIV)

{

var tsm = new ThreefishTransform(rgbKey, rgbIV, FeedbackSize,

ThreefishTransformMode.Encrypt, ModeValue, PaddingValue);

if (tweak != null)

tsm.InternalSetTweak(tweak);

return tsm;

}

public override void GenerateIV()

{ IVValue = GenerateRandomBytes(BlockSizeValue / 8); }

public override void GenerateKey()

{ KeyValue = GenerateRandomBytes(KeySizeValue / 8); }

private static byte[] GenerateRandomBytes(int amount)

{

var rngCrypto = new RNGCryptoServiceProvider();

var bytes = new byte[amount];

rngCrypto.GetBytes(bytes);

return bytes;

}

}

}