Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА**

по дисциплине Защита информации и надежность информационных систем

Тема: Реализация симметричного алгоритма шифрования RC5

Исполнитель

|  |  |  |
| --- | --- | --- |
| cтудент 3 курса группы 2    Руководитель |  | \_\_\_\_\_\_\_\_\_  подпись, дата |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата |
| Курсовой проект защищен с оценкой |  | \_\_\_\_\_\_\_\_\_ |
| Руководитель \_\_\_\_\_\_\_ |  |  |

подпись

Минск 2021

**Оглавление**

[Введение 3](#_Toc69651593)

[1. Проектирование 4](#_Toc69651594)

[1.1 Постановка задачи 4](#_Toc69651595)

[2. Описание метода 5](#_Toc69651596)

[2.1 Алгоритм шифрования RC5 5](#_Toc69651597)

[2.2 Процедура расширения ключа 5](#_Toc69651598)

[2.3 Шифрование, расшифрование и свойства 7](#_Toc69651599)

[2.4 Варианты алгоритма 8](#_Toc69651600)

[2.5 Криптоанализ алгоритма 9](#_Toc69651601)

[3. Безопасность и криптостойкость алгоритма 12](#_Toc69651602)

[4. Описание программного средства 13](#_Toc69651603)

[5. Тестирование программного средства 16](#_Toc69651604)

[6. Руководство пользователя 18](#_Toc69651605)

[Заключение 19](#_Toc69651606)

[Список используемых источников 20](#_Toc69651607)

[ПРИЛОЖЕНИЕ А 21](#_Toc69651608)

[ПРИЛОЖЕНИЕ Б 26](#_Toc69651609)

## **Введение**

На сегодняшний день всё большую роль играют вычислительные машины, и в целом электронные средства передачи, обработки и хранения информации.

Для того, чтобы можно было использовать различные информационные технологии в разного рода областях, требуется обеспечить их надёжность и безопасность. Под безопасностью понимается способность информационной системы сохранять свою целостность и работоспособность при случайных или преднамеренных внешних воздействиях[3]. Поэтому широкое использование различных информационных технологий привело к бурному развитию самых различных методов защиты информации.

Простейшие способы шифрования появились очень давно, однако научный подход к исследованию и разработке криптографических методов появился только в прошлом веке. К настоящему времени криптография содержит множество теорем и алгоритмов. Занятие криптографией невозможно без серьезной математической подготовки. Особенно необходимы знания в области дискретной математики, теории чисел, абстрактной алгебры и теории алгоритмов. Вместе с тем не следует забывать, что криптографические методы предназначены в первую очередь для практического применения, а теоретически стойкие алгоритмы могут оказаться незащищенными перед атаками, не предусмотренными математической моделью. Поэтому после анализа абстрактной математической модели всегда необходим анализ полученного алгоритма с учетом ситуации, в которой он будет использоваться на практике.

1. **Проектирование**
   1. **Постановка задачи**

В целом, задачу можно сформулировать следующим образом: необходимо изучить литературу, касающуюся описания симметричного алгоритма шифрования RC5. Ознакомится уже с имеющимися материалами реализации данного алгоритма, псевдокодом реализации механизмов зашифрования и дешифрования входного сообщения. Поставленная задача поможет в реализации программного обеспечения для зашифрования и дешифрования входного сообщения.

Функционально должны быть выполнены следующие задачи:

* разработать программное средство для реализации механизмов зашифрования и дешифрования входного сообщения с использованием симметричного алгоритма шифрования RC5;
* провести тестирование программного средства;
* составить руководство пользователя.

Программное средство будет написано на языке C# в среде программирования Visual Studio 2019. В качество графического интерфейса будет использоваться интерфейс программирования приложений – Windows Forms, где Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework[1]. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счёт создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависящие от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++ и др.

1. **Описание метода**
   1. Алгоритм шифрования RC5

В криптографии RC5 представляет собой симметричный ключ блочного шифра, отличающегося своей простотой[4]. Данный шифр был разработан Рональдом Ривестом в 1994 году, RC означает «Rivest Cipher» или, альтернативно, «Код Рона».

В отличие от многих схем, RC5 имеет переменный размер блока (32, 64 или 128 бит), размер ключа (от 0 до 2040 бит) и количество раундов (от 0 до 255). Первоначально предложенный выбор параметров был размером блока 64 бита, 128-битным ключом и 12 раундами. Ключевой особенностью RC5 является использование ротации, зависящей от данных.

Существует несколько различных вариантов алгоритма, в которых преобразования в «пол-раундах» классического RC5 несколько изменены. В классическом алгоритме используются три примитивных операции и их инверсии:

* сложение по модулю 2w;
* побитовое исключающее ИЛИ;
* операции циклического сдвига на переменное число бит(x<<<y).

Шифрование по алгоритму RC5 состоит из двух этапов. Процедура расширения ключа и непосредственное шифрование. Для расшифрования выполняется сначала процедура расширения ключа, а затем операции, обратные процедуре шифрования. Все операции сложения и вычитания выполняются по модулю 2w.

Так как алгоритм RC5 имеет переменные параметры, то для спецификации алгоритма с конкретными параметрами принято обозначение RC5-W/R/b, где данные параметры означают:

* W – половина длины блока в битах, возможные значения 16, 32 и 64. Для эффективной реализации величину W рекомендуют брать равным машинному слову. Например, для 32-битных платформ оптимальным будет выбор W = 32, что соответствует размеру блока 64 бита;
* R – число раундов, возможные значения от 0 до 255. Увеличение числа раундов обеспечивает увеличение уровня безопасности шифра. Так, при R=0 информация шифроваться не будет. Также алгоритм RC5 использует таблицу расширенных ключей размера 2(R+1) слов, которая получается из ключа, заданного пользователем;
* b – длина ключа в байтах, возможные значения от 0 до 255.
  1. **Процедура расширения ключа**

Перед непосредственно шифрованием или расшифрованием данных выполняется процедура расширения ключа. Процедура генерации ключа состоит из четырёх этапов:

* генерация констант;
* разбиение ключа на слова;
* построение таблицы расширенных ключей;
* перемешивание.

Для заданного параметра W генерируются две псевдослучайные величины, используя две математические константы: экспоненту(e) и золотое сечение(f).

, (2.2.1)

где Odd() – это округление до ближайшего нечётного целого.

Для w = 16, 32, 64 получаются следующие константы:

* P16 = B7E116;
* Q16 = 9E3716;
* P32 = B7E1516316;
* Q32 = 9E3779B916;
* P64 = B7E151628AED2A6B16;
* Q64 = 9E3779B97F4A7C1516.

На этапе разбиения ключа происходи копирование ключа K0…Kb-1 в массив слова L0…Lc-1, где c = b/u, где u = W/8, то есть, количество байт в слове.

Если b не кратен W/8, то L дополняется нулевыми битами до ближайшего большего c, кратного W/8.

В случае если b = c = 0, то мы устанавливаем значение c = 1, а L0 = 0.

Этап построения таблицы расширенных ключей S0…S2\*(R+1)-1 выполняется следующим образом:

, (2.2.2)

На этапе перемешивания N раз выполняются следующие действия:

(2.2.3)

Причём G, H, i, j – временные переменные, начальные значения которых равно 0. Количество итераций цикла N – это максимальное из двух значений 3\* c и (3\*2\*(R+1)).

* 1. **Шифрование, расшифрование** **и свойства**

Перед первым раундом выполняются операции наложения расширенного ключа на шифруемые данные:

(2.3.1)

В каждом выполняются следующие действия:

(2.3.2)

Псевдокод алгоритма шифрования представлен на рисунке 2.3.1.

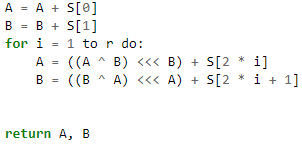


Рисунок 2.3.1 – Псевдокод алгоритма шифрования

Для расшифрование данных используются обратные операции, то есть для i = R, R – 1,…, 1 выполняются следующие раунды:

(2.3.3)

После выполнения всех раундов, исходное сообщение находится из выражения:

(2.3.4)

Псевдокод алгоритма расшифрования представлен на рисунке 2.3.2.

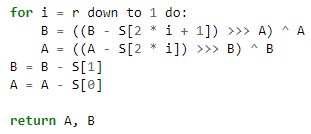


Рисунок 2.3.2 – Псевдокод алгоритма расшифрования

Алгоритм RC5 обладает следующими свойствами:

* пригодный как для аппаратной, так и для программной реализации (алгоритм использует операции, выполняющиеся одинаково быстро на всех [процессорах](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80));
* каждый раунд обрабатывает весь блок целиком (типичный раунд [сети Фейстеля](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D1%8C_%D0%A4%D0%B5%D0%B9%D1%81%D1%82%D0%B5%D0%BB%D1%8F) обрабатывает только «подблок»);
* одинаково хорош для машин с разной длиной машинного слова (то есть работает также хорошо и на 64-битных машинах).
* имеет повторяющуюся структуру с переменным числом раундов, что позволяет пользователю самому выбирать между более высокой скоростью шифрования и большей защищенностью шифра;
* имеет переменную длину ключа, что позволяет пользователю самому выбирать уровень безопасности, соответствующий специфике его приложения;
* достаточно простой в реализации и анализе.
* не требователен к памяти, что позволяет использовать его даже в мобильных и переносных устройствах.
  1. **Варианты алгоритма**

Структура алгоритма RC5, несмотря на свою простоту, представлялась многим криптологам как поле для возможных усовершенствований. Соответственно появилось множество известных вариантов алгоритма RC5, в которых преобразования в «пол-раундах» классического RC5 несколько изменены.

 Алгоритм RC5XOR, в котором сложение с ключом раунда по модулю 2 заменено операцией XOR.

(2.4.1)

Этот алгоритм оказался уязвим к дифференциальному и линейному криптоанализу. Бирюкову и Кушилевицу удалось найти атаку методом дифференциального криптоанализа для алгоритма RC5XOR-32/12/16, используя 228 выбранных открытых текстов.

Алгоритм RC5P. В этом алгоритме сложение с ключом раунда по модулю 2w заменено операцией XOR.

(2.4.2)

Алгоритм RC5PFR. В данном алгоритме циклический сдвиг осуществляется на фиксированное для данного раунда число бит, а не на переменное.

(2.4.3)

где Ri фиксированное число.

Этот алгоритм не достаточно хорошо изучен, однако предполагается,что он неустойчив к дифференциальному криптоанализу.

Алгоритм RC5KFR. В этом алгоритме число бит сдвига зависит от ключа алгоритма и от текущего раунда.

(2.4.4)

Этот алгоритм также не достаточно хорошо изучен.

Алгоритм RC5RA. В этом алгоритме число бит сдвига определяется с помощью некоторой функции от другого «подблока».

(2.4.5)

Предполагается, что алгоритм RC5RA еще более стоек к известным методам криптоанализа, чем RC5.

* 1. **Криптоанализ алгоритма**

Считается, что именно революционная идея сдвига на переменное число бит привлекла внимание криптоаналитиков к алгоритму RC5 - он стал одним из алгоритмов, наиболее изученных на предмет возможных уязвимостей.

Начало криптоанализу алгоритма RC5 было положено сотрудниками RSA Laboratories (научного подразделения фирмы RSA Data Security) Бертоном Калиски-младшим (Burton S. Kaliski Jr.) и Икван Лайзой Ин (Yiqun Lisa Yin). В период с 1995 по 1998 г. они опубликовали ряд отчетов, в которых подробно проанализировали криптостойкость алгоритма RC5. Сделанные из них выводы приведены ниже.

RC5 почти невозможно вскрыть методом линейного криптоанализа. Во многом это свойство алгоритма предопределено наличием операции циклического сдвига на переменное число бит. Однако дальнейшие исследования показали, что существует класс ключей, при использовании которых алгоритм можно вскрыть линейным криптоанализом.

Дифференциальный криптоанализ существенно более эффективен при атаках на алгоритм RC5. Калиски и Ин предложили атаку на алгоритм RC5-32/12/16, для которой требовалось 263 пар выбранных открытых текстов и соответствующих им шифртекстов. Этот результат улучшили Ларс Кнудсен (Lars R. Knudsen) и Уилли Мейер (Willi Meier), которым для атаки потребовалось 254 выбранных открытых текстов. Они же нашли несколько классов слабых ключей, упрощающих дифференциальный криптоанализ. А наилучшим результатом стал криптоаналитический метод, предложенный криптологами Алексом Бирюковым (Alex Biryukov) и Эйялом Кушилевицем (Eyal Kushilevitz), в котором необходимо 244 выбранных открытых текстов для успешной атаки. Тем не менее все описанные выше атаки не слишком практичны - для их выполнения требуется огромное число выбранных открытых текстов. Бирюков и Кушилевиц считают, что для обеспечения полной невскрываемости алгоритма дифференциальным криптоанализом достаточно выполнения 18-20 раундов вместо 12.

На основании того факта, что на ряде платформ операция циклического сдвига на переменное число бит выполняется за различное число тактов процессора, изобретатель метода вскрытия алгоритмов шифрования по времени исполнения Пол Кохер (Paul C. Kocher) высказал предположение о возможности атаки по времени исполнения на алгоритм RC5 на таких платформах. Два варианта подобной атаки были сформулированы криптоаналитиками Говардом Хейзом (Howard M. Heys) и Хеленой Хандшух (Helena Handschuh), которые показали, что секретный ключ можно вычислить, выполнив около 220 операций шифрования с высокоточными замерами времени исполнения и затем от 228 до 240 тестовых операций шифрования. Однако Калиски и Ин предложили весьма простое "противоядие" - принудительно выполнять все сдвиги за одинаковое число тактов (т. е. взять наиболее медленный из возможных сдвигов - это, несомненно, несколько снизит среднюю скорость шифрования). Аналогичную методику противодействия атакам по времени исполнения советует и сам Кохер.

Таким образом, наиболее реальный метод взлома алгоритма RC5 (не считая вариантов с небольшим числом раундов и с коротким ключом) - полный перебор возможных вариантов ключа шифрования. Это означает, что у алгоритма RC5 практически отсутствуют недостатки с точки зрения стойкости. Косвенно этот вывод подтверждается тем, что достаточно много исследований стойкости алгоритма было направлено против вариантов с усеченным числом раундов: такие варианты обычно исследуются в случае отсутствия серьезных уязвимостей у полноценных вариантов алгоритма.

1. **Безопасность и криптостойкость алгоритма**

RSA потратила много времени на анализ его работы с 64-битным блоком. Так в период с 1995 по 1998 г. они опубликовали ряд отчётов, в которых подробно проанализировали криптостойкость алгоритма RC5. Оценка для линейного криптоанализа показывает, что алгоритм безопасен после 6 раундов. Дифференциальный криптоанализ требует 224 выбранных открытых текстов для алгоритма с 5 раундами, 245 для 10 раундов, 253 для 12 раундов и 268 для 15 раундов. А так как существует всего лишь 264 возможных различных открытых текстов, то дифференциальный криптоанализ невозможен для алгоритма в 15 и более раундов. Так что рекомендуется использовать 18-20 раундов, или по крайней мере не меньше 15 вместо тех 12 раундов, которые рекомендовал сам Ривест.

Для стимуляции изучения и применения шифра RC5 RSA Security 28 января 1997 года предложила взломать серию сообщений, зашифрованных алгоритмом RC5 с разными параметрами, назначив за взлом каждого сообщения приз в $10 000. Шифр с самыми слабыми параметрами RC5-32/12/5 был взломан в течение нескольких часов. Тем не менее, последний осуществлённый взлом шифра RC5-32/12/8 потребовал уже 5 лет вычислений в рамках проекта распределённых вычислений RC5-64 (здесь 64 = b·8, длина ключа в битах) под руководством [distributed.net](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fdistributed.net). По-прежнему неприступными пока остаются RC5-32/12/b для b от 9 до 16. [distributed.net](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fdistributed.net) запустил проект RC5-72 для взлома RC5-32/12/9, в котором по состоянию на октября 2013 года удалось перебрать около 3 % ключей.

На платформах, где операция циклического сдвига на переменное число битов выполняется за различное число тактов процессора, возможна атака по времени исполнения на алгоритм RC5. Два варианта подобной атаки были сформулированы криптоаналитиками Говардом Хейзом и Хеленой Хандшух (англ. Helena Handschuh). Они установили, что ключ может быть вычислен после выполнения около 220 операций шифрования с высокоточными замерами времени исполнения и затем от 228 до 240 пробных операций шифрования. Самый простой метод борьбы с подобными атаками — принудительное выполнение сдвигов за постоянное число тактов (например, за время выполнения самого медленного сдвига).

1. **Описание программного средства**

В ходе выполнения курсового проекта для возможности взаимодействия пользователя с системой было спроектировано десктопное приложения с использованием технологии Windows Forms. Для работы с Windows Forms использовалась среда разработки Visual Studio 2019. Язык программирования, на котором было написано приложение – C#. Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счёт создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависящие от языка разработки[2].

Программное средство осуществляет выполнение следующих функций:

* зашифровывает сообщение, вписанное в текстовое поле, при помощи алгоритма шифрования «RC5»;
* вывод зашифрованного сообщения в тестовое поле;
* производит обратное преобразование.

Интерфейс программного средства представлен на рисунке 4.1.

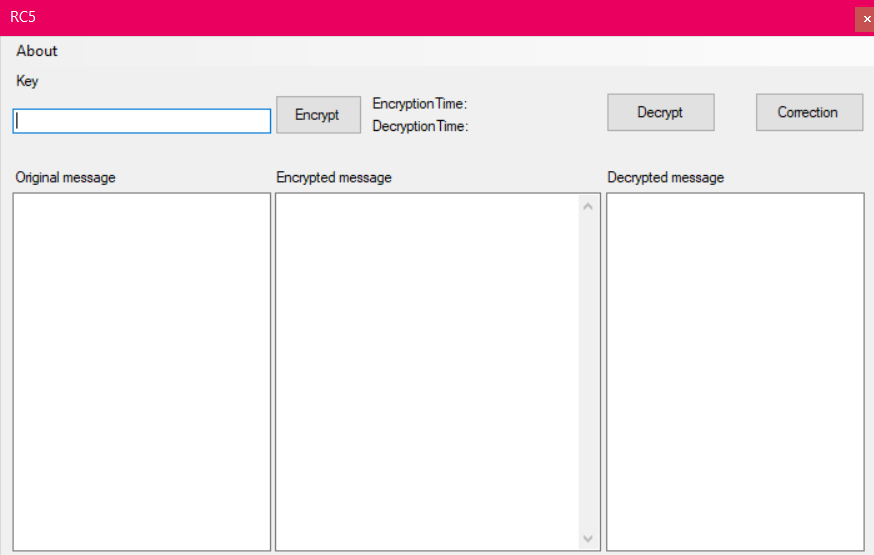


Рисунок 4.1 – Интерфейс программного средства

В верхнем левом углу находится кнопка «about», при нажатии на которую отобразится окно с информацией о размере сообщения, максимальной длине ключа и числа раундов.

Данное окно представлено на рисунке 4.2.

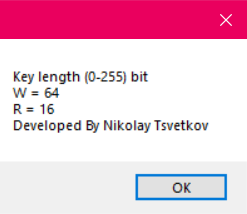


Рисунок 4.2 – Окно информации

В программном средстве предусмотрена валидация данных: проверка поля ввода ключа на пустую строку. Окно с валидацией представлено на рисунке 4.3.

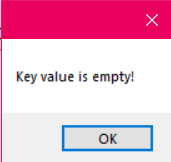


Рисунок 4.3 – Окно валидации

В программном средстве имеются кнопки для шифрования, дешифрования. Также имеются текстовые поля для ввода сообщения, вывода зашифрованного и расшифрованного сообщения и поля для вывода времени выполнения функций шифрования и дешифрования сообщения. Реализация кнопки «Encrypt» представлена на рисунке 4.4.

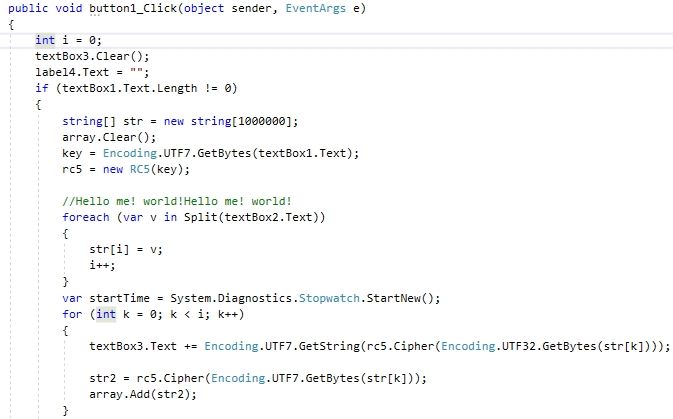


Рисунок 4.4 – Программная реализация кнопки Encrypt

Как видно из рисунка в данном методе происходит проверка поля для ввода ключа на пустоту. Затем осуществляется перевод значения ключа, полученного из текстового поля в байтовое представление. Затем осуществляется вызов функции шифрования и вывод результата её выполнения в текстовое поле. Также в этом методе происходит подсчёт времени выполнения функции шифрования.

Программная реализация кнопки дешифрования представлена на рисунке 4.5.

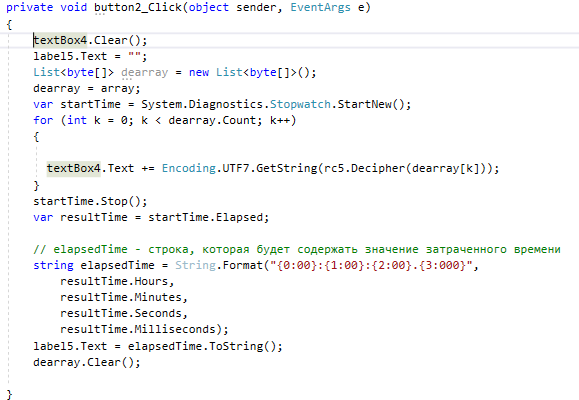


Рисунок 4.4 – Программная реализация кнопки Decrypt

Реализация кнопки «Decrypt» аналогична реализации кнопки «Encrypt». Различие заключается лишь в вызове функции дешифрования.

Так как в программном средстве происходит шифрование текстового сообщения длинной 16 байт, то требовалось добавлять необходимое количество символов, чтобы остаток от деления длины сообщения на шестнадцать давал результат равный нулю. Следовательно, после дешифрования требуется удалять ненужные символы в конце сообщения. Для этого была добавлена кнопка «Correction». Реализация данной кнопки представлена на рисунке 4.5.

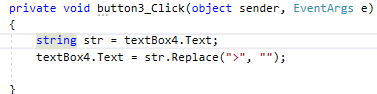


Рисунок 4.5 – Программная реализация кнопки Correction

В данном методе происходит удаление добавленного символа путём использования метода Replace.

**5. Тестирование программного средства**

Тестирование программного средства необходимо, чтобы удостовериться в том, что исходное сообщение успешно шифруется и дешифруется. Попытаемся зашифровать тестовые сообщения различных размеров и оценить время выполнения шифрования и дешифрования. Также рассмотрим вариант использования ключа, который будет отличаться от правильного на один символ. На основе описанных тестов сделаем соответствующие выводы.

На рисунке 5.1 представлено шифрование, дешифрование текстового сообщения длинной 2000 символов.

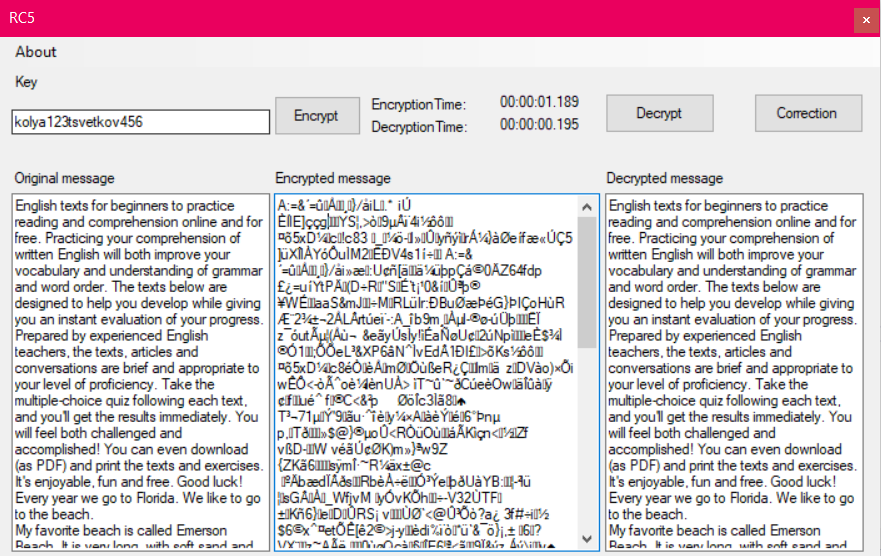


Рисунок 5.1 – Шифрование и дешифрование

Затраченное время шифрования и дешифрования составили 1.189 и 0.195 секунд. Исходя из этого можно сделать вывод, что скорость шифрования достаточно быстрая. Это объясняется тем, что алгоритм шифрования RC5 включает в себя примитивные операции: сложение, вычитание, умножение, исключающее, сдвиги.

В результате тестирования был также рассмотрен вариант использования ключа, который отличается от верного на 1 символ. Исходный ключ представлен на рисунке 5.2.

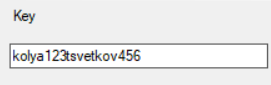


Рисунок 5.2 – Исходный ключ

Теперь введем ключ при дешифровании «kolya123tsvetkov4567», который будет отличаться от исходного ключа всего лишь последним символом.

Ввод изменённого сообщения представлен на рисунке 5.3.

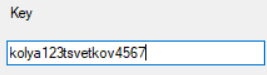


Рисунок 5.3 – Изменённый ключ

Исходя из результата расшифровки, представленного на рисунке 5.4 было установлено, что степень перемешивания бит при кодировании очень высока, т.к. всего лишь один неверный символ в ключе привел к полному отличию расшифрованного файла от исходного.

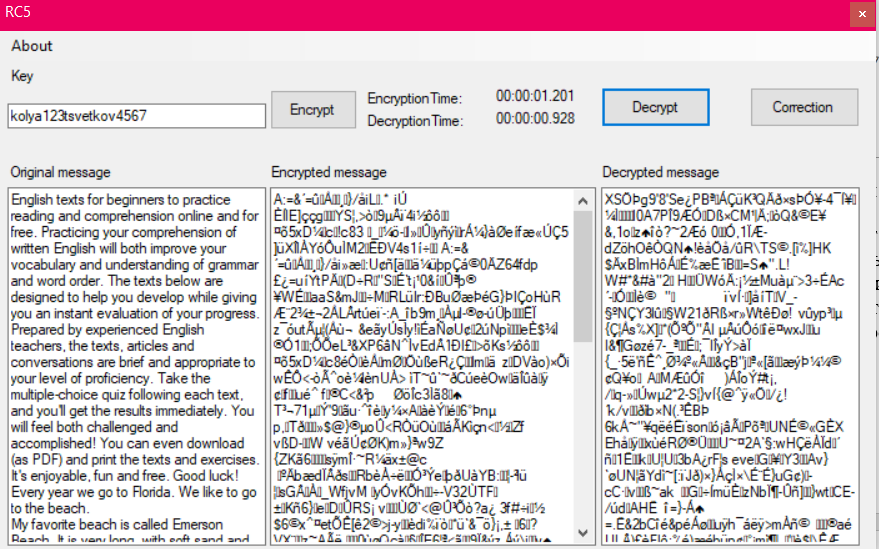


Рисунок 5.4 – Изменённый ключ

Также был рассмотрен вариант шифрования при частичной замене содержимого исходного файла с использованием того же пароля: в результате замены нескольких символов в начале исходного файла было установлено, что зашифрованные файлы резко отличаются друг от друга. Это также является подтверждением высокой степени перемешивания бит в результате работы алгоритма шифрования.

1. **Руководство пользователя**

В этом разделе будет пошагово описано взаимодействие пользователя с приложением.

Алгоритм принимает как кириллицу, так и латиницу.

Итак, после загрузки приложения пользователь попадает в основное окно программы, представленное на рисунке 6.1.

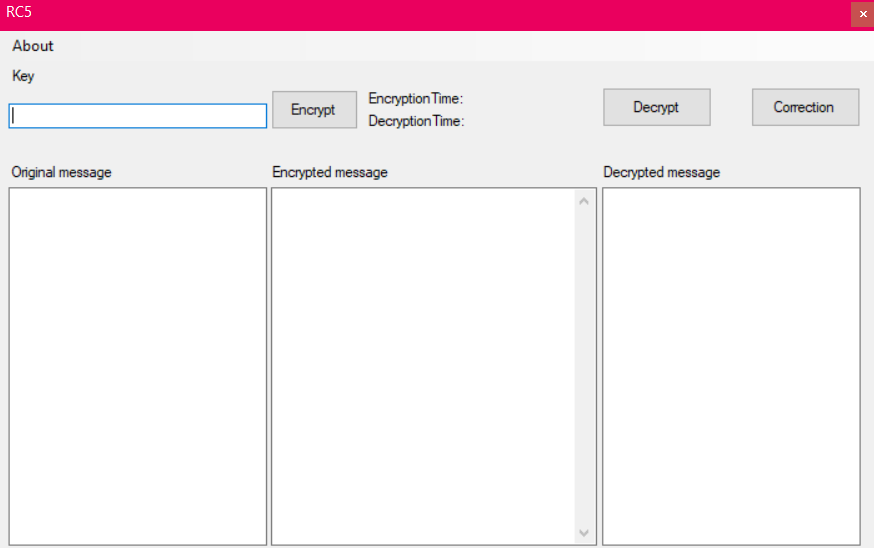


Рисунок 6.1 – Окно после загрузки программы

Чтобы использовать программное средство нужно выполнить следующие действия:

* ввести ключ в тактовое поле «Key»;
* ввести текстовое сообщение в поле «Original message»;
* нажать на кнопку «Encrypt» для зашифрования данных, после чего в текстовое поле «Encrypted message» будет выведено зашифрованное сообщение;
* нажать на кнопку «Decrypt» для расшифрования текстового сообщения, после чего расшифрованное сообщение будет выведено в тактовое поле «Decrypted message»;
* нажать на кнопку «Correction» для удаления добавочных символов.

## **Заключение**

В ходе выполнения данной курсовой работы было разработано программное средство, которое позволяет зашифровать и расшифровать текстовое сообщение при помощи алгоритма RC5.

Созданное программное средство было протестировано. Результаты тестирования показали её правильную работу. Полученные результаты работы программного средства были проанализированы и позволяют сделать вывод о том, что поставленная задача была успешно решена.

Реализация системы проводилась с использованием интегрированной среды разработки Visual Studio 2019 и технологии Windows Forms.

# **Список используемых источников**

1. Шилдт Г. Полный справочник по C# Вильямс, 2004. – 752 с
2. Рихтер Д. Программирование на платформе Microsoft .NET Framework, СПБ.: Русская редакция, 2005. – 512 с
3. М.Г. АДИГЕЕВ Введение в Криптографию Ростов – на – Дону, 2002. – 34с
4. [RSA Security’s RC5 page](https://web.archive.org/web/20061016164128/http:/www.rsasecurity.com/rsalabs/node.asp?id=2512) [Электронный ресурс] –https://ru.wikibooks.org/wiki/RC5

**ПРИЛОЖЕНИЕ А**

**RC5.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseWork

{

public class RC5

{

const int W = 64; // половина длины блока в битах.

// Возможные значения 16, 32 и 64.

// Для эффективной реализации величину W

// рекомендуют брать равным машинному слову.

// Например, для 64-битных платформ оптимальным будет

// выбор W=64, что соответствует размеру блока в 128 бит.

const int R = 16; // число раундов. Возможные значения 0…255.

// Увеличение числа раундов обеспечивает увеличение

// уровня безопасности шифра. Так, если R = 0,

// то информация шифроваться не будет.

const UInt64 PW = 0xB7E151628AED2A6B; // 64-битная константа

const UInt64 QW = 0x9E3779B97F4A7C15; // 64-битная константа

UInt64[] L; // массив слов для секретного ключа пользователя

UInt64[] S; // таблица расширенных ключей

int t; // размер таблицы

int b; // длина ключа в байтах. Возможные значения 0…255.

int u; // кол-во байтов в одном машинном слове

int c; // размер массива слов L

public RC5(byte[] key)

{

/\*

\* Перед непосредственно шифрованием или расшифровкой данных выполняется процедура расширения ключа.

\* Процедура генерации ключа состоит из четырех этапов:

\* 1. Генерация констант

\* 2. Разбиение ключа на слова

\* 3. Построение таблицы расширенных ключей

\* 4. Перемешивание

\*/

// основные переменные

UInt64 x, y;

int i, j, n;

/\*

\* Этап 1. Генерация констант

\* Для заданного параметра W генерируются две псевдослучайные величины,

\* используя две математические константы: e (экспонента) и f (Golden ratio).

\* Qw = Odd((e - 2) \* 2^W;

\* Pw = Odd((f - 1) \* 2^W;

\* где Odd() - это округление до ближайшего нечетного целого.

\*

\* Для оптимизации алгоритмы эти 2 величины определены заранее (см. константы выше).

\*/

/\*

\* Этап 2. Разбиение ключа на слова

\* На этом этапе происходит копирование ключа K[0]..K[255] в массив слов L[0]..L[c-1], где

\* c = b/u, а u = W/8. Если b не кратен W/8, то L[i] дополняется нулевыми битами до ближайшего

\* большего размера c, при котором длина ключа b будет кратна W/8.

\*/

u = W >> 3;

b = key.Length;

c = b % u > 0 ? b / u + 1 : b / u;

L = new UInt64[c];

for (i = b - 1; i >= 0; i--)

{

L[i / u] = ROL(L[i / u], 8) + key[i];

}

/\* Этап 3. Построение таблицы расширенных ключей

\* На этом этапе происходит построение таблицы расширенных ключей S[0]..S[2(R + 1)],

\* которая выполняется следующим образом:

\*/

t = 2 \* (R + 1);

S = new UInt64[t];

S[0] = PW;

for (i = 1; i < t; i++)

{

S[i] = S[i - 1] + QW;

}

/\* Этап 4. Перемешивание

\* Циклически выполняются следующие действия:

\*/

x = y = 0;

i = j = 0;

n = 3 \* Math.Max(t, c);

for (int k = 0; k < n; k++)

{

x = S[i] = ROL((S[i] + x + y), 3);

y = L[j] = ROL((L[j] + x + y), (int)(x + y));

i = (i + 1) % t;

j = (j + 1) % c;

}

}

/// <summary>

/// Циклический сдвиг битов слова влево

/// </summary>

/// <param name="a">машинное слово: 64 бита</param>

/// <param name="offset">смещение</param>

/// <returns>машинное слово: 64 бита</returns>

private UInt64 ROL(UInt64 a, int offset)

{

UInt64 r1, r2;

r1 = a << offset;

r2 = a >> (W - offset);

return (r1 | r2);

}

/// <summary>

/// Циклический сдвиг битов слова вправо

/// </summary>

/// <param name="a">машинное слово: 64 бита</param>

/// <param name="offset">смещение</param>

/// <returns>машинное слово: 64 бита</returns>

private UInt64 ROR(UInt64 a, int offset)

{

UInt64 r1, r2;

r1 = a >> offset;

r2 = a << (W - offset);

return (r1 | r2);

}

/// <summary>

/// Свертка слова (64 бит) по 8-ми байтам

/// </summary>

/// <param name="b">массив байтов</param>

/// <param name="p">позиция</param>

/// <returns></returns>

private static UInt64 BytesToUInt64(byte[] b, int p)

{

UInt64 r = 0;

for (int i = p + 7; i > p; i--)

{

r |= (UInt64)b[i];

r <<= 8;

}

r |= (UInt64)b[p];

return r;

}

/// <summary>

/// Развертка слова (64 бит) по 8-ми байтам

/// </summary>

/// <param name="a">64-битное слово</param>

/// <param name="b">массив байтов</param>

/// <param name="p">позиция</param>

private static void UInt64ToBytes(UInt64 a, byte[] b, int p)

{

for (int i = 0; i < 7; i++)

{

b[p + i] = (byte)(a & 0xFF);

a >>= 8;

}

b[p + 7] = (byte)(a & 0xFF);

}

/// <summary>

/// Операция шифрования

/// </summary>

/// <param name="inBuf">входной буфер для шифруемых данных (64 бита)</param>

/// <param name="outBuf">выходной буфер (64 бита)</param>

public byte[] Cipher(byte[] inBuf)

{

int q = 0;

UInt64 a = BytesToUInt64(inBuf, 0);

UInt64 b = BytesToUInt64(inBuf, 8);

a = a + S[0];

b = b + S[1];

for (int i = 1; i < R + 1; i++)

{

a = ROL((a ^ b), (int)b) + S[2 \* i];

b = ROL((b ^ a), (int)a) + S[2 \* i + 1];

}

int m = 0;

UInt64ToBytes(a, inBuf, 0);

UInt64ToBytes(b, inBuf, 8);

return inBuf;

}

/// <summary>

/// Операция расшифрования

/// </summary>

/// <param name="inBuf">входной буфер для шифруемых данных (64 бита)</param>

/// <param name="outBuf">выходной буфер (64 бита)</param>

public byte[] Decipher(byte[] inBuf)

{

UInt64 a = BytesToUInt64(inBuf, 0);

UInt64 b = BytesToUInt64(inBuf, 8);

for (int i = R; i > 0; i--)

{

b = ROR((b - S[2 \* i + 1]), (int)a) ^ a;

a = ROR((a - S[2 \* i]), (int)b) ^ b;

}

b = b - S[1];

a = a - S[0];

UInt64ToBytes(a, inBuf, 0);

UInt64ToBytes(b, inBuf, 8);

return inBuf;

}

}

}

**ПРИЛОЖЕНИЕ Б**

**Form1.cs**

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 CourseWork

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

byte[] key;

public RC5 rc5;

public RC5 rc5\_2;

byte[] str2 = new byte[1000000];

List<byte[]> array = new List<byte[]>();

public void button1\_Click(object sender, EventArgs e)

{

int i = 0;

textBox3.Clear();

label4.Text = "";

if (textBox1.Text.Length != 0)

{

string[] str = new string[1000000];

array.Clear();

key = Encoding.UTF7.GetBytes(textBox1.Text);

rc5 = new RC5(key);

//Hello me! world!Hello me! world!

foreach (var v in Split(textBox2.Text))

{

str[i] = v;

i++;

}

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

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

{

textBox3.Text += Encoding.UTF7.GetString(rc5.Cipher(Encoding.UTF32.GetBytes(str[k])));

str2 = rc5.Cipher(Encoding.UTF7.GetBytes(str[k]));

array.Add(str2);

}

startTime.Stop();

var resultTime = startTime.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);

label4.Text = elapsedTime.ToString();

}

else

{

MessageBox.Show("Key value is empty!");

}

}

static IEnumerable<string> Split(string str)

{

while (str.Length % 16 != 0)

{

str = str.Insert(str.Length,">");

}

while (str.Length > 0)

{

yield return new string(str.Take(16).ToArray());

str = new string(str.Skip(16).ToArray());

}

}

private void button2\_Click(object sender, EventArgs e)

{

textBox4.Clear();

label5.Text = "";

key = Encoding.UTF7.GetBytes(textBox1.Text);

rc5\_2 = new RC5(key);

List<byte[]> dearray = new List<byte[]>();

dearray = array;

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

for (int k = 0; k < dearray.Count; k++)

{

textBox4.Text += Encoding.UTF7.GetString(rc5\_2.Decipher(dearray[k]));

}

startTime.Stop();

var resultTime = startTime.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);

label5.Text = elapsedTime.ToString();

dearray.Clear();

}

private void button3\_Click(object sender, EventArgs e)

{

string str = textBox4.Text;

textBox4.Text = str.Replace(">", "");

}

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

{

MessageBox.Show("Key length (0-255) bit" + '\n' + "W = 64 bit" + '\n' + "R = 16" + '\n' + "Developed By Nikolay Tsvetkov");

}

}

}