МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ 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: «Визуализация работы алгоритма шифрования RC2»

Исполнитель

Студент 3 курса группы 2 Говоронок В.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 2021

**Содержание**

[Введение 3](#_Toc71745186)

[1. Постановка задачи 4](#_Toc71745187)

[2. Аналитический обзор литературы 5](#_Toc71745188)

[3. Безопасность и криптостойкость алгоритма 12](#_Toc71745189)

[4. Описание программного средства 13](#_Toc71745190)

[5. Тестирование программного средства 17](#_Toc71745191)

[6. Руководство пользователя 22](#_Toc71745192)

[Заключение 28](#_Toc71745193)

[Список используемых источников 29](#_Toc71745194)

[ПРИЛОЖЕНИЕ А 30](#_Toc71745195)

# Введение

Алгоритм шифрования RC2 был разработан в конце 1980-х гг. Рональдом Ривестом, который, в частности, разработал алгоритм RC5.

Является собственностью компании RSA Data Security; при этом известен тот факт, что разработку данного алгоритма инициировала и частично спонсировала фирма Lotus, которой требовался сильный (но не являющийся широко распространенным) алгоритм шифрования для последующего использования в программной системе Lotus Notes. Причем криптостойкость алгоритма должна была быть проверена Агентством Национальной Безопасности (АНБ) США. АНБ также внесло свой вклад в разработку алгоритма, предложив некоторые детали реализации, внедренные в алгоритм Ривестом [5].

Еще одна интересная особенность состоит в том, что RC2 был разрешен к экспорту за пределы США. Есть источники, в которых утверждается, что взамен на такое благоволение со стороны контролирующих органов Ривест внес в процедуру расширения ключа некие ослабляющие алгоритм действия.

Не менее интересен и тот факт, что долгое время после разработки RC2 структура данного алгоритма оставалась в секрете, но в начале 1996 г. анонимный автор опубликовал в Usenet-конференции sci.crypt исходные тексты алгоритма. Причем не ясно, была ли это утечка информации из RSA Data Security или результат реверс-инжиниринга какой-либо программной реализации алгоритма RC2.

Не известно, насколько долго алгоритм оставался бы в секрете, не будь этой анонимной публикации, но в марте 1998 г. Ривест полностью опубликовал алгоритм в виде RFC 2268.

Алгоритм получил широкое распространение, как минимум, благодаря тому факту, что он является основным и обязательным для реализации алгоритмом шифрования согласно стандарту защиты сообщений электронной почты S/MIME.

Алгоритм RC2 шифрует данные блоками по 64 бита с использованием ключей переменного размера: от 8 до 1024 битов включительно; рекомендуемым размером ключа является 64 бита.

Алгоритм является сетью Фейстеля, в нем выполняются 18 раундов преобразований. Причем раунды алгоритма делятся на два типа: смешивающие(mix) раунды и объединяющие (mesh) раунды.

Цель работы – разработка приложения визуализации работы блочного алгоритма шифрования RC2.

В процессе выполнения работы были поставлены следующие задачи:

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

# Постановка задачи

Требуется разработать приложение визуализации работы блочного алгоритма шифрования RC2 на примере шифрования данных и их расшифрования. Данными могут служить пароль, закрытый ключ и т.д.

Требование к средствам разработки:

1. Язык программирования C#.
2. Средство разработки MS Visual Studio.

Программа должна:

* шифровать данные при помощи алгоритма RC2;
* расшифровывать данные;
* отображать ввод закрытого ключа. Закрытый ключ – это строка определенной длины. Для метода RC2 длина строки закрытого ключа должна составлять не менее 5 символов;
* отображать ввод строки для шифрования;
* отображать результаты шифрования;
* отображать результаты расшифрования;
* отображать результат каждого раунда шифрования.

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

# 2. Аналитический обзор литературы

Алгоритм RC2 шифрует данные блоками по 64 бита с использованием ключей переменного размера: от 8 до 1024 битов включительно; рекомендуемым размером ключа является 64 бита.

В [США](https://ru.wikipedia.org/wiki/%D0%A1%D0%A8%D0%90) длина ключа для использования внутри страны рекомендуется равной 128 битам, но соглашение, заключённое между Software Publishers Association (SPA) и правительством США, даёт RC2 специальный статус, который означает, что разрешено экспортировать шифры длиной ключа до 40 битов. 56-битные ключи разрешено использовать заграничным отделениям американских компаний.

RC2 с ключами 128 битов обеспечивает такой же уровень безопасности, как и [IDEA](https://ru.wikipedia.org/wiki/IDEA) или [тройной DES](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B9%D0%BD%D0%BE%D0%B9_DES). RC2 широко используется разработчиками, чьи продукты экспортируются за пределы США, поскольку экспортировать DES долгое время было запрещено. RC2 используется, в частности, в неамериканских версиях программы [Outlook Express](https://ru.wikipedia.org/wiki/Outlook_Express) корпорации [Microsoft](https://ru.wikipedia.org/wiki/Microsoft).

Алгоритм является сетью Фейстеля, в нем выполняются 18 раундов преобразований. Причем раунды алгоритма делятся на 2 типа: смешивающие (mix) раунды и объединяющие (mesh) раунды.

Опишем алгоритм RC2. Общая структура алгоритма такова:

1. Выполняются 5 смешивающих раундов.
2. Выполняется 1 объединяющий раунд.
3. Выполняются 6 смешивающих раундов.
4. Выполняется 1 объединяющий раунд.
5. Выполняются 5 смешивающих раундов.

Предполагается, что шифруемый блок данных разделен на 4 16-битных слова R0..R3, над которыми смешивающий раунд (рисунок 2.1, таблица 2.1) в цикле по i от 0 до 3 выполняет следующие операции, как показано в формулах 2.1, 2.2 и 2.3.

(2.1)

(2.2)

(2.3)

где Kj – фрагмент расширенного ключа, определяемый глобальной переменной;

j – данная переменная в начале шифрования является нулевой и увеличивается на 1 (как показано выше) в каждом смешивающем раунде, процедура расширения ключа подробно описана далее;

& – побитовая логическая операция «и»;

~x  – побитовый комплемент к x;

<<<  – циклический сдвиг влево на число битов, определяемое значением Si.

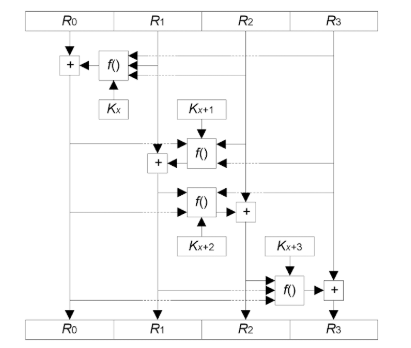


Рисунок 2.1 – Смешивающий раунд алгоритма RC2 [5]

Таблица 2.1 – Смешивающий раунд алгоритма RC2 [5]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I | 0 | 1 | 2 | 3 |
| Si | 1 | 2 | 3 | 5 |

Таким образом, в каждой i-й итерации смешивающего раунда выполняется описанное выше преобразование, которое модифицирует Ri  на основе текущих значений трех остальных слов шифруемого блока и фрагмента расширенного ключа. Функция предоставлена на рисунке 2.2.



Рисунок 2.2 – Функция алгоритма RC2 [5]

Аналогично смешивающему раунду, в объединяющем раунде выполняется цикл по i от 0 до 3; в каждой итерации цикла выполняется следующая операция [5]:

, (2.4)

где

(2.5)

Таким образом, итерация объединяющего раунда представляет собой наложение операцией сложения по модулю 216 фрагмента расширенного ключа, индекс которого определяется 6 младшими битами текущего значения слова Ri-1mod4.

Как было сказано выше, алгоритм RC2 использует ключи шифрования размером от 8 до 1024 битов, т. е. от 1 до 128 байтов.

Расширение ключа подразумевает получение из ключа шифрования 16-битных фрагментов расширенного ключа K0...K63, используемых в смешивающих раундах – по одному в каждой из 4-х итераций каждого из 16 смешивающих раундов. Данная процедура выполняется в несколько шагов:

1. Инициализируется байтовый массив L0...L127 по формулам 2.6 и 2.7, используемый при расширении ключа:

, (2.6)

, (2.7)

Где K0…KT-1 – исходный ключ шифрования, имеющий размер T байтов.

Остальные байты массива L обнуляются.

1. Инициализируются другие переменные, участвующие в расширении ключа:

, (2.8)

, (2.9)

где T1 – размер ключа в битах;

T8 – эффективный размер ключа в байтах;

TM – битовая маска, учитывающая остаточные биты ключа, если его размер в битах не кратен 8.

1. В цикле по i от T до 127 выполняется следующая операция [5]:

(2.10)

где P – табличная замена, приведенная в таблице 2.2.

Таблица 2.2 – Табличная замена [5]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D9 | 79 | F9 | C4 | 19 | dd | B5 | ed | 28 | E9 | fd | 79 | 4a | A0 | D8 | 9d |
| C6 | 7e | 37 | 83 | 2b | 76 | 53 | 8e | 62 | 4c | 64 | 88 | 44 | 8b | fb | A2 |
| 17 | 9a | 59 | F5 | 87 | B3 | 4f | 13 | 61 | 45 | 6d | 8d | 09 | 81 | 7d | 32 |
| bd | 8f | 40 | eb | 86 | B7 | 7b | 0b | F0 | 95 | 21 | 22 | 5c | 6b | 4e | 82 |

Продолжение таблицы 2.2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 54 | D6 | 65 | 93 | ce | 60 | B2 | 1c | 73 | 56 | C0 | 14 | A7 | 8c | F1 | dc |
| 12 | 75 | ca | 1f | 3b | be | E4 | D1 | 42 | 3d | D4 | 30 | A3 | 3c | B6 | 26 |
| 6f | B6 | 0e | da | 46 | 69 | 07 | 57 | 27 | F2 | 1d | 9b | bc | 94 | 43 | 03 |
| F8 | 11 | C7 | F6 | 90 | ef | 3e | E7 | 06 | C3 | D5 | 2f | C8 | 66 | 1e | D7 |
| 08 | E8 | ea | de | 80 | 52 | ee | F7 | 84 | aa | 72 | ac | 35 | 4d | 6a | 2a |
| 96 | 1a | D2 | 71 | 5a | 15 | 49 | 74 | 4b | 9f | D0 | 5e | 04 | 18 | A4 | ec |

Таким образом, значение 0 заменяется на D9, 1 – на 78 и т. д.

Псевдослучайная таблица P сформирована на основе шестнадцатеричной записи дробной части числа π.

1. Вычисляется L128-T8:

(2.11)

1. В цикле по i выполняется действие по формуле 2.5:

(2.12)

1. Для использования в шифрующих преобразованиях 128-байтная последовательность представляется в виде 16-битных слов K0..K63:

(2.13)

Расшифровывание выполняется по той же общей схеме, что и зашифровывание. Однако при расшифровывании используются другие операции, выполняемые в смешивающем и объединяющем раундах.

Смешивающий раунд расшифровывания в цикле по i от 3 до 0 выполняет следующие операции по формулам 2.14, 2.15, 2.16:

(2.14)

(2.15)

(2.16)

где >>>  – циклический сдвиг вправо на число битов, определяемое значением Si.

А начальное значение j устанавливается не в 0, а в 63.

Аналогичным образом изменен и объединяющий раунд при расшифровывании по сравнению с зашифровыванием. В нем в цикле по i от 3 до 0 выполняется следующая операция по формулам 2.17 и 2.18:

, (2.17)

где

(2.18)

Далее приведем аналоги программ для шифрования, в которых присутствует алгоритм шифрования RC2.

VeraCrypt – один из самых популярных инструментов безопасности, предоставляющий шифрование корпоративного уровня для важных данных. Интерфейс предоставлен на рисунке 2.3.

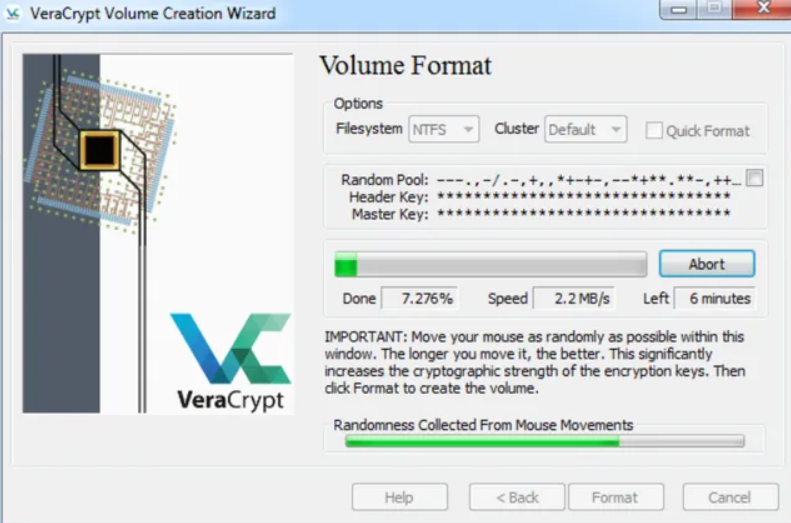


Рисунок 2.3 – Интерфейс программы VeraCrypt

Система довольно проста в использовании, и всё, что она действительно делает, это добавляет зашифрованные пароли к данным и разделам. Всё, что нужно сделать, это дать инструменту несколько деталей о данных, таких как размер тома, местоположение и алгоритм хеширования.

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

Плюсы VeraCrypt:

* базовая версия полностью бесплатна;
* обеспечивает эффективное шифрование.

Минусы VeraCrypt:

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

AxCrypt – шифрование для небольших групп и отдельных лиц. Интерфейс предоставлен на рисунке 2.4.

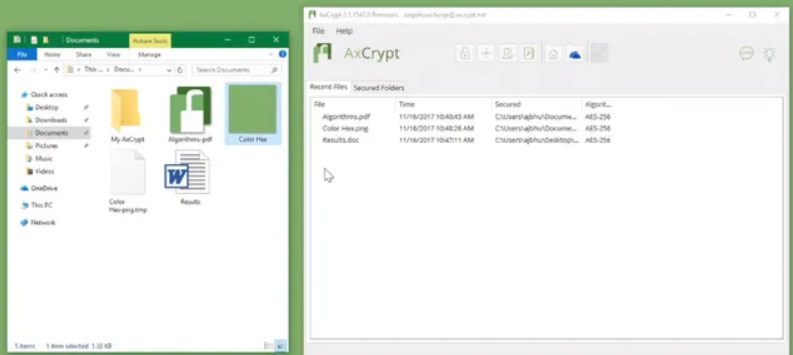


Рисунок 2.4 – Интерфейс программы AxCrypt

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

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

AxCrypt многоязычен и может работать с такими языками, как нидерландский, французский, немецкий, итальянский, корейский, испанский, шведский, русский и португальский. Кроме того, есть управление паспортами, и можно получить доступ к зашифрованным файлам через приложение для смартфона.

Плюсы AxCrypt:

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

Минусы AxCrypt:

* Ориентирована на мобильные устройства.

Folder Lock – эффективное шифрование для физических лиц. Интерфейс предоставлен на рисунке 2.5.

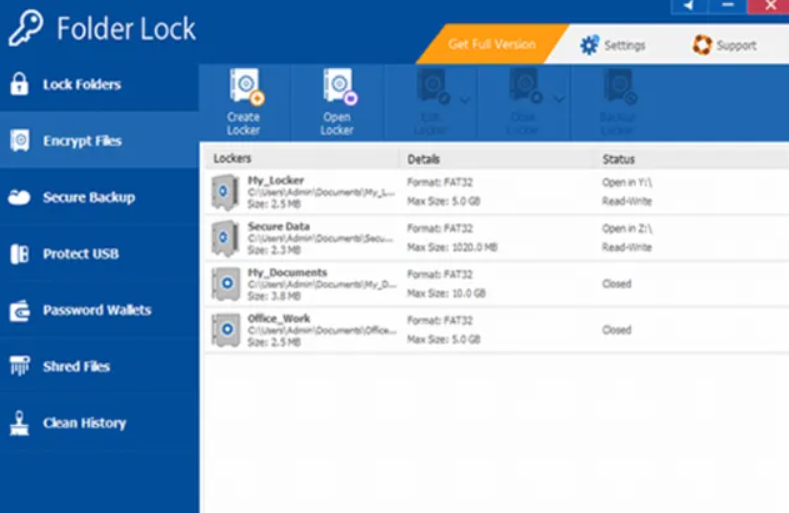


Рисунок 2.5 – Интерфейс программы Folder Lock

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

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

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

Плюсы Folder Lock:

* Бесплатная базовая версия;
* Эффективное личное шифрование.

Минусы Folder Lock:

* Ориентирован на мобильные устройства.

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

# Безопасность и криптостойкость алгоритма

Почти сразу после публикации RC2 вышла работа ряда известных криптологов: Рональда Ривеста, Ларса Кнудсена, Винсента Риджмена и Мэта Робшоу, – в которой исследовалось воздействие дифференциального и линейного криптоанализа на алгоритм RC2. Результаты оказались таковы:

* алгоритм не подвержен атаке методом линейного криптоанализа;
* алгоритм может быть теоретически вскрыт методом дифференциального криптоанализа, для чего необходимо наличие не менее 259 пар выбранных открытых текстов и соответствующих им шифртекстов – такая атака вряд ли реализуема на практике.

Не более практичной является атака на связанных ключах, для успешного проведения которой требуется наличие 234 выбранных открытых текстов и соответствующих им шифртектов, причем зашифровывание должно выполняться на ключе, связанном с искомым определенным простым соотношением. Данная атака изобретена не менее известными криптологами: Джоном Келси, Брюсом Шнайером и Дэвидом Вагнером.

Других методов вскрытия алгоритма RC2 на настоящий момент не известно.

# Описание программного средства

Для разработки программного средства, проведем анализ существующих языков программирования и выберем наиболее оптимальный для данного решения.

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

1. Техническое сообщество не так давно отпраздновало 25-летний юбилей Java. Это один из наиболее широко принятых языков программирования, используемый около 9 миллионами разработчиков, и работает на 7 млрд устройств по всему миру. Это язык программирования, используемый для разработки всех родных приложений Android. Популярность Java-разработчиков исходит из того, что этот язык имеет долгосрочную совместимость, которая гарантирует, что старые приложения продолжат работать и сейчас и в будущем. Единственная сложность заключается в том, что этот язык достаточно сложен в освоении особенно для новичков.
2. JavaScript. Серверные языки сценариев идеально подходят для разработки сложных веб приложений, но каждая такая задача сильно нагружает сервер. Поэтому разработчики делегировали часть функций на сторону клиента и использовали JavaScript. JavaScript – это язык программирования, выполняемый в клиентском браузере и обрабатывает команды на компьютере конечного пользователя, а не сервера, что приводит к снижению нагрузки на сервер и увеличению скорости работы приложения. JavaScript был разработан компанией Netscape и вряд ли есть сайты, которые не используют его.
3. С# принадлежит семье языков программирования Microsoft и был разработан в 2000 году и стал частью первого релиза .NET framework. Язык С# сочетает в себе надежность С++ с дополнительными возможностями Java. Поэтому если программист хорошо знает Java, можно легко переключиться на С# и наоборот. Язык С# позволяет разрабатывать практически любые приложения, которые связаны с Visual Studio IDE [2].
4. Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java и Objective-C. C ценят за его эффективность; он является самым популярным языком для создания системного программного обеспечения. Изучение этого языка ведет к пониманию и других языков. Язык С используется для разработки низкоуровневых приложений, так как считается ближе всего к аппаратному, уступая только ассемблеру.
5. Python – высокоуровневый язык программирования, который часто считается самым легким языком благодаря своей простоте, читаемости и синтаксису. 8 из 10 факультетов информатики в США и 27 из 39 лучших университетов используют Python для обучения студентов программированию.
6. PHP – это один из наиболее широко используемых языков для разработки динамических веб сайтов. PHP открытый язык разработки, поэтому написаны уже тысячи модулей, которые можно модифицировать до требуемой функциональности. На PHP разработано большинство сайтов, ориентированных на большой объём данных.

В результате исследования приложение будет написано на языке программирования С#, так как данный язык является на сегодняшний момент. Используемая среда MS Visual Studio имеет бесплатную версию, т.е. для разработчика не потребуется никаких материальных затрат, связанных с написанием программы.

Для реализации программы шифрования RC2 были разработаны такие методы, как:

* CheckEncr;
* MixRound;
* MeshRound;
* Key;
* TextToBinaryText;
* BinaryTextToText;
* EncryptInMemory;
* DecryptInMemory;
* EncryptToFile;
* DecryptFromFile;
* CyclicShiftleft;
* ClearAllFieldsButton;
* CiclycShiftRight.

Все методы можно посмотреть на рисунке 4.1. Как видно, каждый метод принимает входные параметры, т.к. некоторые отвечают за кнопки, а некоторые выполняют сложные операции.

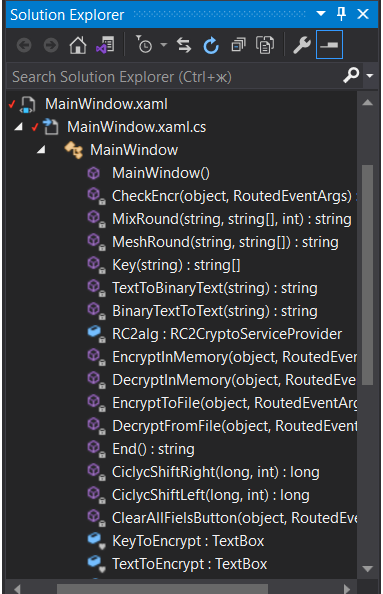


Рисунок 4.1 – Solution explorer

Компонент MainWindow.xaml содержит в себе всю разметку приложения, а также логику, для него.

MainWindow – это конструктор окна.

CheckEncr – метод, который вызывается при нажатии кнопки шифрования на страничке подробного шифрования.

MixRound – метод, который выполняет действия, соответствующие раунду смешивания.

MeshRound – метод, который выполняет действия для объединяющего раунда алгоритма.

Key – метод, служащий для преобразования ключа.

TextToBinaryText – метод для преобразования обычного текста в бинарный вид.

BinaryTextToText – метод для преобразования текста в бинарном виде в обычный текст.

EncryptInMemory – метод, который выполняется после нажатия кнопки «Encrypt» на главной странице, для выполнения шифрования в приложении.

DecryptInMemory – метод, который выполняется после нажатия кнопки «Decrypt» на главной странице, для выполнения расшифровки в приложении.

End – для завершения алгоритма и вывода конечного результата.

CyclicShiftRight – метод для циклического сдвига вправо.

CyclicShiftLeft – метод для циклического сдвига влево.

ClearAllFieldsButton – метод для очистки всех полей.

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

# Тестирование программного средства

Протестируем разработанное программное средство. Для начала проверим программу на запуск. Для этого запустим исполняемый файл программы. После запуска появляется окно, как показано на рисунке 5.1.

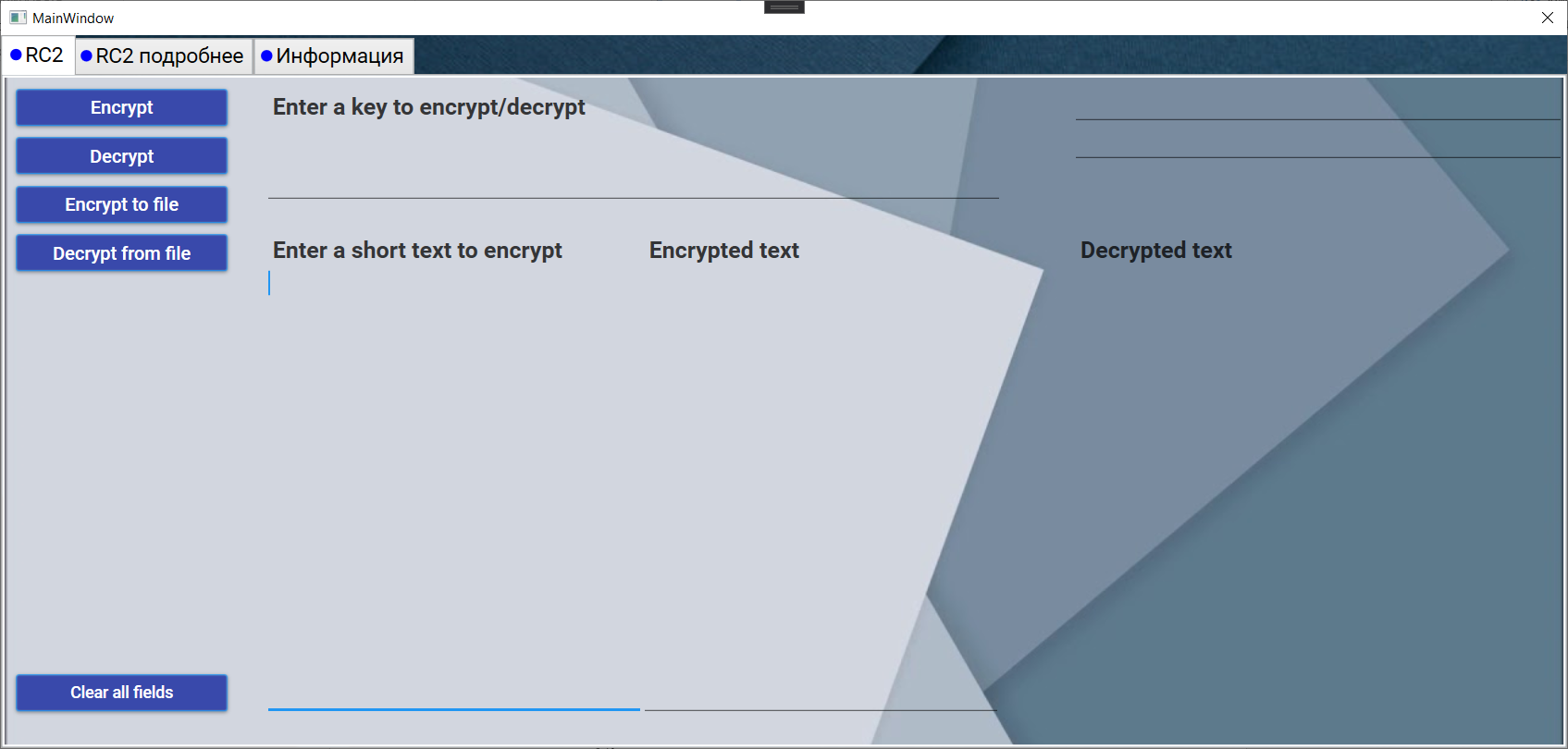


Рисунок 5.1 – Успешный запуск программы

Как видно из рисунка 5.1 программа успешно запустилась, далее проверим на контрольном примере шифрование строки. Для этого произвольно введем ключ, как показано на рисунке 5.2.

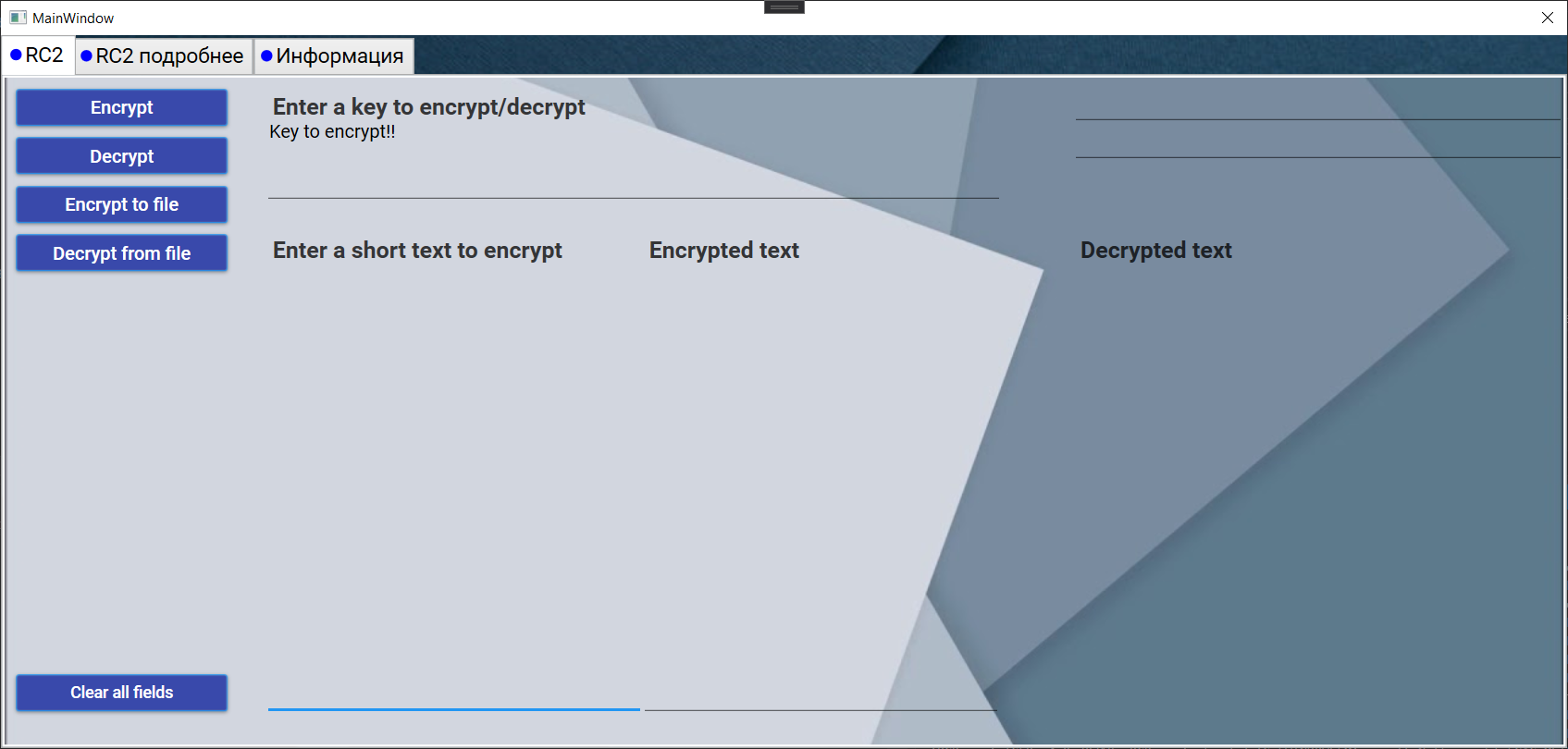


Рисунок 5.2 – Ввод ключа

Далее введем строку, которую будем зашифровывать как показано на рисунке 4.3.

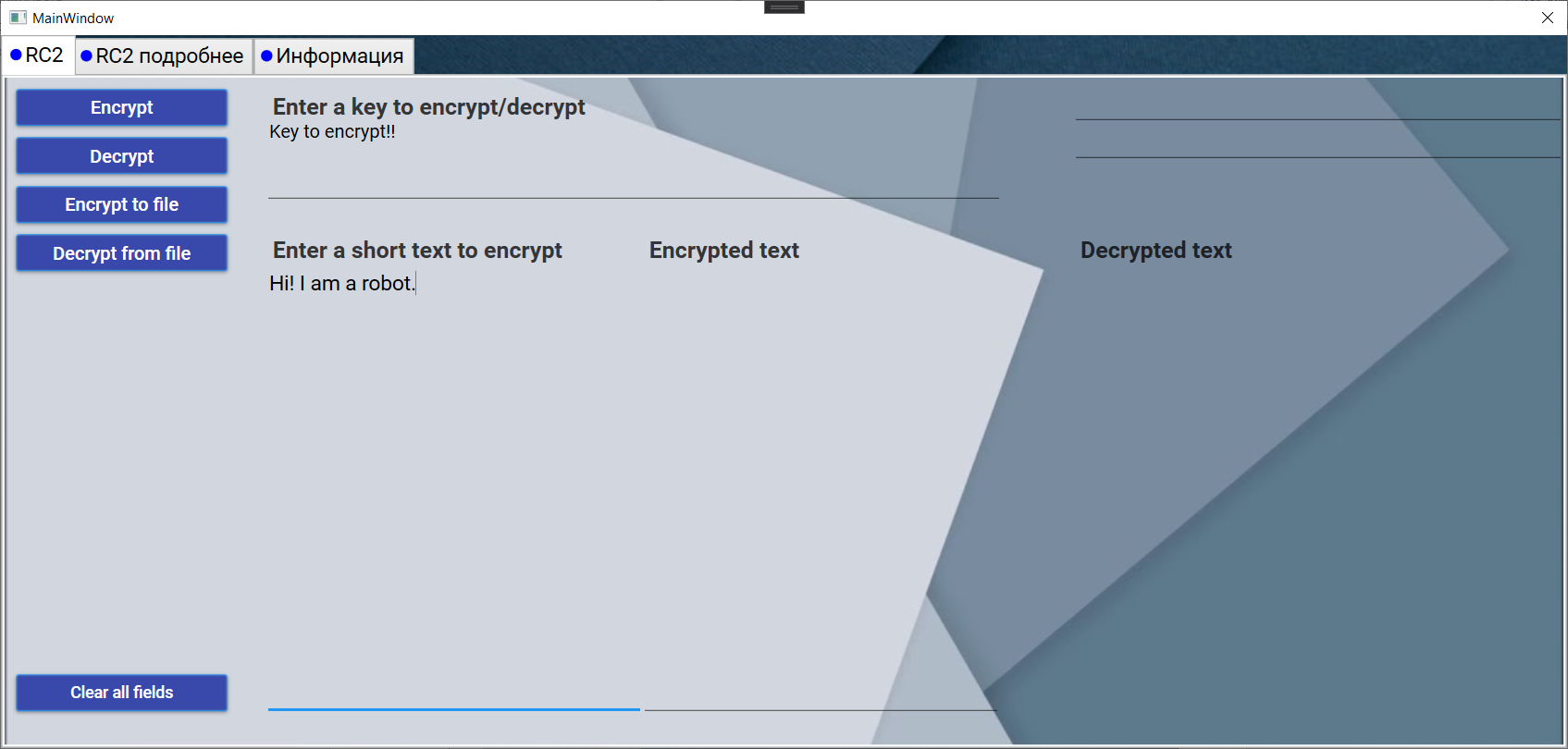


Рисунок 5.3 – Ввод строки

Дальше при нажатии на кнопку «Encrypt» программа должна успешно зашифровать строку, как показано на рисунке 5.4.

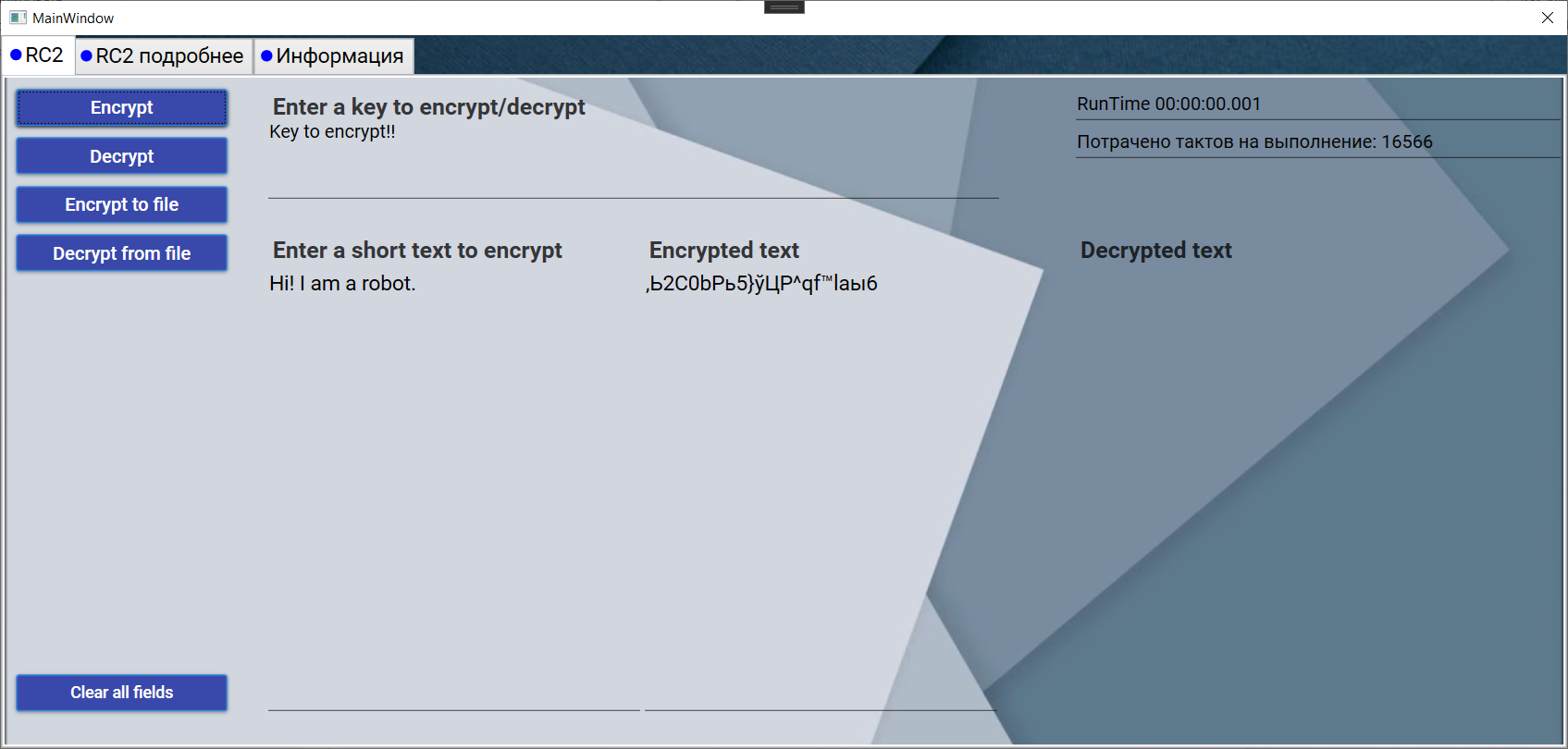


Рисунок 5.4 – Успешное шифрование строки

Как видно из рисунка 5.4 шифрование прошло успешно.

Далее протестируем программу на расшифрование зашифрованной строки. Для этого нужно нажать на кнопку «Decrypt». Полученный результат видно на рисунке 5.5.

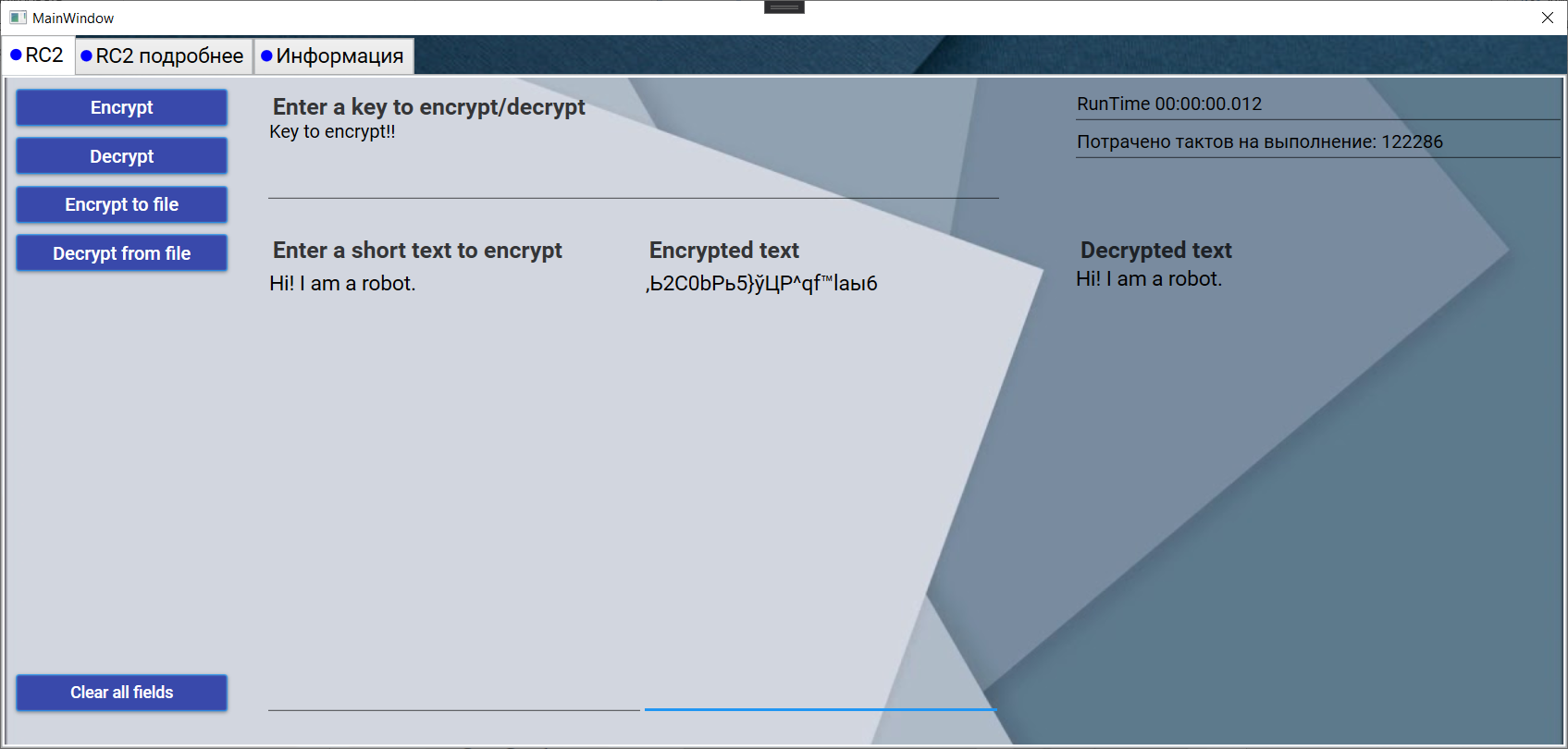


Рисунок 5.5 – Успешное расшифрование зашифрованной строки

Как видно из рисунка 5.5 расшифрование строки прошло успешно, и мы получили первоначальную строку.

Проверим программу на выполнение шифрования и записи зашифрованного текста в файл. Вводим текст в программу и нажимаем на кнопку «Encrypt to file». Как показано на рисунке 5.6.



Рисунок 5.6 – Ввод текста для записи в файл

Откроем файл, в который зашифровали текст через программу «HxD» и посмотрим данные. Результат на рисунке 5.7.

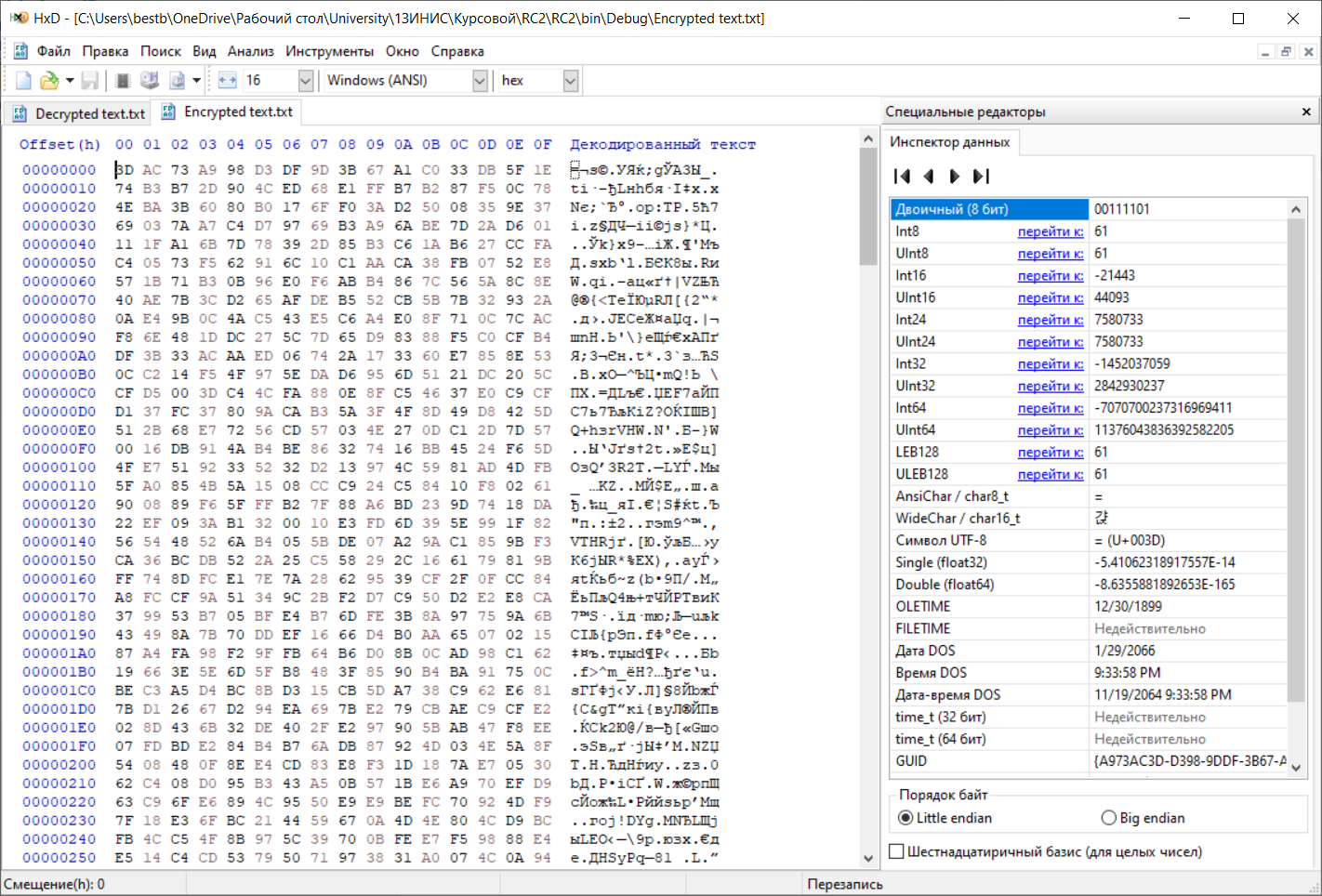


Рисунок 5.7 – Зашифрованный текст в файле

Нажмем на кнопку «Decrypt from file», и данные из файла будут расшифрованы в другой файл. После открытия в той же программе видим результат расшифровки на картинке 5.8.

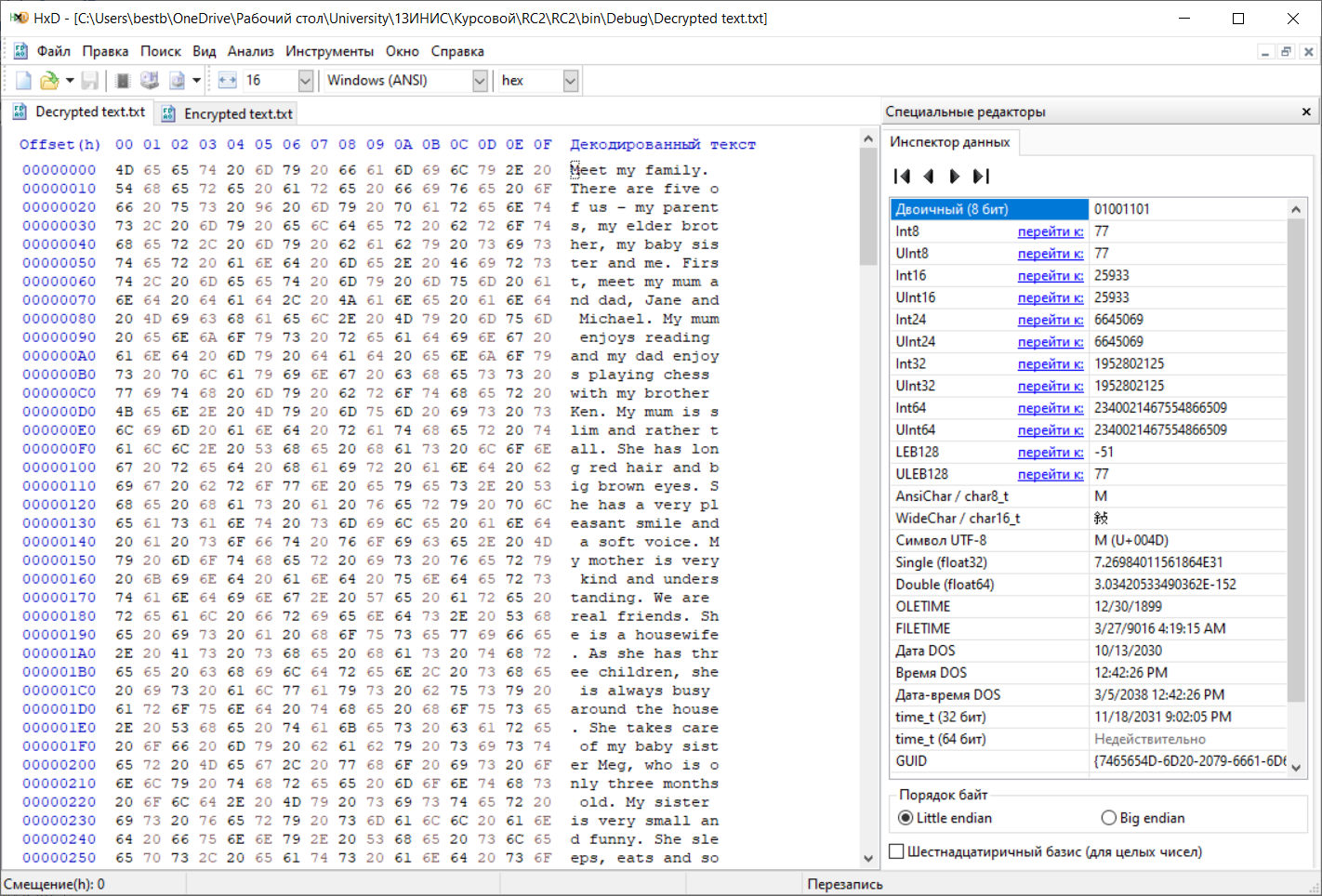


Рисунок 5.8 – Расшифрованный текст в файле

Проверим программу на ошибочный ввод. Оставим ключ пустым, как показано на рисунке 5.9.

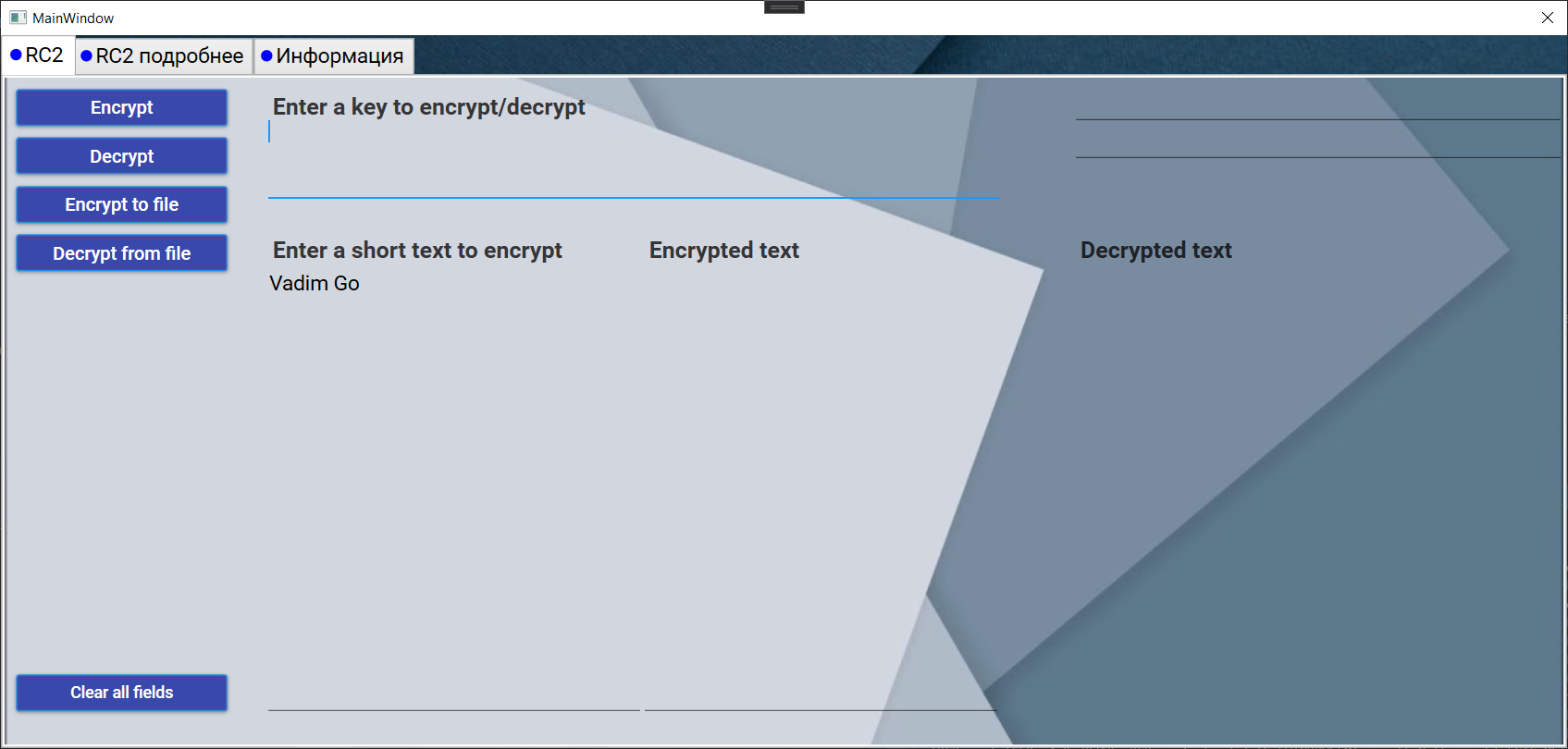


Рисунок 5.9 – Ввод новых данных ключа для проверки ошибочного ввода

Далее нажмем на кнопку «Encrypt», и видим ошибку на рисунке 5.10.

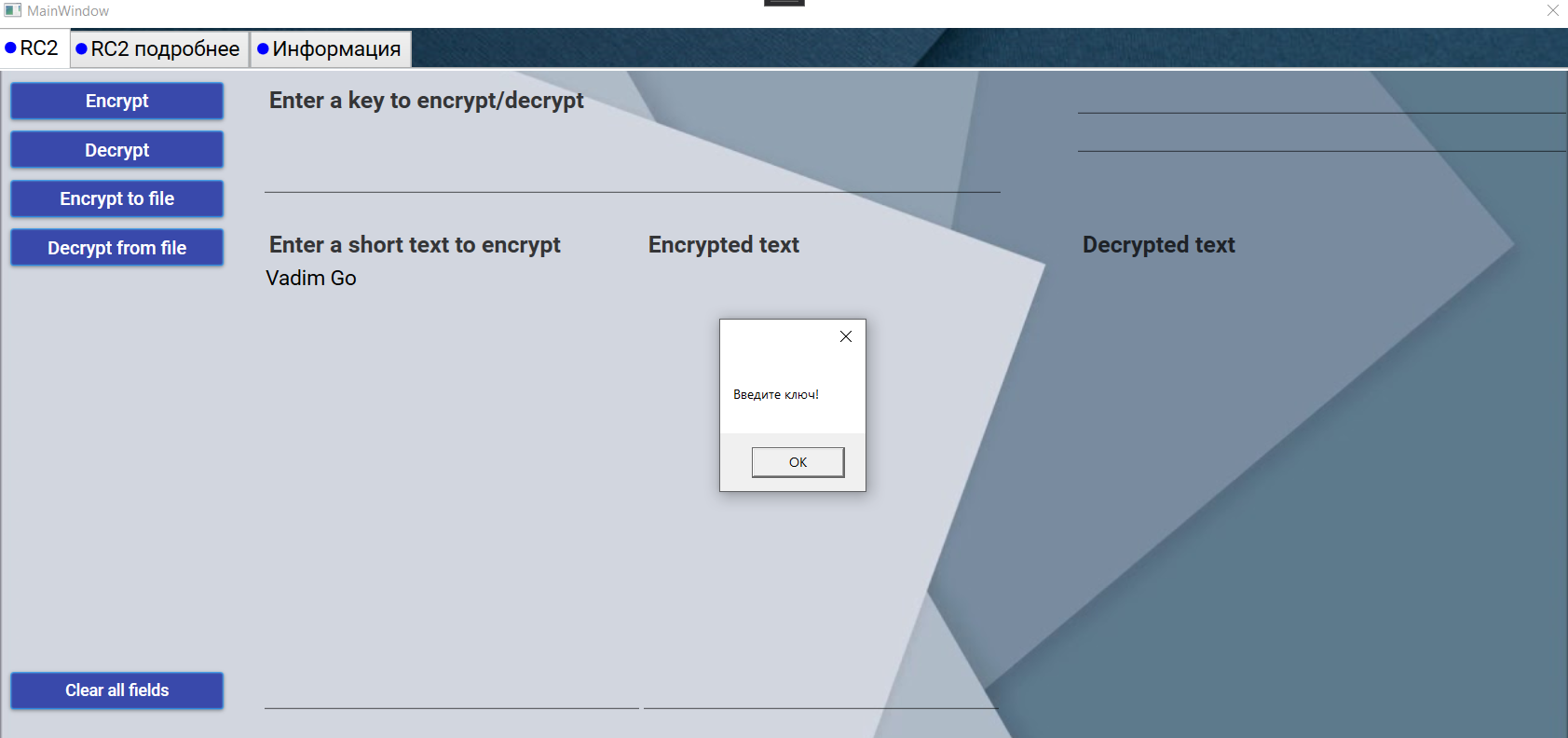


Рисунок 5.10 – Сообщение об ошибке ввода ключа

Как видно из рисунка 5.10, программа вывела сообщение об ошибке ввода, так как пустое поле для ввода ключа нельзя использовать.

Таким образом, при проведенном тестировании можно говорить о том, что разработанная программа успешно шифрует и расшифровывает строки при помощи алгоритма RC2, а также успешно предотвращает ошибочный и некорректный ввод данных, оповещая соответствующим сообщением.

# Руководство пользователя

Для работы с программой ее можно переместить в любое удобное место на жестком диске. После чего запустить программу. Откроется главная страница программы как показано на рисунке 6.1.

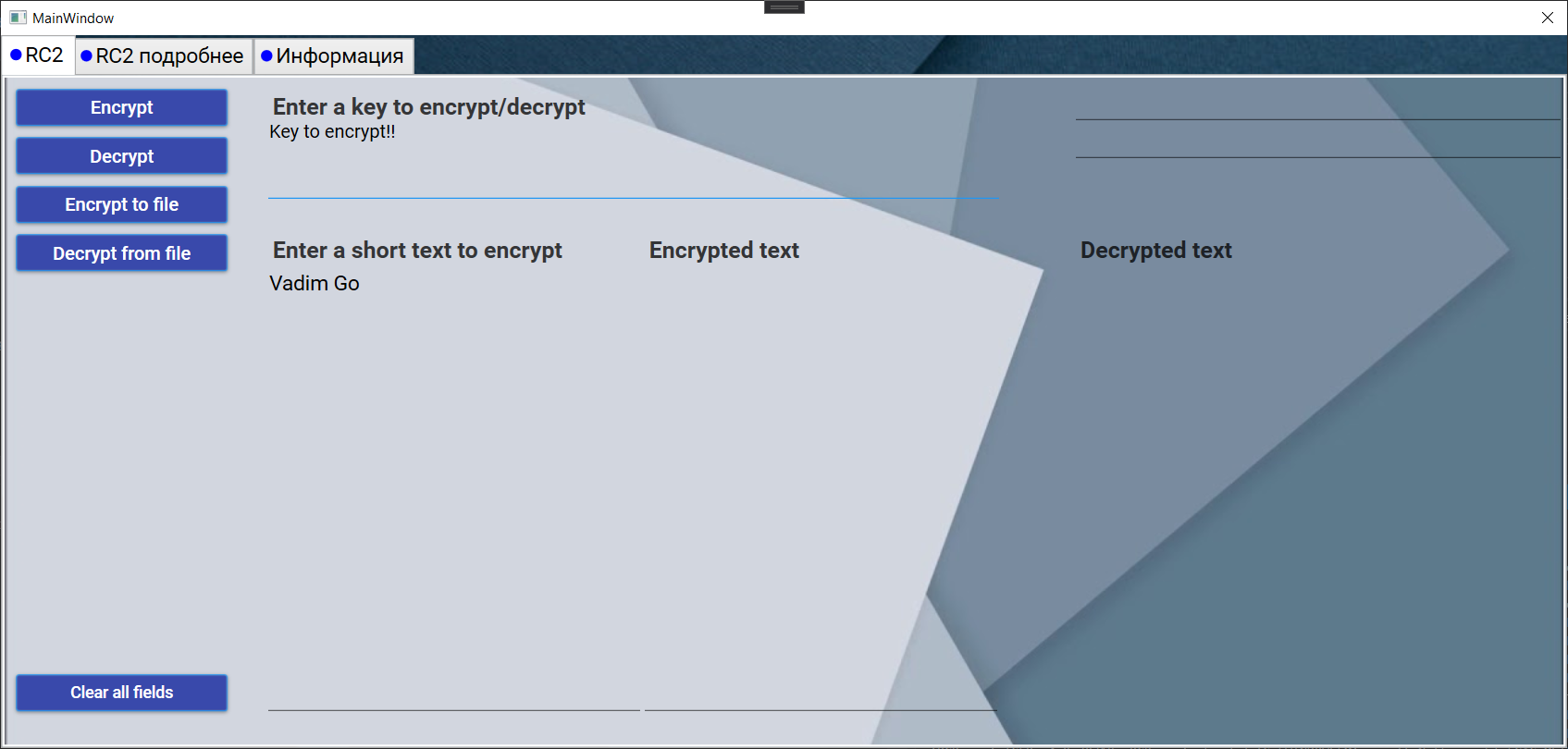


Рисунок 6.1 – Главная страница программы

Как видно из рисунка 6.1 поле для ключа и поле данных для зашифрования заполнены по умолчанию. Для начала шифрования требуется ввести значение ключа, в поле для ключа, как указано на рисунке 6.2. Значение ключа не должно быть пустым иначе программа выдаст сообщение об ошибке, как это было ранее на рисунке 5.10.

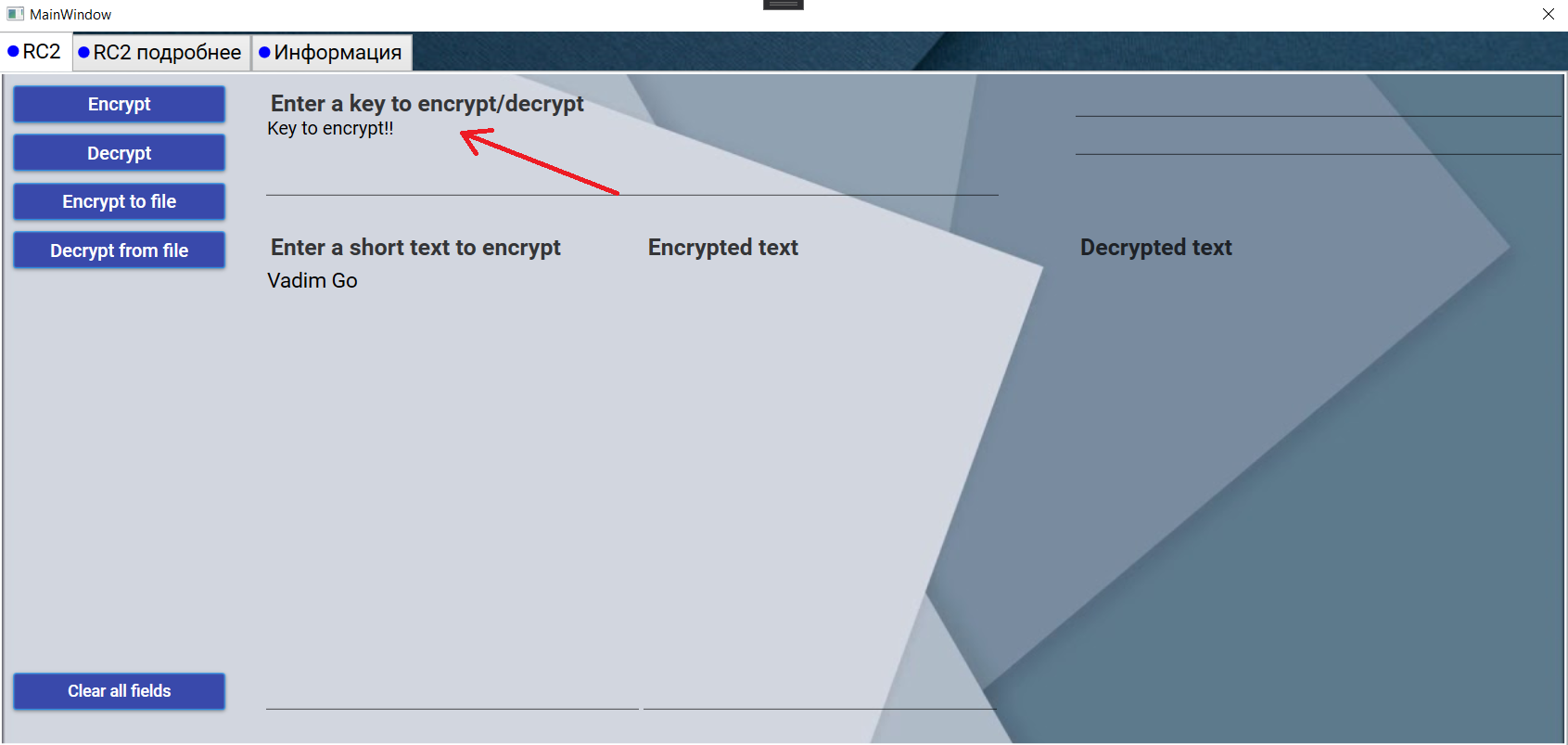


Рисунок 6.2 – Ввод ключа

Далее требуется ввести строку для шифрования, строка не должна быть пустой. Поле для ввода текста на показано рисунке 6.3.

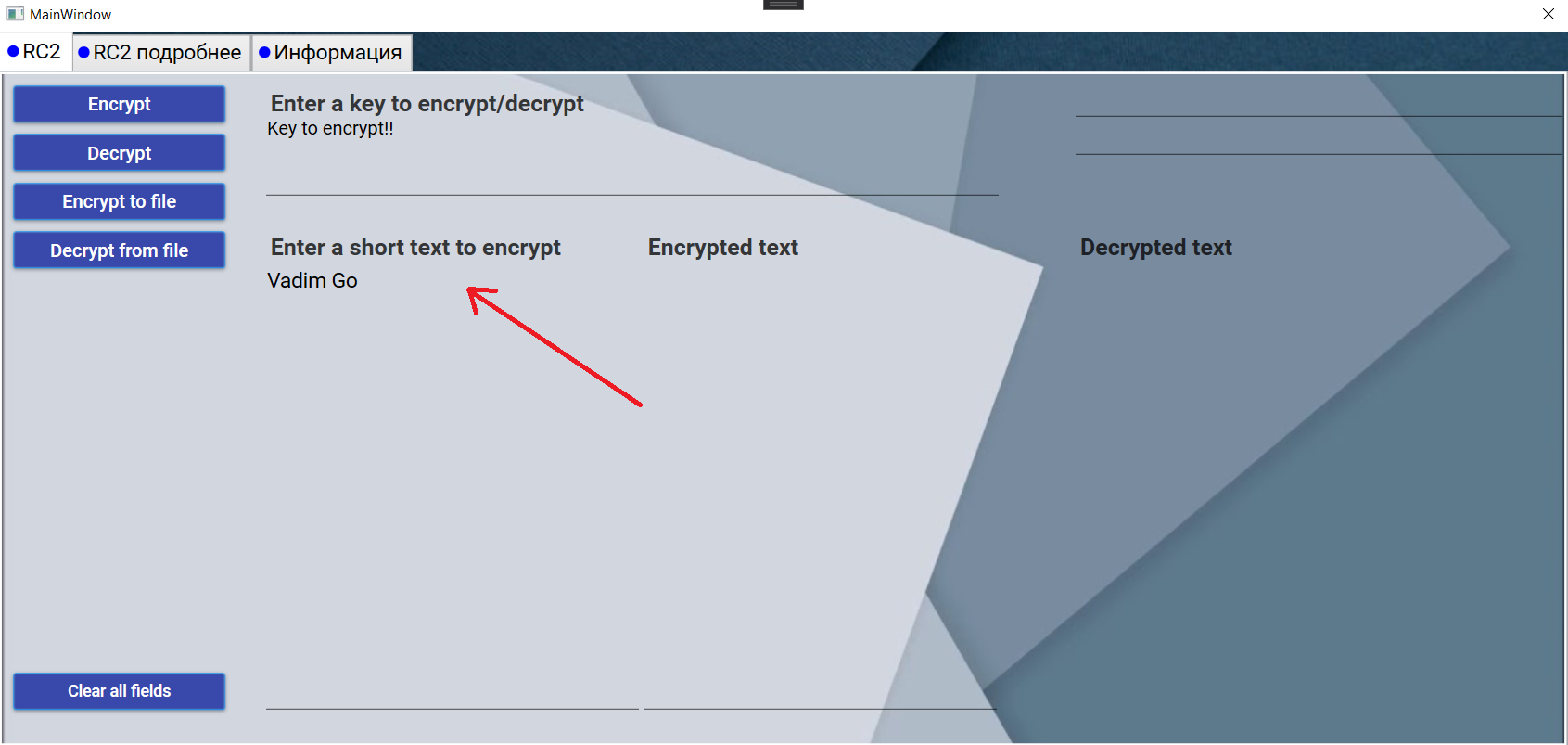


Рисунок 6.3 – Ввод строки для шифрования

После ввода строки, для того чтобы ее зашифровать, нужно нажать на кнопку «Encrypt». Кнопка показана на рисунке 6.4.

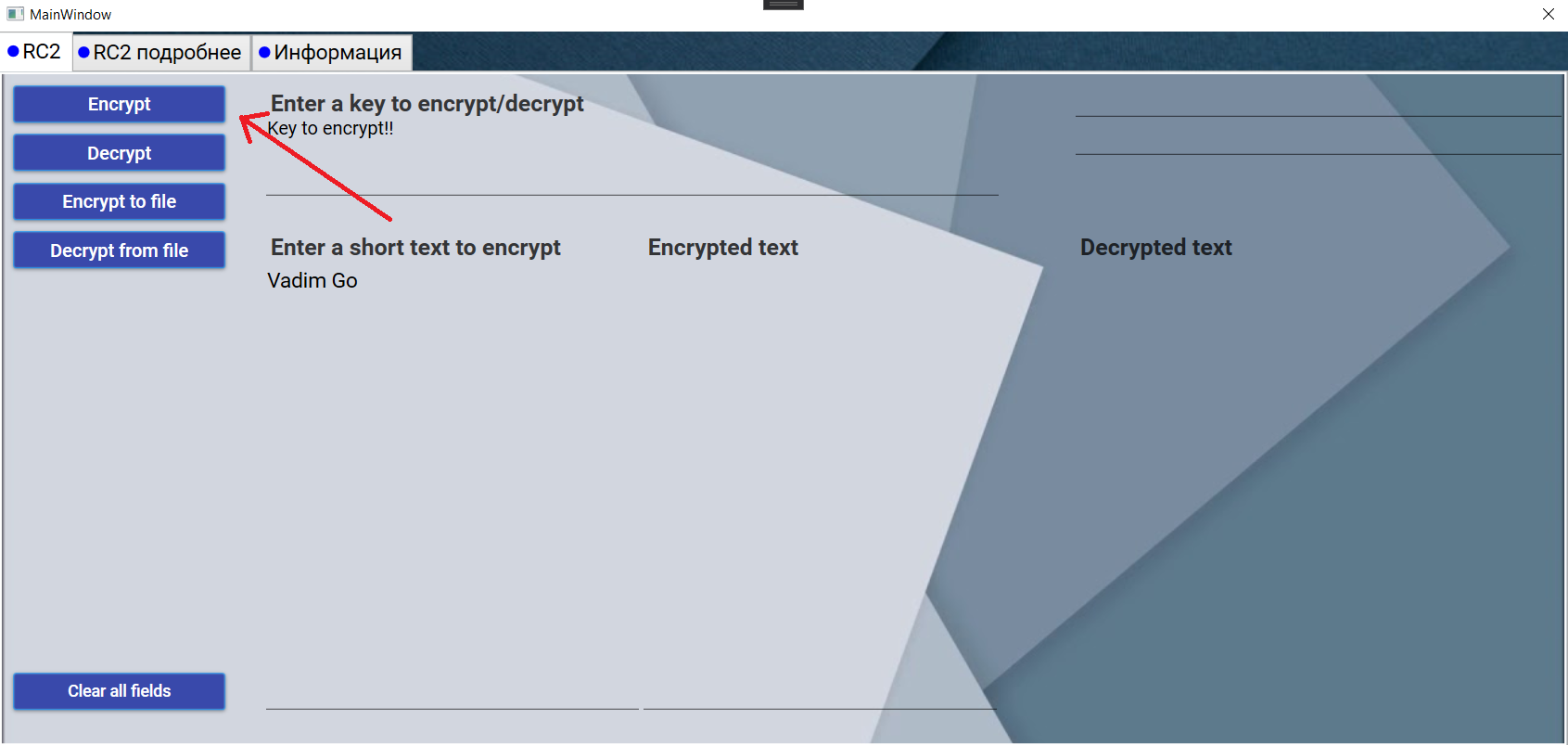


Рисунок 6.4 – Нажатие на кнопку «Encrypt»

После того как нажали на кнопку «Encrypt» в «Encrypted text» будет показано результирующие значения шифрования, как на рисунке 6.5.

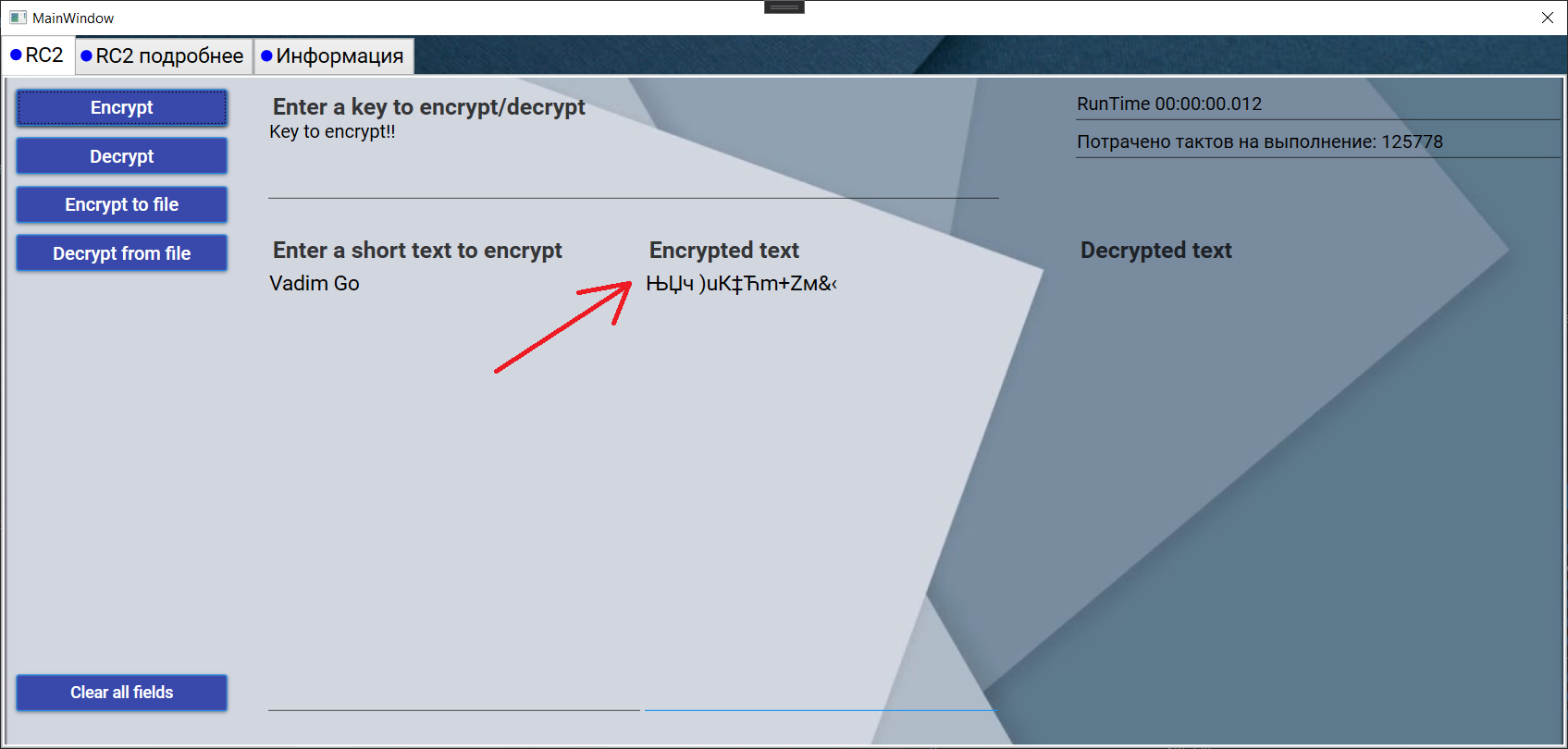


Рисунок 6.5 – Зашифрованное значение

Далее, чтобы расшифровать зашифрованную строку, нужно нажать на кнопку «Decrypt», которая показана на рисунке 6.6.

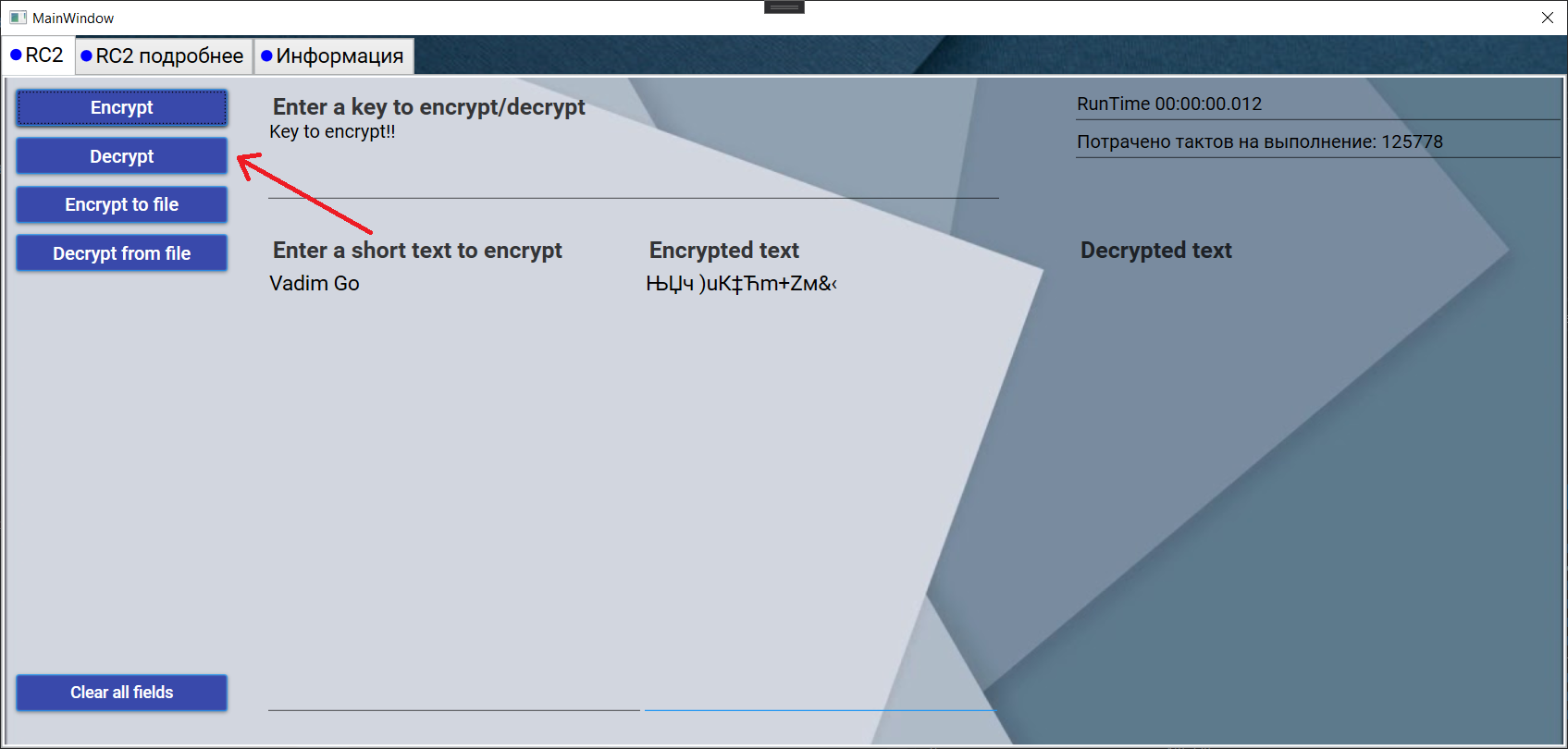


Рисунок 6.6 – Нажатие на кнопку «Decrypt»

После того как нажали на кнопку расшифровать, в поле «Decrypted text» будет отображена расшифрованная исходная строка. Результат на рисунке 6.7.

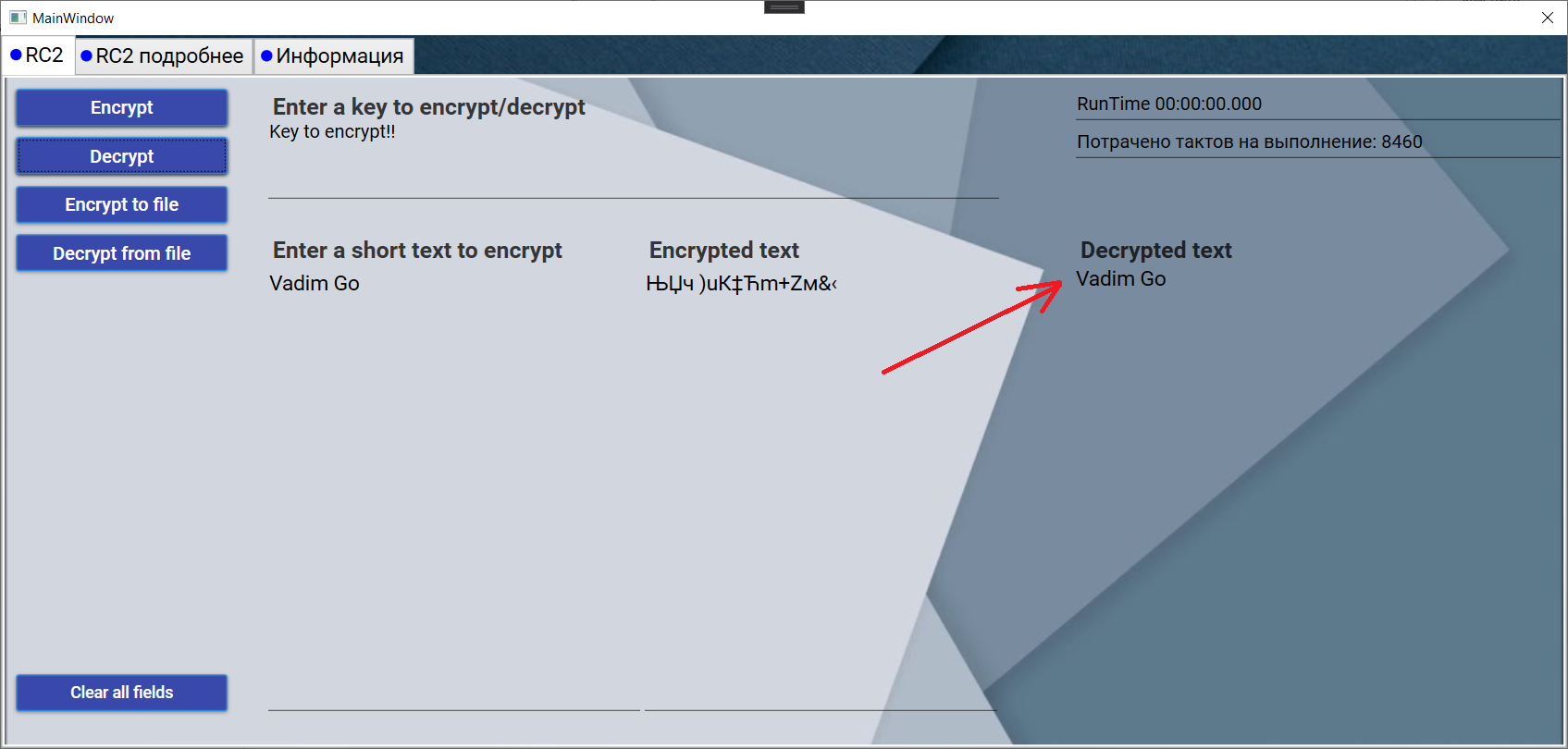


Рисунок 6.7 – Отображение расшифрованной исходной строки

В программе также можно посмотреть информация по алгоритму RC2. Если пользователь перейдет на вкладку «Информация», то увидит краткие сведения об алгоритме, а также куда перейти для выполнения шифрования. Интерфейс вкладки предоставлен на рисунке 6.8.

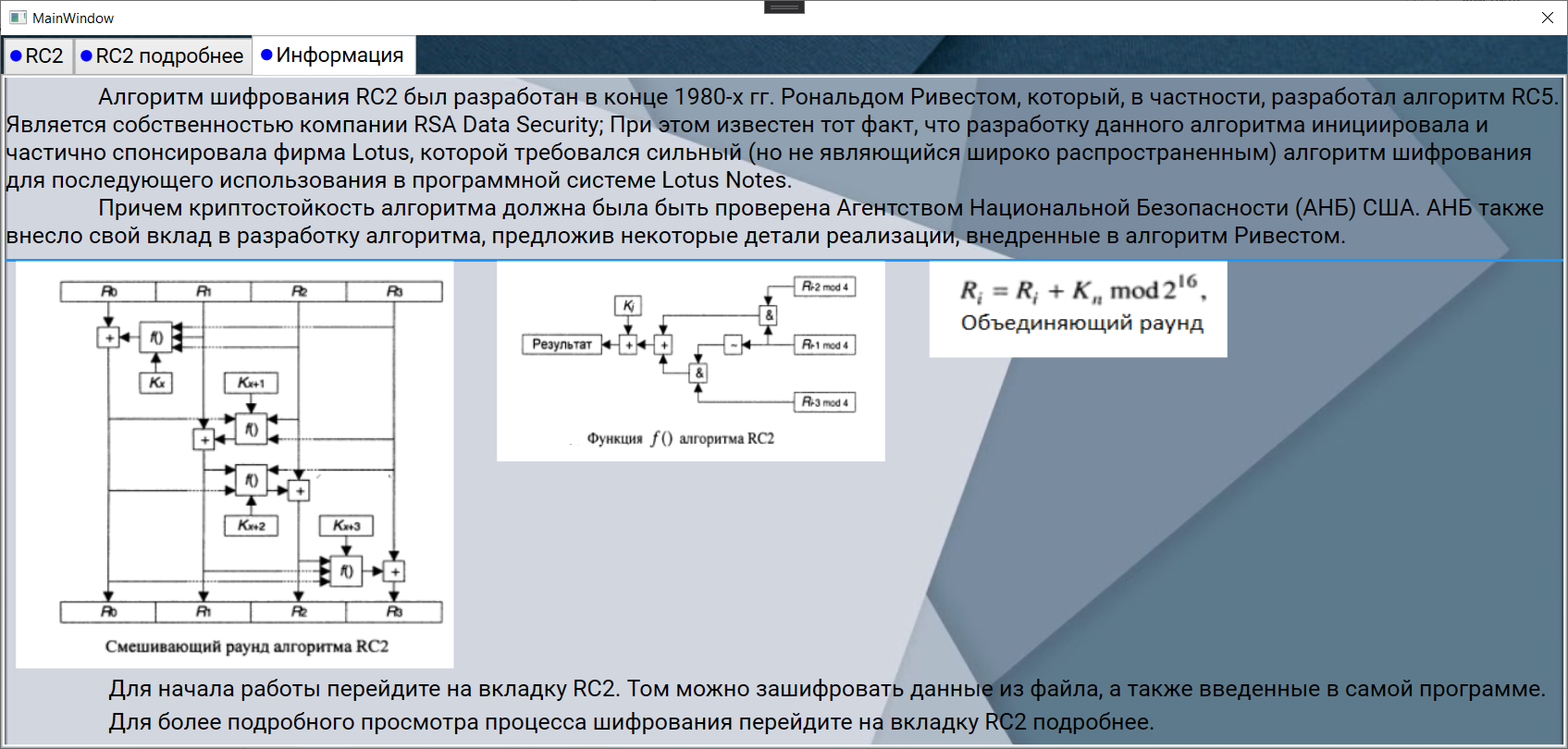


Рисунок 6.8 – Вкладка с информацией об алгоритме

На случай, если пользователь захочет посмотреть подробный анализ шифруемого сообщения, он может перейти на вкладку «RC2 подробнее», где можно ввести ключ и данные для шифрования и подробно изучить каждый раунд алгоритма. Интерфейс вкладки предоставлен на рисунке 6.9.



Рисунок 6.9 – Вкладка «RC2 подробнее»

В этой вкладке будут выведены результаты смешивающих и объединяющих раундов. Для начала работы нужно ввести соответствующие данные и нажать кнопку «Encrypt». После нажатия результат будет выведен на экран, как на рисунке 6.10.

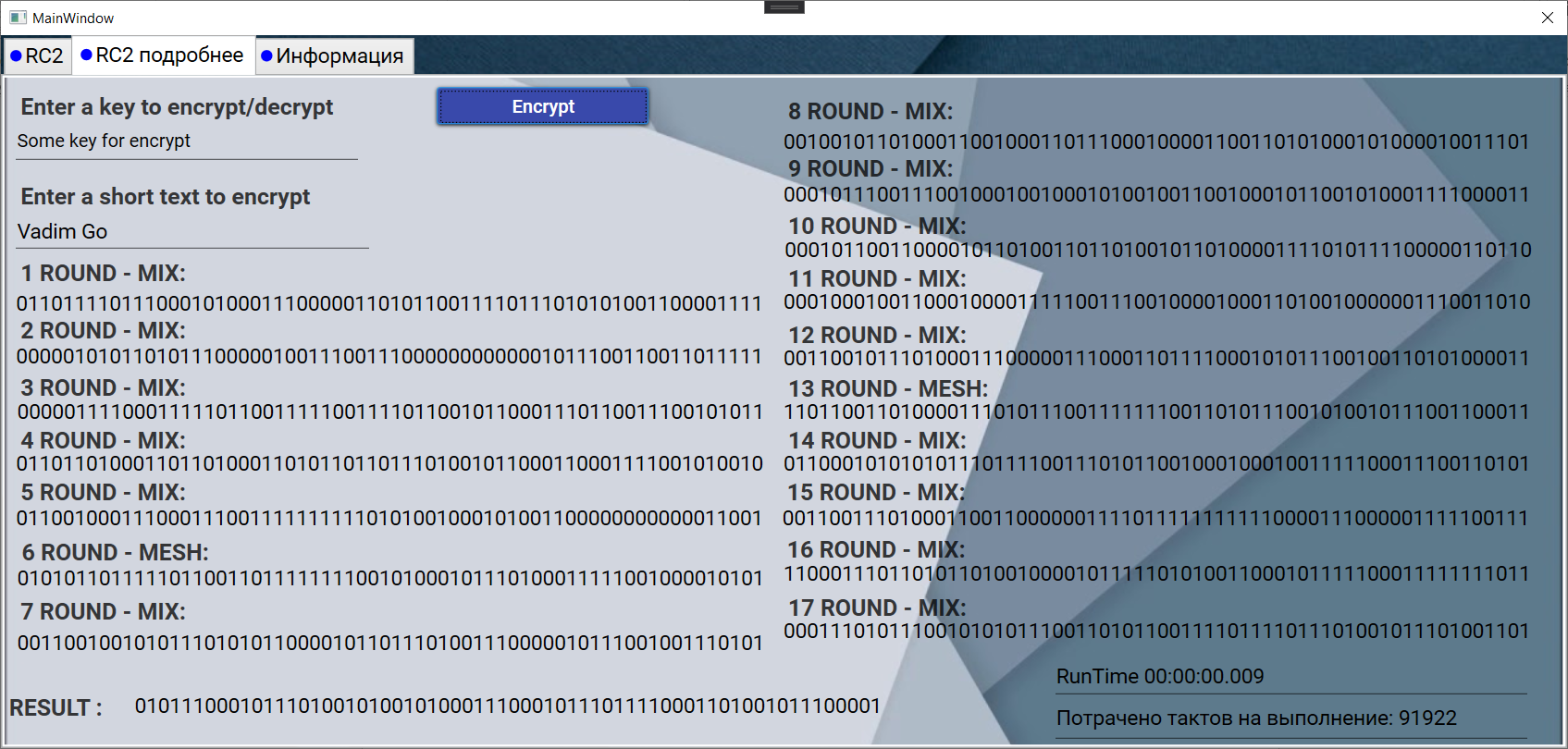


Рисунок 6.10 – Вкладка «RC2 подробнее» с результатом

Для выхода из программы следует нажать крестик в правом верхнем углу главной формы приложения, как показано на рисунке 6.11.



Рисунок 6.11 – Выход из приложения

# Заключение

В результате выполнения данной работы было разработано программное обеспечение, позволяющее шифровать и расшифровывать строку по алгоритму RC2.

Программа может:

* шифровать строку при помощи алгоритма RC2;
* расшифровывать строку;
* отображать ввод закрытого ключа;
* отображать ввод строки для шифрования;
* отображать результаты шифрования;
* отображать результаты расшифрования;
* отображение строки вектора инициализации.

В результате курсовой работы были выполнены следующие задачи:

1. Был проведен аналитический обзор литературы а также поиск и сравнение аналогов.
2. Было спроектировано и разработано программное средство.
3. Разработанное программное средство было описано.
4. Проведено тестирование разработанного программного средства.
5. Написано руководство пользователя разработанного программного средства.

Программу можно усовершенствовать путем добавления других алгоритмов шифрования, к примеру DES, AES и другие. Поэтому поле «Способ шифрования» в программе неактивно и там по умолчанию выбран алгоритм RC2. После добавлений других алгоритмов шифрования данное поле можно активировать, для отображения списка добавленных алгоритмов.

# Список используемых источников

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

# ПРИЛОЖЕНИЕ А

**MainWindow.cs**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Media.Imaging;  using System.Windows.Navigation;  using System.Windows.Shapes;  using System.IO;  using System.Diagnostics;  namespace RC2  {  public partial class MainWindow : Window  {  public MainWindow()  {  InitializeComponent();  }    private void CheckEncr(object sender, RoutedEventArgs e)  {  if (TextToEncryptMore.Text.Length > 8)  {  MessageBox.Show("Для этого раздела текст для шифрования должен состоять не более, чем из 8 букв");  }  else if(KeyToEncryptMore.Text.Length < 5)  {  MessageBox.Show("Ключ должен быть не менее 5 символов!");  }  else if (String.IsNullOrEmpty(TextToEncryptMore.Text))  {  MessageBox.Show("Введите текст для шифрования");  }  else  {  Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  int j = 0;  string round1 = MixRound(TextToBinaryText(TextToEncryptMore.Text), Key(KeyToEncryptMore.Text), j);  Round1.Text = round1;  Console.WriteLine("ROUND 1 = " + round1);  string round2 = MixRound(round1, Key(KeyToEncryptMore.Text), j);  Round2.Text = round2;  Console.WriteLine("ROUND 2 = " + round2);  string round3 = MixRound(round2, Key(KeyToEncryptMore.Text), j);  Round3.Text = round3;  Console.WriteLine("ROUND 3 = " + round3);  string round4 = MixRound(round3, Key(KeyToEncryptMore.Text), j);  Round4.Text = round4;  Console.WriteLine("ROUND 4 = " + round4);  string round5 = MixRound(round4, Key(KeyToEncryptMore.Text), j);  Round5.Text = round5;  Console.WriteLine("ROUND 5 = " + round5);  string round6 = MeshRound(round5, Key(KeyToEncryptMore.Text));  Round6.Text = round6;  Console.WriteLine("ROUND 6 = " + round6);  string round7 = MixRound(round6, Key(KeyToEncryptMore.Text), j);  Round7.Text = round7;  Console.WriteLine("ROUND 7 = " + round7);  string round8 = MixRound(round7, Key(KeyToEncryptMore.Text), j);  Round8.Text = round8;  Console.WriteLine("ROUND 8 = " + round8);  string round9 = MixRound(round8, Key(KeyToEncryptMore.Text), j);  Round9.Text = round9;  Console.WriteLine("ROUND 9 = " + round9);  string round10 = MixRound(round9, Key(KeyToEncryptMore.Text), j);  Round10.Text = round10;  Console.WriteLine("ROUND 10 = " + round10);  string round11 = MixRound(round10, Key(KeyToEncryptMore.Text), j);  Round11.Text = round11;  Console.WriteLine("ROUND 11 = " + round11);  string round12 = MixRound(round11, Key(KeyToEncryptMore.Text), j);  Round12.Text = round12;  Console.WriteLine("ROUND 12 = " + round12);  string round13 = MeshRound(round12, Key(KeyToEncryptMore.Text));  Round13.Text = round13;  Console.WriteLine("ROUND 13 = " + round13);  string round14 = MixRound(round13, Key(KeyToEncryptMore.Text), j);  Round14.Text = round14;  Console.WriteLine("ROUND 14 = " + round14);  string round15 = MixRound(round14, Key(KeyToEncryptMore.Text), j);  Round15.Text = round15;  Console.WriteLine("ROUND 15 = " + round15);  string round16 = MixRound(round15, Key(KeyToEncryptMore.Text), j);  Round16.Text = round16;  Console.WriteLine("ROUND 16 = " + round16);  string round17 = MixRound(round16, Key(KeyToEncryptMore.Text), j);  Round17.Text = round17;  Console.WriteLine("ROUND 17 = " + round17);  string round18 = MixRound(round17, Key(KeyToEncryptMore.Text), j);  Result.Text = round18;  Console.WriteLine("Result 18 = " + round17);  stopwatch.Stop();  var resultTime = stopwatch.Elapsed;  // elapsedTime - строка, которая будет содержать значение затраченного времени  string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",  resultTime.Hours,  resultTime.Minutes,  resultTime.Seconds,  resultTime.Milliseconds);  Console.WriteLine("RunTime " + elapsedTime);  Console.WriteLine("Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks);  Timer.Text = "RunTime " + elapsedTime ;  Timer2.Text = "Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks;  }  }  private string MixRound(string textToMix, string[] K, int j)  {  string[] Rt = new string[4];  int counter = 1;  string tempString = "";  int n = 0;  for(int i=0; i<textToMix.Length; i++)  {  tempString += textToMix[i];  if (counter%16==0)  {  Rt[n] = tempString.ToString();  tempString = "";  n++;  }  counter++;  }  //Console.WriteLine(" RT0 = " + Rt[0] + " RT1 = " + Rt[1] + " RT2 = " + Rt[2] + " RT3 = " + Rt[3]);  long[] R = new long[4];  R[0] = Convert.ToInt64(Rt[0]);  R[1] = Convert.ToInt64(Rt[1]);  R[2] = Convert.ToInt64(Rt[2]);  R[3] = Convert.ToInt64(Rt[3]);  int counter2 = 1;  string tempString2 = "";  int n2 = 0;  for (int i = 0; i < textToMix.Length; i++)  {  tempString2 += textToMix[i];  if (counter2 % 16 == 0)  {  R[n2] = Convert.ToInt64(tempString2.ToString(),2);  tempString2 = "";  n2++;  }  counter2++;  }  for (int i = 0; i<4; i++)  {  switch (i)  {  case 0:  R[i] = (R[i] + Convert.ToInt64(K[j],2) + (R[3] & R[2]) + (~R[3] & R[1])) % 65536;  j++;  //Console.WriteLine("R0 = " + R[i]);  R[i] = CiclycShiftLeft(R[i], 1);  break;  case 1:  R[i] = (R[i] + Convert.ToInt64(K[j], 2) + (R[0] & R[3]) + (~R[0] & R[2])) % 65536;  j++;  //Console.WriteLine("R1 = " + R[i]);  R[i] = CiclycShiftLeft(R[i], 2);  break;  case 2:  R[i] = (R[i] + Convert.ToInt64(K[j], 2) + (R[1] & R[0]) + (~R[1] & R[3])) % 65536;  j++;  //Console.WriteLine("R2 = " + R[i]);  R[i] = CiclycShiftLeft(R[i], 3);  break;  case 3:  R[i] = (R[i] + Convert.ToInt64(K[j], 2) + (R[2] & R[1]) + (~R[2] & R[0])) % 65536;  j++;  //Console.WriteLine("R3 = " + R[i]);  R[i] = CiclycShiftLeft(R[i], 5);  break;  }  //R[i] = R[i] + (R[i - 1] & R[i - 2]) + (~R[i - 1] & R[i - 3]) % 65536;  }  Rt[0] = R[0].ToString();  Rt[1] = R[1].ToString();  Rt[2] = R[2].ToString();  Rt[3] = R[3].ToString();  string mixedText = Rt[0] + " " + Rt[1] + " " + Rt[2] + " " + Rt[3];  string[] Rtemp = new string[4];  Rtemp[0] = Convert.ToString(R[0], 2);  Rtemp[1] = Convert.ToString(R[1], 2);  Rtemp[2] = Convert.ToString(R[2], 2);  Rtemp[3] = Convert.ToString(R[3], 2);  for (int i = 0; i < 4; i++)  {  if (Rtemp[i].Length < 16)  {  if(Rtemp[i].Length == 15)  Rtemp[i] = "0" + Rtemp[i];  if (Rtemp[i].Length == 14)  Rtemp[i] = "00" + Rtemp[i];  if (Rtemp[i].Length == 13)  Rtemp[i] = "000" + Rtemp[i];  if (Rtemp[i].Length == 12)  Rtemp[i] = "0000" + Rtemp[i];  if (Rtemp[i].Length == 11)  Rtemp[i] = "00000" + Rtemp[i];  if (Rtemp[i].Length == 10)  Rtemp[i] = "000000" + Rtemp[i];  if (Rtemp[i].Length == 9)  Rtemp[i] = "0000000" + Rtemp[i];  if (Rtemp[i].Length == 8)  Rtemp[i] = "00000000" + Rtemp[i];  if (Rtemp[i].Length == 7)  Rtemp[i] = "000000000" + Rtemp[i];  if (Rtemp[i].Length == 6)  Rtemp[i] = "0000000000" + Rtemp[i];  if (Rtemp[i].Length == 5)  Rtemp[i] = "00000000000" + Rtemp[i];  if (Rtemp[i].Length == 4)  Rtemp[i] = "000000000000" + Rtemp[i];  if (Rtemp[i].Length == 3)  Rtemp[i] = "0000000000000" + Rtemp[i];  if (Rtemp[i].Length == 2)  Rtemp[i] = "00000000000000" + Rtemp[i];  if (Rtemp[i].Length == 1)  Rtemp[i] = "000000000000000" + Rtemp[i];  }  }  mixedText = Rtemp[0] + Rtemp[1] + Rtemp[2] + Rtemp[3];  if (j == 16)  {  mixedText = End();  }  return mixedText;  }  private string MeshRound(string textToMesh, string[] K)  {  string[] Rt = new string[4];  int counter = 1;  string tempString = "";  int n = 0;  for (int i = 0; i < textToMesh.Length; i++)  {  tempString += textToMesh[i];  if (counter % 16 == 0)  {  Rt[n] = tempString.ToString();  tempString = "";  n++;  }  counter++;  }  //Console.WriteLine(" RT0 = " + Rt[0] + " RT1 = " + Rt[1] + " RT2 = " + Rt[2] + " RT3 = " + Rt[3]);  long[] R = new long[4];  R[0] = Convert.ToInt64(Rt[0], 2);  R[1] = Convert.ToInt64(Rt[1],2 );  R[2] = Convert.ToInt64(Rt[2],2 );  R[3] = Convert.ToInt64(Rt[3],2);  //Console.WriteLine(" R0 = " + R[0] + " R1 = " + R[1] + " R2 = " + R[2] + " R3 = " + R[3]);  long nn;  for (int i = 0; i < 4; i++)  {  if(i==0)  nn= R[(3) % 4 & 63]%64;  else  nn = R[(i - 1) % 4 & 63] %64;  //Console.WriteLine("R[i] : " + R[i]);  //Console.WriteLine("[nn] : " + nn);  //Console.WriteLine("K[nn] : " + K[nn]);  //Console.WriteLine("Convert.ToInt64(K[nn]) % 65536 : " + Convert.ToInt64(K[nn]) % 65536);  //Console.WriteLine(Convert.ToInt64(R[i]) + Convert.ToInt64(K[nn]) % 65536);  R[i] = (R[i] + Convert.ToInt64(K[nn],2)) % 65536;  }  for (int i = 0; i < 4; i++)  {  Rt[i] = Convert.ToString(R[i],2);  if (Rt[i].Length < 16)  {  if (Rt[i].Length == 15)  Rt[i] = "0" + Rt[i];  if (Rt[i].Length == 14)  Rt[i] = "00" + Rt[i];  if (Rt[i].Length == 13)  Rt[i] = "000" + Rt[i];  if (Rt[i].Length == 12)  Rt[i] = "0000" + Rt[i];  if (Rt[i].Length == 11)  Rt[i] = "00000" + Rt[i];  if (Rt[i].Length == 10)  Rt[i] = "000000" + Rt[i];  if (Rt[i].Length == 10)  Rt[i] = "000000" + Rt[i];  if (Rt[i].Length == 9)  Rt[i] = "0000000" + Rt[i];  if (Rt[i].Length == 8)  Rt[i] = "00000000" + Rt[i];  if (Rt[i].Length == 7)  Rt[i] = "000000000" + Rt[i];  if (Rt[i].Length == 6)  Rt[i] = "0000000000" + Rt[i];  if (Rt[i].Length == 5)  Rt[i] = "00000000000" + Rt[i];  if (Rt[i].Length == 4)  Rt[i] = "000000000000" + Rt[i];  if (Rt[i].Length == 3)  Rt[i] = "0000000000000" + Rt[i];  if (Rt[i].Length == 2)  Rt[i] = "00000000000000" + Rt[i];  if (Rt[i].Length == 1)  Rt[i] = "000000000000000" + Rt[i];  }  }  string meshedText = Rt[0] + Rt[1] + Rt[2] + Rt[3];  return meshedText;  }  private string[] Key(string startKey)  {  string[] keyMassive = new string[128];  string keybin = TextToBinaryText(startKey);  //заполнение исходным ключем массива  int counter = 1;  string tempString = "";  int n = 0;  for (int i = 0; i < keybin.Length; i++)  {  tempString += keybin[i];  if (counter % 16 == 0)  {  keyMassive[n] = tempString.ToString();  tempString = "";  n++;  }  counter++;  }  //заполнение оставшегося массива 0  for (int i = 0; i < keyMassive.Length; i++) {  if (keyMassive[i] == null)  keyMassive[i] = "00000000";  }  //размер ключа в байтах  int T = keybin.Length / 8;  //размер ключа в битах  int T1 = keybin.Length;  //эффективный размер ключа в байтах  int T8 = (T1 + 7) / 8;  //битовая маска, учитывающая остаточные биты ключа, если его размер в битах не кратен 8  long TM = 255 % (8 + T1 - 8 \* T8);  string[] table = new string[160] {  "11011001","01111000","11111001","11000100","00011001","11011101","10110101",  "11101101","00101000","11101001","11111101","01111001","01001010","10100000",  "11011000","10011101","11000110","01111110","00110111","10000011","00101011",  "01110110","01010011","10001110","01100010","01001100","01100100","10001000",  "01000100","10001011","11111011","10100010","00010111","10011010","01011001",  "11110101","10000111","10110011","01001111","00010011","01100001","01000101",  "01101101","10001101","00001001","10000001","01111101","00110010","10111101",  "10001111","01000000","11101011","10000110","10110111","01111011","00001011",  "11110000","10010101","00100001","00100010","01011100","01101011","01001110",  "10000010","01010100","11010110","01100101","10010011","11001110","01100000",  "10110010","00011100","01110011","01010110","11000000","00010100","10100111",  "10001100","11110001","11011100","00010010","01110101","11001010","00011111",  "00111011","10111110","11100100","11010001","01000010","00111101","11010100",  "00110000","10100011","00111100","10110110","00100110","01101111","10111111",  "00001110","11011010","01000110","01101001","00000111","01010111","00100111",  "11110010","00011101","10011011","10111100","10010100","01000011","00000011",  "11111000","00010001","11000111","11110110","10010000","11101111","00111110",  "11100111","00000110","11000011","11010101","00101111","11001000","01100110",  "00011110","11010111","00001000","11101000","11101010","11011110","10000000",  "01010010","11101110","11110111","10000100","10101010","01110010","10101100",  "00110101","01001101","01101010","00101010","10010110","00011010","11010010",  "01110001","01011010","00010101","01001001","01110100","01001011","10011111",  "11010000","01011110","00000100","00011000","10100100","11101100"  };  for (int i = T; i < 128; i++)  {  keyMassive[i] = table[(Convert.ToInt64(keyMassive[i - 1]) + Convert.ToInt64(keyMassive[i - T])%256)%160];  }  keyMassive[128 - T8] = table[Convert.ToInt64(keyMassive[128 - T8]) & TM];  for (int i = 127 - T8; i > 0; i--)  {  keyMassive[i] = table[(Convert.ToInt64(keyMassive[i + 1], 2) ^ Convert.ToInt64(keyMassive[i + T8], 2))%160];  }  string[] K = new string[64];    for(int i = 0; i < 64; i++)  {  K[i] = Convert.ToString((Convert.ToInt64(keyMassive[2 \* i]) + 256 \* Convert.ToInt64(keyMassive[2 \* i + 1])),2);  }  return K;  }  private string TextToBinaryText(string textToBinary)  {  byte[] message = Encoding.UTF8.GetBytes(textToBinary);  StringBuilder sb = new StringBuilder(message.Length \* 8);  foreach (byte b in message)  {  sb.Append(Convert.ToString(b, 2).PadLeft(8, '0'));  }  string binaryStr = sb.ToString();  return binaryStr;  }  private string BinaryTextToText(string binaryToText)  {  var stringArray = Enumerable.Range(0, binaryToText.Length / 8)  .Select(i => Convert.ToByte(binaryToText  .Substring(i \* 8, 8), 2))  .ToArray();  var str = Encoding.UTF8.GetString(stringArray);  return str;  }  RC2CryptoServiceProvider RC2alg = (RC2CryptoServiceProvider)System.Security.Cryptography.RC2.Create("RC2");  private void EncryptInMemory(object sender, RoutedEventArgs e)  {  if (KeyToEncrypt.Text.Length > 16)  {  MessageBox.Show("Введите ключ до 16 символов");  }  if (String.IsNullOrEmpty(KeyToEncrypt.Text))  {  MessageBox.Show("Введите ключ!");  }  if (!String.IsNullOrEmpty(TextToEncrypt.Text))  {  try  {  Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  string sData = TextToEncrypt.Text;  byte[] key = Encoding.ASCII.GetBytes(KeyToEncrypt.Text);  byte[] Data = EncryptTextToMemory(sData, key, RC2alg.IV);  EncryptedText.Text = Encoding.Default.GetString(Data);  //BitConverter.ToString(Data);  stopwatch.Stop();  var resultTime = stopwatch.Elapsed;  // elapsedTime - строка, которая будет содержать значение затраченного времени  string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",  resultTime.Hours,  resultTime.Minutes,  resultTime.Seconds,  resultTime.Milliseconds);  Console.WriteLine("RunTime " + elapsedTime);  Console.WriteLine("Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks);  FirstTimer.Text = "RunTime " + elapsedTime;  FirstTimer2.Text = "Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks;  }  catch (Exception ee)  {  Console.WriteLine(ee.Message);  }  }  else  {  MessageBox.Show("Введите текст для шифрования");  }  }  private void DecryptInMemory(object sender, RoutedEventArgs e)  {  if (!String.IsNullOrEmpty(EncryptedText.Text))  {  try  {  Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  string sData = TextToEncrypt.Text;  byte[] key = Encoding.ASCII.GetBytes(KeyToEncrypt.Text);  byte[] Data2 = Encoding.Default.GetBytes(EncryptedText.Text);  DecryptedText.Text = DecryptTextFromMemory(Data2, key, RC2alg.IV);  stopwatch.Stop();  var resultTime = stopwatch.Elapsed;  // elapsedTime - строка, которая будет содержать значение затраченного времени  string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",  resultTime.Hours,  resultTime.Minutes,  resultTime.Seconds,  resultTime.Milliseconds);  Console.WriteLine("RunTime " + elapsedTime);  Console.WriteLine("Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks);  FirstTimer.Text = "RunTime " + elapsedTime;  FirstTimer2.Text = "Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks;  }  catch (Exception ee)  {  Console.WriteLine(ee.Message);  }  }  else  {  MessageBox.Show("Введите текст для расшифровки");  }  }  private void EncryptToFile(object sender, RoutedEventArgs e)  {  if (KeyToEncrypt.Text.Length > 16)  {  MessageBox.Show("Введите ключ до 16 символов");  }  if (String.IsNullOrEmpty(KeyToEncrypt.Text))  {  MessageBox.Show("Введите ключ!");  }  try  {  Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  string sData = TextToEncrypt.Text;  string FileName = "Encrypted text.txt";  byte[] key = Encoding.ASCII.GetBytes(KeyToEncrypt.Text);  EncryptTextToFile(sData, FileName, key, RC2alg.IV);  stopwatch.Stop();  var resultTime = stopwatch.Elapsed;  // elapsedTime - строка, которая будет содержать значение затраченного времени  string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",  resultTime.Hours,  resultTime.Minutes,  resultTime.Seconds,  resultTime.Milliseconds);  Console.WriteLine("RunTime " + elapsedTime);  Console.WriteLine("Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks);  FirstTimer.Text = "RunTime " + elapsedTime;  FirstTimer2.Text = "Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks;  }  catch (Exception ee)  {  Console.WriteLine(ee.Message);  }  }  private void DecryptFromFile(object sender, RoutedEventArgs e)  {  string FileName = "Encrypted text.txt";  byte[] key = Encoding.ASCII.GetBytes(KeyToEncrypt.Text);  string Final = DecryptTextFromFile(FileName, key, RC2alg.IV);  using (FileStream fstream = new FileStream("Decrypted text.txt", FileMode.OpenOrCreate))  {  try  {  Stopwatch stopwatch = new Stopwatch();  stopwatch.Start();  byte[] array = System.Text.Encoding.Default.GetBytes(Final);  fstream.Write(array, 0, array.Length);  stopwatch.Stop();  var resultTime = stopwatch.Elapsed;  // elapsedTime - строка, которая будет содержать значение затраченного времени  string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",  resultTime.Hours,  resultTime.Minutes,  resultTime.Seconds,  resultTime.Milliseconds);  Console.WriteLine("RunTime " + elapsedTime);  Console.WriteLine("Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks);  FirstTimer.Text = "RunTime " + elapsedTime;  FirstTimer2.Text = "Потрачено тактов на выполнение: " + stopwatch.ElapsedTicks;  }  catch(Exception ee)  {  Console.WriteLine("Непредвиденная ошибка : " + ee);  }  }  }  private string End()  {  byte[] key = Encoding.ASCII.GetBytes(KeyToEncryptMore.Text);  byte[] Data = EncryptTextToMemory(TextToEncryptMore.Text, key, RC2alg.IV);  return Convert.ToString(Data);    }  private long CiclycShiftRight(long code, int number)  {  string BinaryCode = Convert.ToString(code, 2);  //Console.WriteLine("Bin = " + BinaryCode);  if (BinaryCode.Length == 15)  BinaryCode = "0" + BinaryCode;  if (BinaryCode.Length == 14)  BinaryCode = "00" + BinaryCode;  if (BinaryCode.Length == 13)  BinaryCode = "000" + BinaryCode;  if (BinaryCode.Length == 12)  BinaryCode = "0000" + BinaryCode;  if (BinaryCode.Length == 11)  BinaryCode = "00000" + BinaryCode;  if (BinaryCode.Length == 10)  BinaryCode = "000000" + BinaryCode;  if (BinaryCode.Length == 9)  BinaryCode = "0000000" + BinaryCode;  if (BinaryCode.Length == 8)  BinaryCode = "00000000" + BinaryCode;  if (BinaryCode.Length == 7)  BinaryCode = "000000000" + BinaryCode;  if (BinaryCode.Length == 6)  BinaryCode = "0000000000" + BinaryCode;  if (BinaryCode.Length == 5)  BinaryCode = "00000000000" + BinaryCode;  if (BinaryCode.Length == 4)  BinaryCode = "000000000000" + BinaryCode;  if (BinaryCode.Length == 3)  BinaryCode = "0000000000000" + BinaryCode;  for (int i = 0; i < number; i++)  {  if (BinaryCode[BinaryCode.Length-1] == '0')  BinaryCode = "0" + BinaryCode.Remove(BinaryCode.Length-1, 1);  if (BinaryCode[BinaryCode.Length - 1] == '1')  BinaryCode = "1" + BinaryCode.Remove(BinaryCode.Length - 1, 1);  }  //Console.WriteLine(BinaryCode);  code = Convert.ToInt64(BinaryCode, 2);  //Console.WriteLine(code);  return code;  }  private long CiclycShiftLeft(long code, int number)  {  string BinaryCode = Convert.ToString(code, 2);  //Console.WriteLine("Bin = " + BinaryCode);  if (BinaryCode.Length == 15)  BinaryCode = "0" + BinaryCode;  if (BinaryCode.Length == 14)  BinaryCode = "00" + BinaryCode;  if (BinaryCode.Length == 13)  BinaryCode = "000" + BinaryCode;  if (BinaryCode.Length == 12)  BinaryCode = "0000" + BinaryCode;  if (BinaryCode.Length == 11)  BinaryCode = "00000" + BinaryCode;  if (BinaryCode.Length == 10)  BinaryCode = "000000" + BinaryCode;  if (BinaryCode.Length == 9)  BinaryCode = "0000000" + BinaryCode;  if (BinaryCode.Length == 8)  BinaryCode = "00000000" + BinaryCode;  if (BinaryCode.Length == 7)  BinaryCode = "000000000" + BinaryCode;  if (BinaryCode.Length == 6)  BinaryCode = "0000000000" + BinaryCode;  if (BinaryCode.Length == 5)  BinaryCode = "00000000000" + BinaryCode;  if (BinaryCode.Length == 4)  BinaryCode = "000000000000" + BinaryCode;  if (BinaryCode.Length == 3)  BinaryCode = "0000000000000" + BinaryCode;  for (int i = 0; i < number; i++)  {  if (BinaryCode[0] == '0')  BinaryCode = BinaryCode.Remove(0, 1) + "0";  if (BinaryCode[0] == '1')  BinaryCode = BinaryCode.Remove(0, 1) + "1";  }  //Console.WriteLine(BinaryCode);  code = Convert.ToInt64(BinaryCode, 2);  //Console.WriteLine(code);  return code;  }  private void ClearAllFielsButton(object sender, RoutedEventArgs e)  {  EncryptedText.Text = DecryptedText.Text = TextToEncrypt.Text = KeyToEncrypt.Text = "";  }  }  } |