**Федеральное Государственное Бюджетное**

**Образовательное Учреждение**

**Национальный Исследовательский Университет «МЭИ»**

Институт Автоматики и Вычислительной Техники

Кафедра Прикладной Математики

**«Программная реализация криптоалгоритма SAFER+»**

**Курсовой проект**

по учебной дисциплине

«Защита данных»

Выполнил:

студент группы А-05-16

Зязиков Мохаммад Ибрагимович

Преподаватель:

доцент Хорев Павел Борисович

Москва, 2019

Содержание

1. Введение……………………………………………………………………….3
2. Теоретические сведения……………………………………………………...4
3. Программная реализация……………………………………………………..7
4. Тестирование………………………………………………………………...15
5. Список используемых источников…………………………………………21
6. Заключение…………………………………………………………………...21
7. Приложения………………………………………………………………….22

Введение

Цель работы – разработать и реализовать программное приложение, позволяющее шифровать и расшифровывать файлы и тестовые сообщения с помощью криптоалгоритма SAFER+. Программа должна шифровать и расшифровывать файлы любого типа и текстовые сообщения на ключе, выводимом из парольной фразы с регулируемым пользователем длиной и сложностью.

Задачи, которые должны быть решены для достижения цели работы:

* Проектирование и реализация интерфейса программного продукта
* Реализация взаимодействия модулей программы
* Реализация криптоалгоритма SAFER+
* Тестирование и отладка программы

**Глава I. История и описание алгоритма**

1. 1. История создания

Существует несколько вариантов шифра, отличающихся друг от друга длиной ключа шифрования и размерами блоков исходного текста.

Первая разновидность [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) — SAFER K-64 была разработана Джэймсом Мэсси для калифорнийской корпорации [«Cylinc»](https://ru.wikipedia.org/w/index.php?title=Cylinc&action=edit&redlink=1) в [1993 году](https://ru.wikipedia.org/wiki/1993_%D0%B3%D0%BE%D0%B4). Опубликованный в том же году, алгоритм имел блок и [ключ шифрования](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) длиной в 64 [бита](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82). Для него рекомендовалось использовать 6 раундов шифрования. Однако, из-за необходимости увеличить длину ключа до 128 бит (так как была обнаружена слабость в первоначальном варианте алгоритма), Мэсси разработал новый вариант шифра SAFER K-128, который был опубликован на следующий год после SAFER K-64. Новый алгоритм включал в себя расписание ключей, разработанное министерством внутренних дел [Сингапура](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D0%B3%D0%B0%D0%BF%D1%83%D1%80), и в дальнейшем использовался им для различных целей. Также для этого алгоритма рекомендовалось использовать 10 (максимум 12) раундов шифрования.

Спустя некоторое время в первых вариантах алгоритма выявились некоторые слабости, обнаруженные [Ларсом Кнудсеном](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D1%83%D0%B4%D1%81%D0%B5%D0%BD,_%D0%9B%D0%B0%D1%80%D1%81) и [Шоном Мёрфи](https://ru.wikipedia.org/wiki/%D0%9C%D1%91%D1%80%D1%84%D0%B8,_%D0%A8%D0%BE%D0%BD_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84)). Это повлекло за собой создание новых версий алгоритма, названных SAFER SK-64 и SAFER SK-128, в которых расписание ключей было изменено в соответствии со схемой, предложенной Кнудсеном. Также был разработан вариант с длиной ключа, уменьшенной до 40 бит — SAFER SK-40. Сокращение «SK» в названии алгоритмов расшифровывается как «Strengthened Key schedule» (Усиленное расписание ключей). Для новых вариантов шифра предлагалось использовать не 6, а по крайней мере 8 (максимум 10) раундов шифрования.

Алгоритм SAFER+ был разработан в [1998 году](https://ru.wikipedia.org/wiki/1998_%D0%B3%D0%BE%D0%B4) калифорнийской корпорацией Cylinc совместно с [Армянской академией наук](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F_%D0%BD%D0%B0%D1%83%D0%BA_%D0%90%D1%80%D0%BC%D0%B5%D0%BD%D0%B8%D0%B8) для участия в [конкурсе AES](https://ru.wikipedia.org/wiki/AES_(%D0%BA%D0%BE%D0%BD%D0%BA%D1%83%D1%80%D1%81)), на котором прошёл лишь первый отборочный тур. Данный шифр имеет входной блок длиной 128 бит и размер ключа 128, 192 или 256 бит.

Последней из созданных разновидностей алгоритма SAFER является SAFER++, разработанный Мэсси в [2000 году](https://ru.wikipedia.org/wiki/2000_%D0%B3%D0%BE%D0%B4) и ставший дальнейшим развитием алгоритма SAFER+. Алгоритм принял участие в европейском конкурсе алгоритмов [NESSIE](https://ru.wikipedia.org/wiki/NESSIE), где был представлен в двух вариантах: шифр с 64-битным блоком и 128-битным блоком. Он прошёл во вторую фазу конкурса, но не был выбран в набор рекомендуемых NESSIE криптографических примитивов. Эксперты сочли, что шифр слишком медленный на всех машинах, кроме 8-битных (таких как [смарт-карты](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82-%D0%BA%D0%B0%D1%80%D1%82%D0%B0)), а запас безопасности шифра слишком мал.

Алгоритмы SAFER не являются частной собственностью и не защищены авторскими правами, то есть могут быть использованы без каких-либо ограничений. Поскольку они целиком состоят из простых байтовых операций (за исключением поворота байтов при генерации ключей), эти алгоритмы могут быть реализованы процессорами с малой разрядностью.

Ниже приведена сводная таблица всех существующих вариантов шифра SAFER

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **название** | **англ.** | **дата создания** | **длина блока** | **длина ключа** | **число раундов** |
| *SAFER K-64* | key 64 bit | [1993](https://ru.wikipedia.org/wiki/1993) | 64 | 64 | 6 |
| *SAFER K-128* | key 128 bit | [1995](https://ru.wikipedia.org/wiki/1995) | 64 | 128 | 10 (максимум 12) |
| *SAFER SK-64* | Strengthened Key schedule, 64 bit | [1995](https://ru.wikipedia.org/wiki/1995) | 64 | 64 | 8 (минимум 6, максимум 10) |
| *SAFER SK-128* | Strengthened Key schedule, 128 bit | [1995](https://ru.wikipedia.org/wiki/1995) | 64 | 128 | 10 (максимум 12) |
| *SAFER SK-40* | Strengthened Key schedule, 40 bit | [1995](https://ru.wikipedia.org/wiki/1995) | 64 | 40 |  |
| *SAFER+* | SAFER Plus | [1998](https://ru.wikipedia.org/wiki/1998) | 128 | 128, 192, 256 | 8, 12, 16 |
| *SAFER++* | SAFER Plus Plus | [2000](https://ru.wikipedia.org/wiki/2000) | 64, 128 | 128, 256 | 7, 10 |

Рис. 1.1.1. Сводная таблица всех существующих вариантов шифра

1. 2. Описание криптоалгоритма

**Алгоритм шифрования**

По структуре алгоритм *SAFER+* напоминает *SAFER K-64*. Он состоит из тех же основных этапов, несколько отличающихся по своей структуре. На каждом раунде работы алгоритма сначала происходит подмешивание одного подключа, после этого байты проходят через блоки нелинейной замены, затем подмешивается второй подключ и происходит линейное перемешивание байтов. Подключи последовательно генерируются с использованием входного ключа. Ниже приведено более подробное описание работы одной итерации (*i* — номер итерации):

1. **Наложение ключа K 2 i − 1 {\displaystyle K\_{2i-1}}** : байты входного блока складываются с байтами ключа K 2 i − 1 {\displaystyle K\_{2i-1}} , причём используется сложение по модулю 2 для байтов с номерами 1, 4, 5, 8, 9, 12, 13 и 16, и сложение по модулю 256 для байтов с номерами 2, 3, 6, 7, 10, 11, 14 и 15.
2. **Нелинейное преобразование**: к байтам с номерами 1, 4, 5, 8, 9, 12, 13 и 16 применяется операция 45 x ( m o d 257 ) {\displaystyle 45^{x}(mod257)}  (причём 45 128 ( m o d 257 ) = 256 {\displaystyle 45^{128}(mod257)=256} заменяется нулём). К байтам с номерами 2, 3, 6, 7, 10, 11, 14 и 15 применяется операция l o g 45 ( x ) {\displaystyle log\_{45}(x)}  (причём l o g 45 ( 0 ) = 128 {\displaystyle log\_{45}(0)=128} ). результаты действия этих операций как и для других разновидностей алгоритма SAFER на практике часто хранят в специальных таблицах. В данном случае для этого требуется 512 байт.
3. **Наложение ключа K 2 i {\displaystyle K\_{2i}}** : байты входного блока складываются с байтами ключа K 2 i {\displaystyle K\_{2i}} , но в отличие от п.1 операции сложения по модулю 2 и по модулю 256 меняются местами.
4. **Линейное преобразование**: умножение 16-байтного блока данных справа на специальную [невырожденную матрицу](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B2%D1%8B%D1%80%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) *M* (все операции при этом байтовые и производятся [по модулю 256](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0)). Умножение на эту матрицу эквивалентно четырём уровням преобразования [PHT](https://ru.wikipedia.org/wiki/PHT), между которыми выполняются некоторые байтовые перестановки. Стоит отметить, что эта часть алгоритма является наиболее громоздкой с вычислительной точки зрения.

После проведения *r* раундов шифрования производится подмешивание ключа K 2 r + 1 {\displaystyle K\_{2r+1}} , аналогичное подмешиванию ключей K 2 i − 1 {\displaystyle K\_{2i-1}} .



Рис. 1.2.2 Раунд шифрования алгоритма SAFER+.

**Алгоритм расшифрования**

Операции в алгоритме расшифрования подобны операциям шифрования и производятся в обратном порядке. Разница состоит в следующем:

1. Вместо [матрицы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) M {\displaystyle M}  умножение происходит с обратной ей матрицей M − 1 {\displaystyle M^{-1}} ;
2. Все операции сложения по модулю 256 заменяются операциями вычитания;
3. Операции 45 x  (mod  257 ) {\displaystyle 45^{x}{\mbox{ (mod }}257)}  и l o g 45 x {\displaystyle log\_{45}x}  (являющиеся обратными друг другу) меняются местами.

**Генерация ключей**

Излагаемый алгоритм применим для входных ключей длиной в 128, 192 и 256 бит. Первый подключ  представляет собой первые 16 байт входного ключа. Генерация остальных ключей производится следующим образом: сначала исходный ключ целиком записывается в ключевой [регистр](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_(%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)) длиной на 1 байт длиннее самого ключа (то есть длина регистра равна для разных входных ключей 17, 25 или 33 байтам). Все байты ключа [суммируются по модулю 2](https://ru.wikipedia.org/wiki/XOR) поразрядно, результат записывается в последний байт регистра. Для получения каждого следующего ключа над содержимым регистра выполняются следующие операции (для *i* от 2 до 2*r*+1):

1. Содержимое байтов ключевого регистра циклически сдвигается влево на 3 позиции (сдвиг происходит внутри байтов в отдельности, а не регистра как целого);
2. Из регистра выбираются 16 байт. При этом для ключа K i {\displaystyle K\_{i}}  выбираются байты регистра начиная с *i*-го и далее по циклу.
3. Отобранные 16 байт складываются по модулю 256 с байтами слова смещения B i {\displaystyle B\_{i}} . Результат сложения и будет являться подключом K i {\displaystyle K\_{i}} .

Слова смещения B i {\displaystyle B\_{i}}  — это 16-байтные константы, вычисляемые по следующей формуле: B i , j = { 45 45 17 i + j  mod  257  mod  257 , i  = 2,3,… 17;  j  = 1,3,… 16 45 17 i + j  mod  257 , i  = 18,19,… 33;  j  = 1,3,… 16 {\displaystyle B\_{i,j}={\begin{cases}45^{45^{17i+j}{\mbox{ mod }}257}{\mbox{ mod }}257,&i{\mbox{ = 2,3,… 17; }}j{\mbox{ = 1,3,… 16}}\\45^{17i+j}{\mbox{ mod }}257,&i{\mbox{ = 18,19,… 33; }}j{\mbox{ = 1,3,… 16}}\end{cases}}}

****

 — *j*-й байт *i*-го слова смещения. Если B i , j = 256 {\displaystyle B\_{i,j}=256}  то этот байт заменяется на 0.

Понятно, что поскольку для различных длин ключей количество итераций шифрования различно (и равно 8, 12 и 16 для ключей длиной 128, 192 и 256 бит соответственно), то и использованы будут не все блоки смещения. Так, при длине ключа в 128 бит будут использованы только  B 2 {\displaystyle B\_{2}} , … B 17 {\displaystyle B\_{17}} ,  для ключа в 192 бита — B 2 {\displaystyle B\_{2}} , … B 25 {\displaystyle B\_{25}} , а для ключа в 256 бит — все слова смещения.

**Глава II. Программная реализация**

1. 1. Графический интерфейс

Основной интерфейс приложения сгруппирован в главном окне. Визуально его можно разделить на 3 части, относящиеся к выбору файла, выбору ключа и работе с текстовым сообщением.

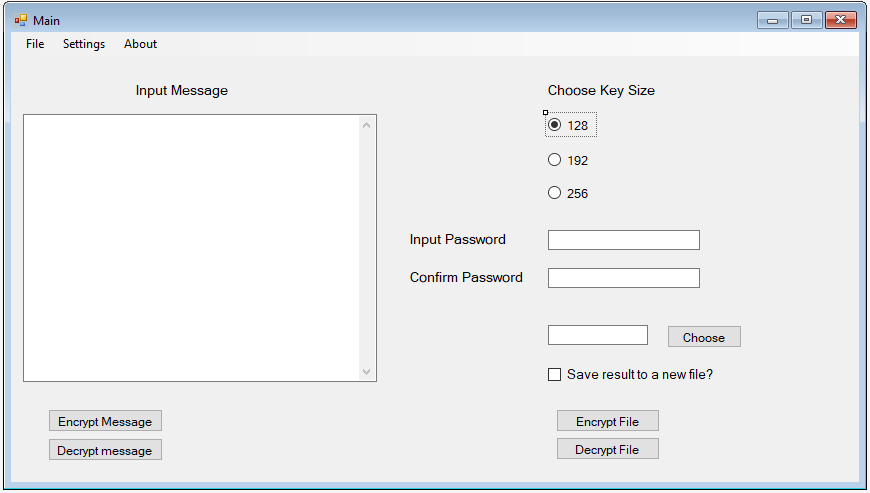


Рис. 2.1.1. Главная форма

### 2.1.1. Выбор ключа.

Генерация секретного ключа осуществляется на основе парольной фразы. Введенная парольная фраза считывается в кодировке UTF-8, после чего символы преобразуются в их численные значения. Полученный массив чисел будет использоваться в качестве ключа.

На минимальную длину парольной фразы установлены следующие ограничения:

* При выборе длины шифрования в 128 бит минимальная длина парольной фразы составляет 4 символа.
* При выборе длины шифрования в 192 бита минимальная длина парольной фразы составляет 6 символов.
* При выборе длины шифрования в 256 бит минимальная длина парольной фразы составляет 8 символов.

Дополнительные ограничения на длину и сложность парольной фразы могут быть установлены в окне настроек.

При шифровании текстовых сообщений и файлов Парольная фраза должна быть подтверждена повторным вводом в текстовое окно «Подтверждение». При расшифровании поле можно оставить пустым.

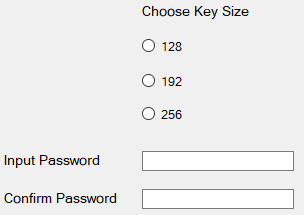


Рис. 2.1.2. Секция ввода пароля

### 2.1.2. Работа с текстовым сообщением

Программа поддерживает следующие возможности:

* Ручной ввод сообщения для шифрования (расшифрования). Введенное сообщение считывается в кодировке UTF-8.
* Загрузка сообщения из текстового файла посредством диалогового окна (Пункт меню Файл->Открыть файл с текстовым сообщением). Содержимое текстового файла загружается в поле для ввода текста.
* Сохранение сообщения в текстовый файл посредством диалогового окна (Пункт меню Файл->Сохранить текстовое сообщение в файл). В текстовый файл загружается содержимое поля для ввода текста.

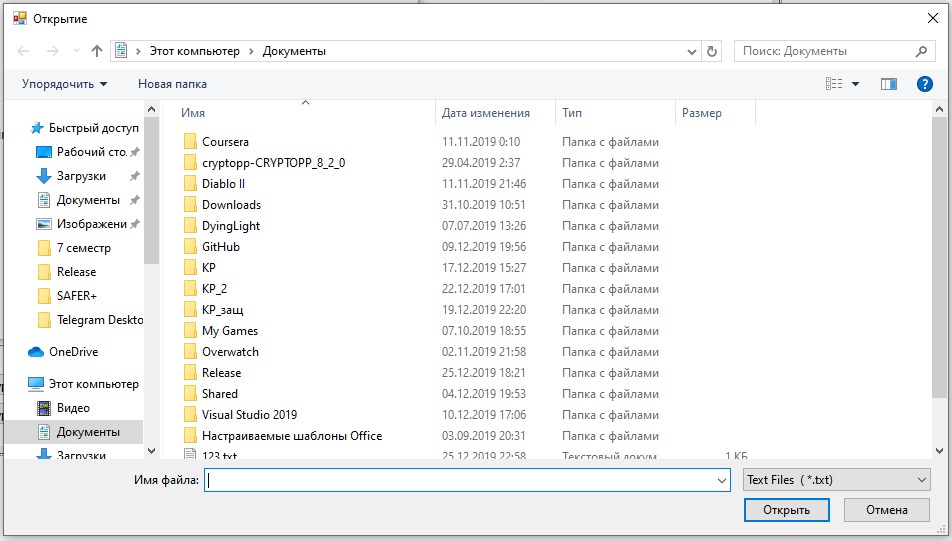


Рис. 2.1.3 Диалог выбора файла

### 2.1.3. Работа с файлом.

Программа поддерживает следующие возможности:

* Шифрование (расшифрование) выбранного файла (кнопки «Зашифровать файл» и «Расшифровать файл» соответственно) на бинарном уровне.
* Создание зашифрованной (расшифрованной) копии выбранного файла (Галочка «Сохранить результат в новый файл?»). Диалог сохранения зашифрованной (расшифрованной) копии автоматически вызывается при успешном завершении операции шифрования (расшифрования).

Если галочка не установлена, то исходный файл будет перезаписан.

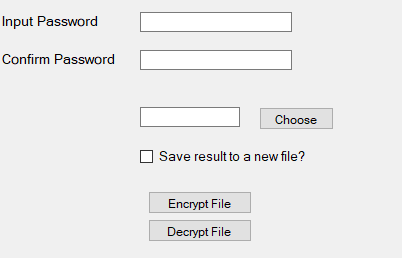


Рис 2.1.4 Работа с файлами

### Настройки программы.

Данные функции располагаются во вкладке меню Настройки



Рис. 2.1.5 Меню Settings

Рисунок 2.7 Меню «Настройки»

С их помощью можно:

* Установить максимальную длину текстового сообщения. (Или снять ограничения вовсе)
* Установить максимальный размер обрабатываемых файлов. (Или снять ограничения вовсе)
* Установить минимальную длину парольной фразы (на минимально возможную длину установлены ограничения связанные с длиной ключа)
* Установить сложность парольной фразы.

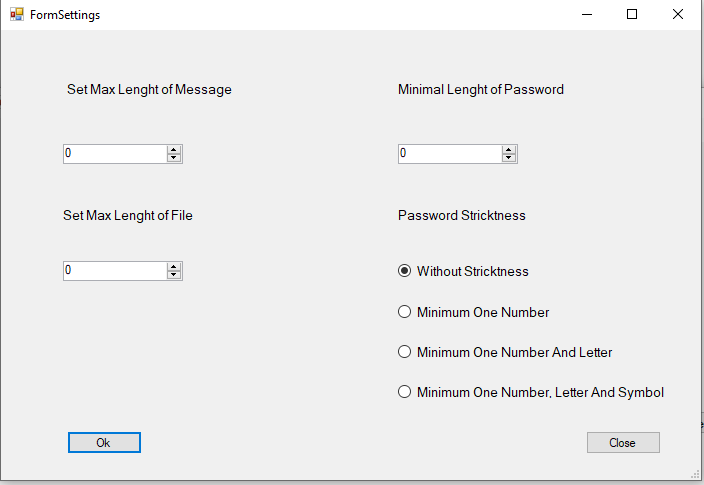


Рис. 2.1.6 Окно настроек

### 2.1.5. Окно «О программе».

Данное окно находится во вкладке меню «О программе».



Рис. 2.1.7 меню About

В нем содержится сведения о программе.

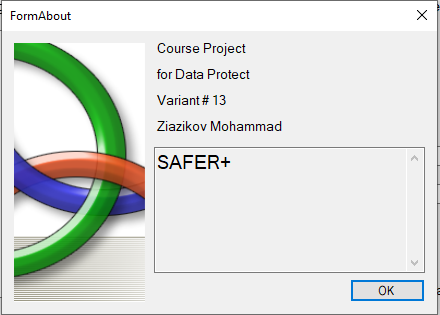


Рисунок 2.11 Окно About

## 2.2. Результаты проектирования алгоритма шифрования.

Алгоритм шифрования реализован через DLL-библиотеку, созданную на основе открыто распространяемой реализации алгоритма на языке С написанной доктором Брайаном Глэдмэном (Brian Gladman) , подключаемую к проекту.

Для подачи входных данных в алгоритм используются следующие функции:

* extern "C" \_\_declspec(dllexport) void encrypt\_block(unsigned long key[8], unsigned long key\_length, unsigned long block[4]) – Дешифрует блок на выбранном ключе.
* extern "C" \_\_declspec(dllexport) void decrypt\_block(unsigned long key[8], unsigned long key\_length, unsigned long block[4]) – Шифрует блок на выбранном ключе.

Во внутренней работе библиотеки используются следующие функции:

* unsigned long \*set\_key(const unsigned long in\_key[], const unsigned long key\_len) – процедура расширения ключа.
* void do\_fr(u1byte x[16], u1byte \*kp) и void encrypt(const unsigned long in\_blk[4], unsigned long out\_blk[4]) – раунд шифрования и функция шифрования блока соответственно
* void do\_ir(u1byte x[16], u1byte \*kp) и void decrypt(const unsigned long in\_blk[4], unsigned long out\_blk[4]) – раунд расшифрования и функция расшифрования блока соответственно.

Интерфейс проекта реализован на языке C#, функции обрабатывающие текст/файл и передающие текст алгоритму (получающие шифротекст) включены в функции события нажатия соответствующих кнопок.

При инициации процедуры шифрования/расшифрования текста (нажатию кнопки «Зашифровать сообщение» или «Расшифровать сообщение» соответственно) коды символов, блоками по четыре символа, подаются на вход алгоритма шифрования (коды четырёх символов составляют один блок поданный на вход алгоритма шифрования). На вывод подаётся набор чисел отвечающих зашифрованным цифровым кодам символов.

При инициации процедуры шифрования/расшифрования файла (нажатию кнопки «Зашифровать файл» или «Расшифровать файл» соответственно) формируется массив байтов копирующий содержимое файла, затем байты подаются на вход алгоритма (16 байт составляют один блок поданный на вход алгоритма). На вывод подаётся набор зашифрованных байт.

Генерация ключа из парольной фразы происходит следующим образом: Коды символов с 1 по 4 в режиме «128-битный ключ» парольной фразы (с 1 по 6 в режиме «192-бит» и с 1 по 8 бит в режиме «256-бит» соответственно) в кодировке UTF-8 формата Decinal. Коды последующих символов складываются с кодом последнего символа ключа до достижения максимальной длины парольной фразы (20 символов).

# 

# **Глава Ш. Результаты тестирования разработанной программы.**

## 

## 3.1. Тестирование шифрования/расшифрования текстовых сообщений

Программа разрабатывалась и отлаживалась на языке программирования C#, ориентированного на .NET Framework 3.5 в среде программирования Microsoft Visual Studio 2015.

Сравним работу программы с реализацией алгоритма [6] (Для тестирования была написана небольшая программа реализующая шифрование четырёх чисел на выбранном ключе)

**Пример**

Результат шифрования сообщения «абвг» - (Коды символов в кодировке UTF-8 в представлении Decimal – 1072,1073,1074,1075) на основе парольной фразы : «3 4 2 1» (Коды в UTF-8 в представлении Decimal - 51 52 50 49)

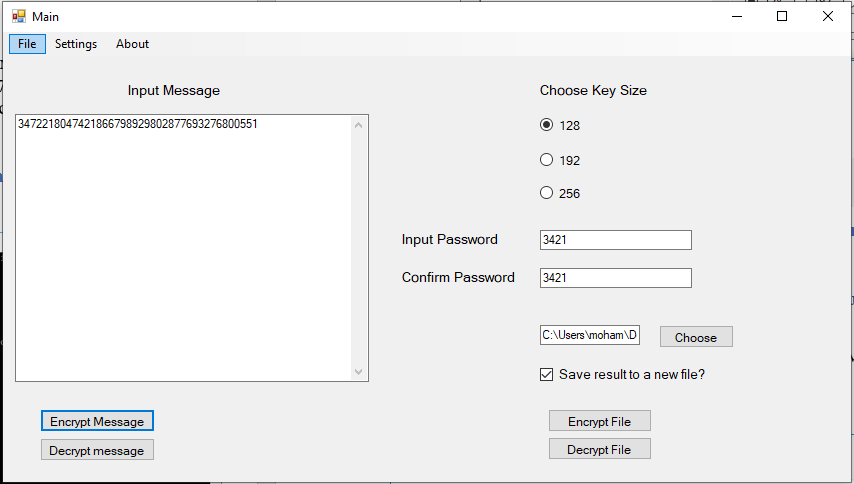


Рисунок 3.1 Результат шифрования в программе

С учётом особенностей работы программы – шифрование было произведено корректно.

## 3.2. Тестирование шифрования/расшифрования файлов

**Пример 1**

Шифрование 30-байтного текстового файла на 256-битном ключе на основе парольной фразы: «abCdeGe8»

Содержимое текстового файла:

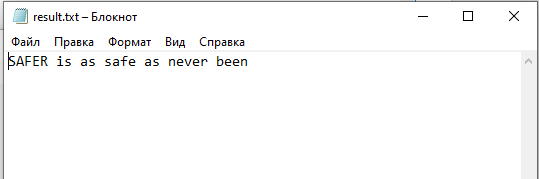


Рисунок 3.2 Пример 1 – Содержимое файла

Результат шифрования:

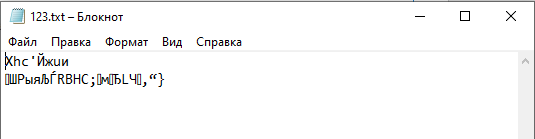


Рисунок 3.3 Пример 1 – Результат шифрования

Результат расшифрования

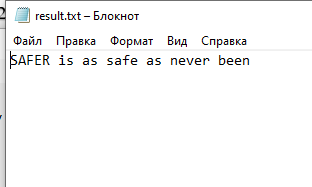


Рисунок 3.4 Пример 1 – Результат расшифрования

**Пример 2**

Шифрование 231-килобайтного файла расширения .jpg на 192-битном ключе с созданием копии на основе парольной фразы: «tiger»

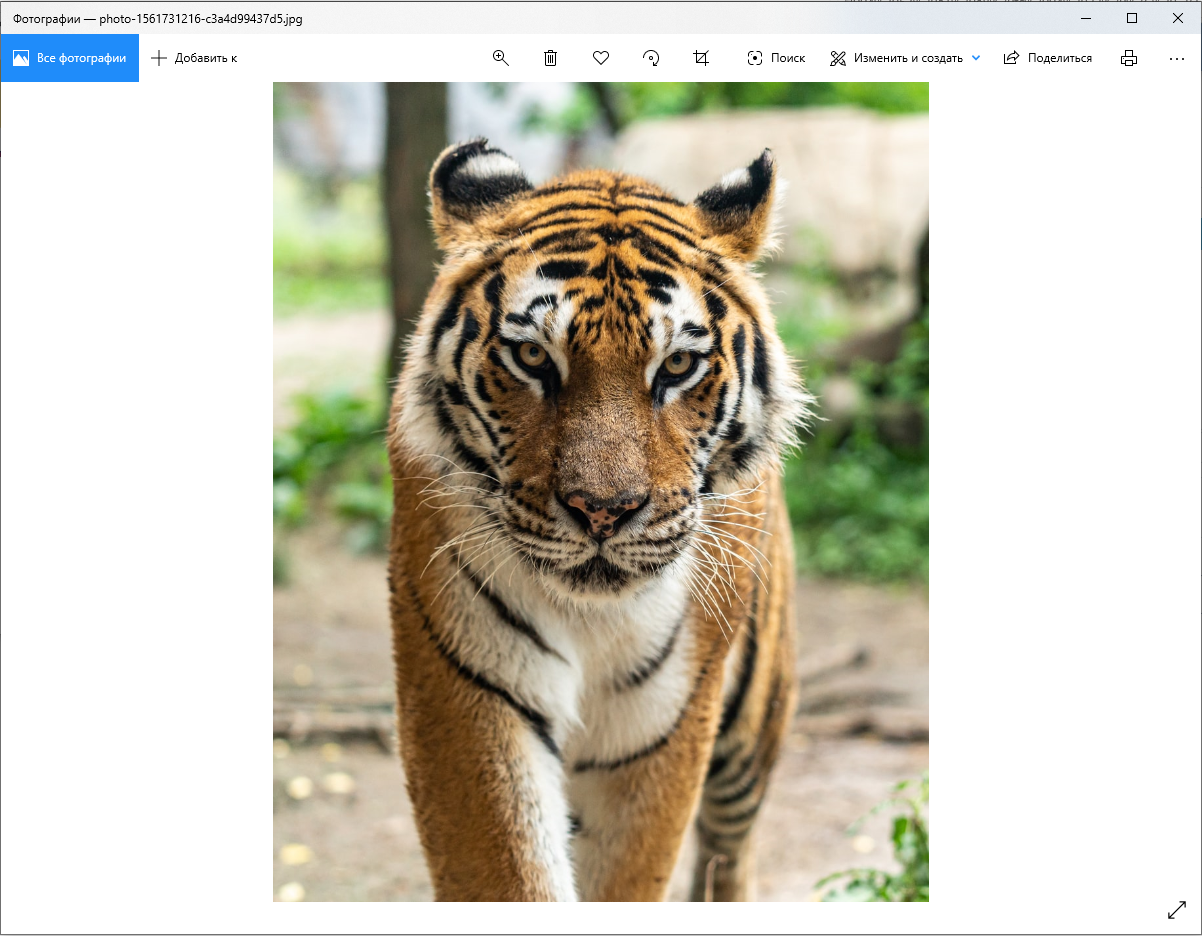


Рисунок 3.5 Исходный файл

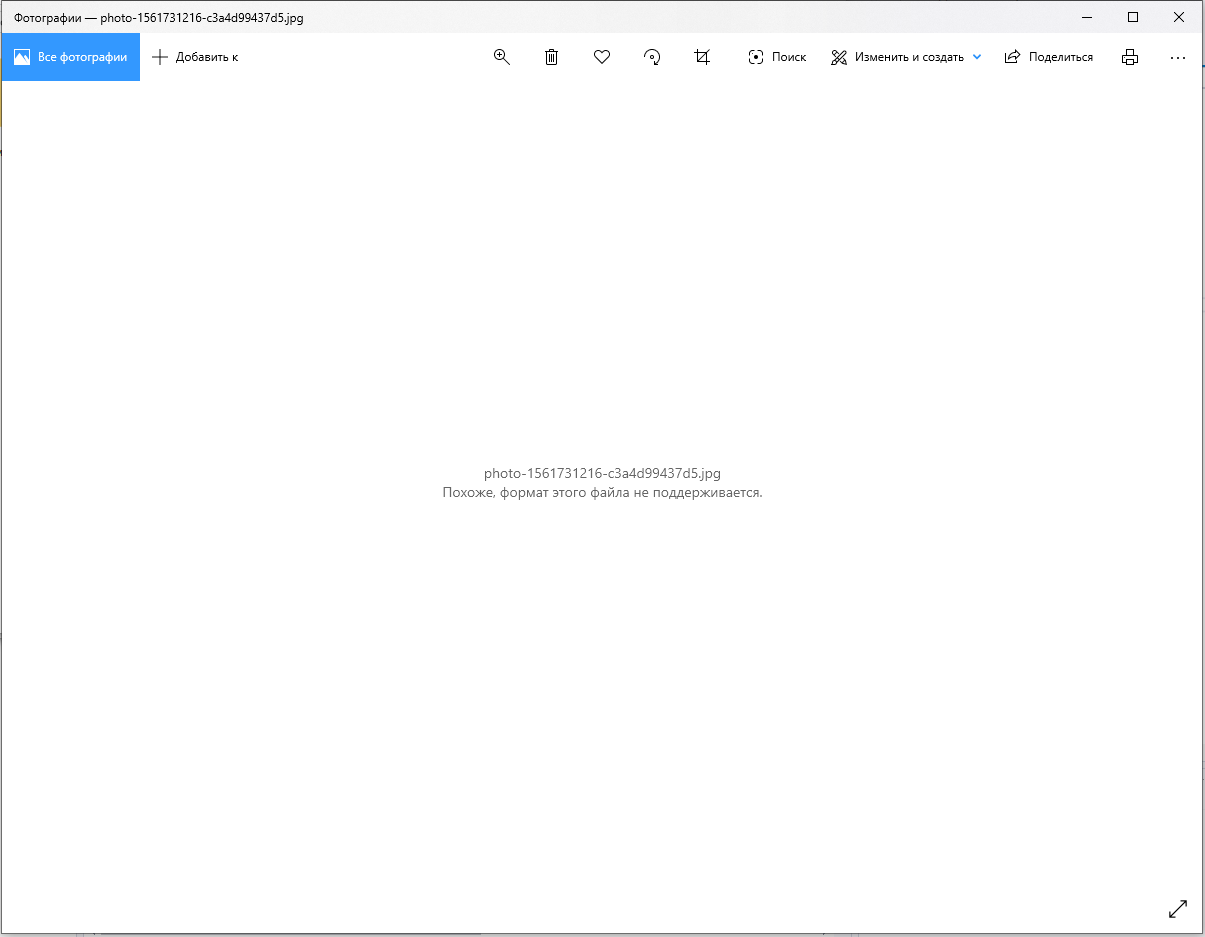


Рисунок 3.6 Файл после шифровки

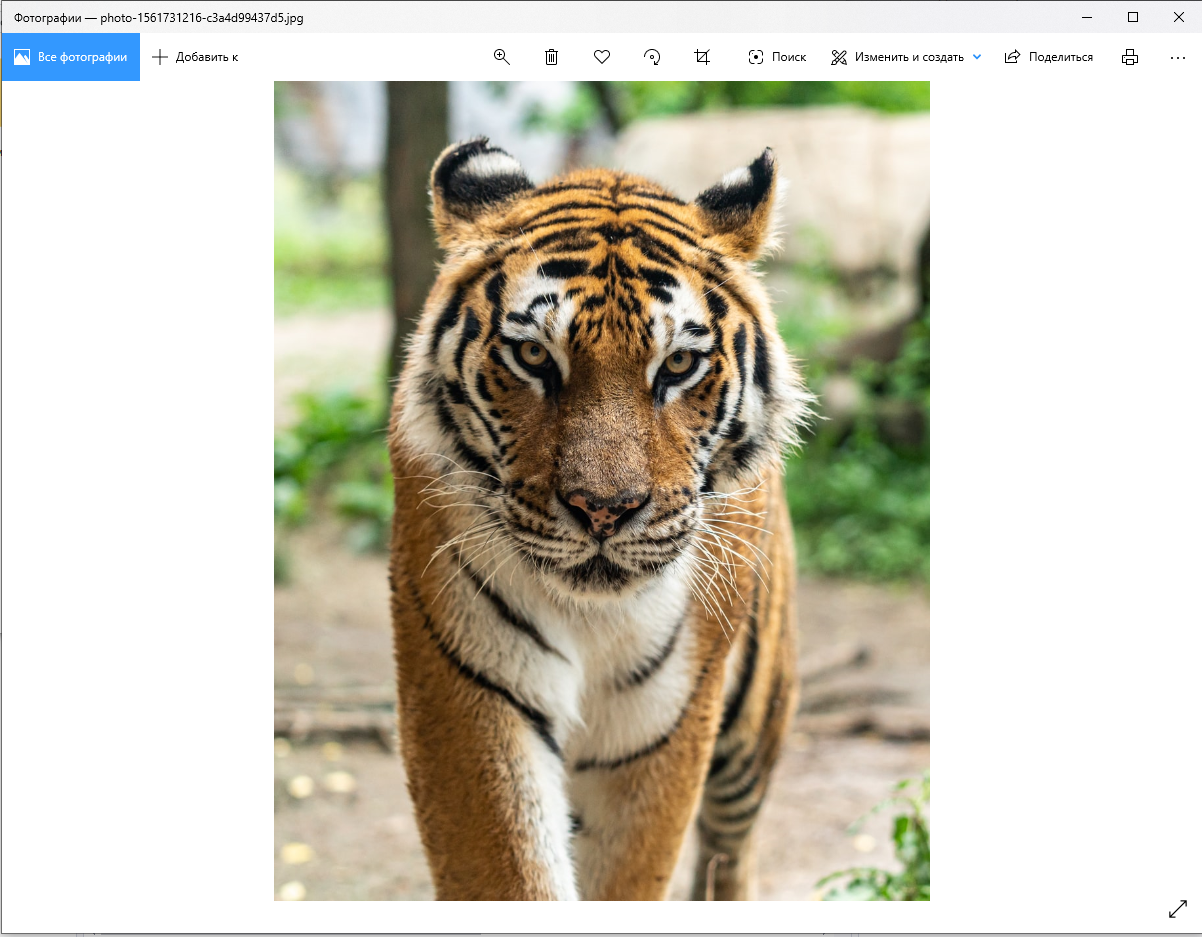


Рисунок 3.7 Файл после расшифровки

Список используемых источников

<https://m.habr.com/ru/post/279197/> - Конечное поле GF(256) и немного магии

https://ru.wikipedia.org/wiki/Конечное\_поле

<https://vuzlit.ru/1017403/algoritm_safer_safer> - Алгоритмы SAFER+, SAFER++

<https://ru.wikipedia.org/wiki/SAFER#SAFER>+ - Семейство криптоалгоритмов SAFER

# <http://book.itep.ru/6/safr_648.htm> - Алгоритм шифрования SAFER

#### Семенов Ю.А. (ИТЭФ-МФТИ) Yu. Semenov (ITEP-MIPT)

<https://ru.wikipedia.org/wiki/SP-сеть> - Sp - сеть и ее принцип работы

<http://crypto.pp.ua/2010/12/algoritm-safer/> - Описание алгоритма SAFER+

# Заключение.

В результате выполнения данной работы было разработано программное обеспечение, с возможностью шифрования/расшифровывания с использованием алгоритма SAFER+, как выбираемых файлов любого типа, так и вводимых текстовых сообщений на ключе, выводимом из парольной фразы с регулируемой пользователем минимальной длиной и сложностью.

Приложение

Main form:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

using System.IO;

//compile with: /unsafe;

namespace SAFER\_OVERPOWERED

{

public partial class Form1 : Form

{

uint key\_length = 128;

public Form1()

{

InitializeComponent();

}

[DllImport("Safer+DLL.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Unicode, SetLastError = true)]

[return: MarshalAs(UnmanagedType.I4)]

private static extern void encrypt\_block(uint[] key, uint key\_length, uint[] input);

[DllImport("Safer+DLL.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Unicode, SetLastError = true)]

[return: MarshalAs(UnmanagedType.I4)]

private static extern void decrypt\_block(uint[] key, uint key\_length, uint[] input);

private void radioButtonKeySize128\_CheckedChanged(object sender, EventArgs e)

{

key\_length = 128;

if (SettingsKeeped.MinKeylength < 4)

{

SettingsKeeped.MinKeylength = 4;

}

}

private void radioButtonKeySize192\_CheckedChanged(object sender, EventArgs e)

{

key\_length = 192;

if (SettingsKeeped.MinKeylength < 6)

{

SettingsKeeped.MinKeylength = 6;

}

}

private void radioButtonKeySize256\_CheckedChanged(object sender, EventArgs e)

{

key\_length = 256;

if (SettingsKeeped.MinKeylength < 8)

{

SettingsKeeped.MinKeylength = 8;

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void settingsToolStripMenuItem\_Click(object sender, EventArgs e)

{

FormSettings SettingsForm = new FormSettings();

SettingsForm.ShowDialog();

}

private void buttonFile\_Click(object sender, EventArgs e)

{

OpenFileDialog dialog = new OpenFileDialog();

if (dialog.ShowDialog() == DialogResult.OK)

{

textBoxChooseFile.Text = dialog.FileName;

}

}

private void openFileWithMessageToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

string path;

OpenFileDialog dialog = new OpenFileDialog();

dialog.Filter = "Text Files | \*.txt";

dialog.DefaultExt = "txt";

if (dialog.ShowDialog() == DialogResult.OK)

{

path = dialog.FileName;

textBoxMessageInput.Text = File.ReadAllText(path);

}

}

private void saveMessageToFileToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

string path;

SaveFileDialog dialog = new SaveFileDialog();

dialog.Filter = "Text Files | \*.txt";

dialog.DefaultExt = "txt";

if (dialog.ShowDialog() == DialogResult.OK)

{

path = dialog.FileName;

File.WriteAllText(path, textBoxMessageInput.Text);

}

}

private void aboutToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

FormAbout ab = new FormAbout();

ab.ShowDialog();

}

private void exitToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

Application.Exit();

}

private void buttonEncryptMessage\_Click\_1(object sender, EventArgs e)

{

//Шифровать сообщение

uint[] key = new uint[8]; uint[] input\_block = new uint[4];

if (textBoxPassword.Text.Length < SettingsKeeped.MinKeylength)

{

MessageBox.Show("Password must be " + SettingsKeeped.MinKeylength + " symbols longer");

return;

}

if (PasswordPhrase.isComplexityCorrect(textBoxPassword.Text) != 0)

{

MessageBox.Show("Password doesn't fit stricktness");

return;

}

if (textBoxPassword.Text != textBoxConfirmPassword.Text)

{

MessageBox.Show("Confirmation is not correct ");

return;

}

else

{

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

{

if (i < key\_length / 32)

key[i] = Convert.ToUInt32(textBoxPassword.Text[i]);

else

key[3] += Convert.ToUInt32(textBoxPassword.Text[i]);

}

string bufferstr = textBoxMessageInput.Text;

if (SettingsKeeped.Maxsize != 0 && (SettingsKeeped.Maxsize < bufferstr.Length))

{

MessageBox.Show("Message is too big");

return;

}

textBoxMessageInput.Clear();

while (bufferstr.Length > 0)

{

if (bufferstr.Length % 4 != 0)

{

for (int i = 0; i < bufferstr.Length % 4; i++)

bufferstr += " ";

}

char[] buffer = bufferstr.Take(4).ToArray();

bufferstr = bufferstr.Remove(0, 4);

for (int i = 0; i < 4; i++)

{

input\_block[i] = Convert.ToUInt32(buffer[i]);

}

//Encoding.Default.GetBytes()массив байтовы скормить алгоритму

encrypt\_block(key, key\_length, input\_block);

foreach (uint i in input\_block)

{

if (i < 1000000000)

{

for (int j = 0; j < (10 - i.ToString().Length); j++)

textBoxMessageInput.Text += "0";

}

textBoxMessageInput.Text += i;

}

}

}

}

private void buttonDecryptMessage\_Click\_1(object sender, EventArgs e)

{

//Дешифровать сообщение

uint[] key = new uint[8]; uint[] input\_block = new uint[4];

if (textBoxPassword.Text.Length < SettingsKeeped.MinKeylength)

{

MessageBox.Show("Password must be " + SettingsKeeped.MinKeylength + " symbols longer");

return;

}

if (PasswordPhrase.isComplexityCorrect(textBoxPassword.Text) != 0)

{

MessageBox.Show("Password doesn't fit stricktness");

return;

}

else

{

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

{

if (i < key\_length / 32)

key[i] = Convert.ToUInt32(textBoxPassword.Text[i]);

else

key[3] += Convert.ToUInt32(textBoxPassword.Text[i]);

}

string bufferstr = textBoxMessageInput.Text;

if ((Convert.ToInt32(bufferstr.Length) % 4 != 0))

{

MessageBox.Show("Encrypted message is damaged");

return;

}

if (SettingsKeeped.Maxsize != 0 && (SettingsKeeped.Maxsize < bufferstr.Length))

{

MessageBox.Show("Message is too big");

return;

}

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

{

if (!(bufferstr[i] >= '0' && bufferstr[i] <= '9'))

{

MessageBox.Show("Only digits are allowed in encrypted");

return;

}

}

textBoxMessageInput.Clear();

while (bufferstr.Length > 0)

{

for (int i = 0; i < 4; i++)

{

string buffer = bufferstr.Substring(0, 10);

bufferstr = bufferstr.Remove(0, 10);

UInt32.TryParse(buffer, out input\_block[i]);

}

// input\_block = richTextBox1.Text.Split(' ').Select(s => uint.Parse(s)).ToArray();

decrypt\_block(key, key\_length, input\_block);

foreach (uint i in input\_block)

textBoxMessageInput.Text += (char)i;

// richTextBox1.Text += i + " ";

}

}

}

private void buttonDecryptFile\_Click\_1(object sender, EventArgs e)

{

if (textBoxPassword.Text.Length < SettingsKeeped.MinKeylength)

{

MessageBox.Show("Password must be " + SettingsKeeped.MinKeylength + " symbols longer");

return;

}

if (PasswordPhrase.isComplexityCorrect(textBoxPassword.Text) != 0)

{

MessageBox.Show("Password doesn't fit stricktness");

return;

}

else

{

uint[] key = new uint[8];

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

{

if (i < key\_length / 32)

key[i] = Convert.ToUInt32(textBoxPassword.Text[i]);

else

key[3] += Convert.ToUInt32(textBoxPassword.Text[i]);

}

byte[] readText;

uint[] input\_block = new uint[4];

try

{

readText = File.ReadAllBytes(textBoxChooseFile.Text);

}

catch

{

MessageBox.Show("Can't open file");

return;

}

FileInfo f = new FileInfo(textBoxChooseFile.Text);

if (SettingsKeeped.Maxfilesize != 0 && (SettingsKeeped.Maxfilesize < f.Length))

{

MessageBox.Show("File is too big");

return;

}

byte[] outText = readText;

if (readText.Length % 16 != 0)

{

MessageBox.Show("Encrypted file is damaged");

return;

}

for (int i = 0; i < readText.Length; i += 16)

{

if (readText.Length - i < 16) //если последний блок размером меньше 16 байт

{

/\* byte[] buffer = new byte[readText.Length - i];

Array.Clear(buffer, 0, buffer.Length);

//buffer.CopyTo(readText, readText.Length);

readText.Concat(buffer); \*/

}

else

{

for (int j = 0; j < 4; j++) //Поправить индексы

{

input\_block[j] = BitConverter.ToUInt32(readText, i + (j \* 4)); //Получили входной блок в удобном формате

}

decrypt\_block(key, key\_length, input\_block);

for (int j = 0; j < 4; j++)

{

BitConverter.GetBytes(input\_block[j]).CopyTo(outText, i + (j \* 4));

}

}

}

if (checkBoxSaveResult.Checked)

{

SaveFileDialog dialog = new SaveFileDialog();

if (dialog.ShowDialog() == DialogResult.OK)

{

if (dialog.FileName != "")

textBoxChooseFile.Text = dialog.FileName;

else

{

MessageBox.Show("Choose output file");

return;

}

File.WriteAllBytes(textBoxChooseFile.Text, outText);

MessageBox.Show("File has been decrypted");

textBoxChooseFile.Text = "";

}

else

{

MessageBox.Show("Choose output file");

return;

}

}

else

{

File.WriteAllBytes(textBoxChooseFile.Text, outText);

MessageBox.Show("File has been decrypted");

textBoxChooseFile.Text = "";

}

}

}

private void buttonEncryptFile\_Click(object sender, EventArgs e)

{

if (textBoxPassword.Text.Length < SettingsKeeped.MinKeylength)

{

MessageBox.Show("Password must be " + SettingsKeeped.MinKeylength + " symbols longer");

return;

}

if (PasswordPhrase.isComplexityCorrect(textBoxPassword.Text) != 0)

{

MessageBox.Show("Password doesn't fit stricktness");

return;

}

if (textBoxPassword.Text != textBoxConfirmPassword.Text)

{

MessageBox.Show("Confirmation is not correct");

return;

}

else

{

uint[] key = new uint[8];

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

{

if (i < key\_length / 32)

key[i] = Convert.ToUInt32(textBoxPassword.Text[i]);

else

key[3] += Convert.ToUInt32(textBoxPassword.Text[i]);

}

byte[] readText;

uint[] input\_block = new uint[4];

try

{

readText = File.ReadAllBytes(textBoxChooseFile.Text);

}

catch

{

MessageBox.Show("Can't open file");

return;

}

FileInfo f = new FileInfo(textBoxChooseFile.Text);

if (SettingsKeeped.Maxfilesize != 0 && (SettingsKeeped.Maxfilesize < f.Length))

{

MessageBox.Show("File is too big");

return;

}

int checksize = 0;

if (readText.Length % 16 != 0)

checksize = 16 - (readText.Length % 16);

byte[] outText = new byte[readText.Length + checksize];

for (int i = 0; i < readText.Length; i += 16)

{

if (readText.Length - i < 16) //если последний блок размером меньше 16 байт

{

//richTextBox1.Text = Convert.ToString(readText.Length);

byte[] buffer = new byte[16];

Array.Clear(buffer, 0, buffer.Length);

for (int j = 0; j < readText.Length - i; j++)

{

buffer[j] = readText[i + j];

}

for (int j = 0; j < 4; j++) //Поправить индексы

{

input\_block[j] = BitConverter.ToUInt32(buffer, j \* 4); //Получили входной блок в удобном формате

}

encrypt\_block(key, key\_length, input\_block);

for (int j = 0; j < 4; j++)

{

BitConverter.GetBytes(input\_block[j]).CopyTo(outText, i + (j \* 4));

}

}

else

{

for (int j = 0; j < 4; j++) //Поправить индексы

{

input\_block[j] = BitConverter.ToUInt32(readText, i + (j \* 4)); //Получили входной блок в удобном формате

// input\_block[j] = BitConverter.ToUInt32(readText, i + (j \* 4)); //Получили входной блок в удобном формате

}

encrypt\_block(key, key\_length, input\_block);

for (int j = 0; j < 4; j++)

{

BitConverter.GetBytes(input\_block[j]).CopyTo(outText, i + (j \* 4));

//richTextBox1.Text = Convert.ToString(outText.Length);

}

}

}

if (checkBoxSaveResult.Checked)

{

SaveFileDialog dialog = new SaveFileDialog();

if (dialog.ShowDialog() == DialogResult.OK)

{

if (dialog.FileName != "")

textBoxChooseFile.Text = dialog.FileName;

else

{

MessageBox.Show("Choose output file");

return;

}

File.WriteAllBytes(textBoxChooseFile.Text, outText);

MessageBox.Show("File has been decrypted");

textBoxChooseFile.Text = "";

}

else

{

MessageBox.Show("Choose output file");

return;

}

}

else

{

File.WriteAllBytes(textBoxChooseFile.Text, outText);

MessageBox.Show("File has been decrypted");

textBoxChooseFile.Text = "";

}

}

}

}

}

FormSettings:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SAFER\_OVERPOWERED

{

public partial class FormSettings : Form

{

public int MaxTextSize;

int KeyComplexity;

public FormSettings()

{

InitializeComponent();

// numericUpDown1.Value = SettingsKeeped.Maxsize;

// numericUpDown3.Value = SettingsKeeped.MinKeylength;

// numericUpDown2.Value = SettingsKeeped.Maxfilesize = 10000;

// numericUpDown2.Minimum = SettingsKeeped.MinKeylength;

KeyComplexity = SettingsKeeped.KeyCompleixity;

if (KeyComplexity == 1)

{

radioButton2.Checked = true;

}

if (KeyComplexity == 2)

{

radioButton3.Checked = true;

}

if (KeyComplexity == 3)

{

radioButton4.Checked = true;

}

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

//SettingsKeeped.Maxsize = Convert.ToInt32(numericUpDown1.Value);

}

private void numericUpDown2\_ValueChanged(object sender, EventArgs e)

{

//SettingsKeeped.Maxfilesize = Convert.ToInt64(numericUpDown2.Value);

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

KeyComplexity = 0;

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

KeyComplexity = 1;

}

private void radioButton3\_CheckedChanged(object sender, EventArgs e)

{

KeyComplexity = 2;

}

private void radioButton4\_CheckedChanged(object sender, EventArgs e)

{

KeyComplexity = 3;

}

private void buttonClose\_Click(object sender, EventArgs e)

{

this.Close();

}

private void buttonOK\_Click(object sender, EventArgs e)

{

SettingsKeeped.Maxsize = Convert.ToInt32(numericUpDown1.Value);

SettingsKeeped.Maxfilesize = Convert.ToInt64(numericUpDown2.Value);

SettingsKeeped.MinKeylength = Convert.ToInt32(numericUpDown3.Value);

// SettingsKeeped.MinKeylength = Convert.ToInt32(numericUpDown3.Minimum);

SettingsKeeped.KeyCompleixity = KeyComplexity;

this.Close();

}

}

}

PasswordPhrase:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SAFER\_OVERPOWERED

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

static class PasswordPhrase

{

public static int isComplexityCorrect(string passWords)

{

if (SettingsKeeped.KeyCompleixity == 0)

return 0;

bool isHaveNumeric = false;

bool isHaveLetter = false;

bool isHaveSpecialCharacters = false;

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

{

if (passWords[i] >= '0' && passWords[i] <= '9')

isHaveNumeric = true;

if ((passWords[i] >= 'a' && passWords[i] <= 'z') || (passWords[i] >= 'а' && passWords[i] <= 'я') || (passWords[i] >= 'A' && passWords[i] <= 'Z') || (passWords[i] >= 'А' && passWords[i] <= 'Я'))

isHaveLetter = true;

if ((passWords[i] >= '!' && passWords[i] <= '/') || (passWords[i] >= ':' && passWords[i] <= '?'))

isHaveSpecialCharacters = true;

}

if (SettingsKeeped.KeyCompleixity == 1)

{

if (isHaveNumeric)

return 0;

return 1;

}

if (SettingsKeeped.KeyCompleixity == 2)

{

if (isHaveNumeric && isHaveLetter)

return 0;

return 1;

}

if (SettingsKeeped.KeyCompleixity == 3)

{

if (isHaveNumeric && isHaveLetter && isHaveSpecialCharacters)

return 0;

return 1;

}

return 2;

}

private static int isLengthCorrect(string passWords)

{

return 1;

}

}

}