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 симметричного алгоритма шифрования TEA»

Исполнитель

Студент 3 курса группы 1 Демидкевич А.С.

(Ф.И.О.)

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

Курсовой проект защищен с оценкой

Председатель Берников В.О.

(подпись)

Минск 2021

**Оглавление**

[Введение 3](#_Toc70550149)

[1 Аналитический обзор литературы 4](#_Toc70550150)

[1.1 Описание симметричного алгоритма шифрования TEA 4](#_Toc70550151)

[1.1.1 Генерация ключа 4](#_Toc70550152)

[1.1.2 Шифрование 4](#_Toc70550153)

[1.1.3 Расшифрование 5](#_Toc70550154)

[2 Обзор аналогов 6](#_Toc70550155)

[3 Проектирование 8](#_Toc70550156)

[4 Разработка программного средства 9](#_Toc70550157)

[5 Тестирование 11](#_Toc70550158)

[6 Руководство пользователя 13](#_Toc70550159)

[Список используемых источников 17](#_Toc70550160)

[Приложение А 18](#_Toc70550161)

[Приложение Б 20](#_Toc70550162)

# **Введение**

Обеспечение защиты информации всегда было и остается актуальным направлением. В связи с быстрым развитием информационных технологий возникает необходимость в контроле и защите информации от несанкционированного доступа, а также в шифровании данных.

Шифрование является лучшей имеющейся в наличии технологией, для защиты данных от злоумышленников. На текущий момент оно достигло такого уровня развития, что при корректном использовании его практически невозможно взломать.

Шифрование – это способ изменения сообщения или другого документа, обеспечивающее искажение (сокрытие) его содержимого.

Предметной областью данного курсового проекта является блочный алгоритм шифрования TEA. Для его использования будет разработано соответствующее программное средство, которое будет иметь максимально простой и понятный интерфейс.

Приложение предоставит пользователю возможность шифрования и расшифрования сообщения. При реализации операций шифрования и расшифрования результат будет отображаться в соответствующих окнах, где пользователь сможет наглядно проанализировать используемый метод.

Перед началом использования приложения будет проведено тестирование. Это поможет выявить большинство ошибок и определить соответствия между реальным и ожидаемым поведением ПО. Тестирование будет проведено на основе исключительных ситуаций.

Чтобы облегчить работу пользователю с программным средством или помочь в случае возникновения затруднительных ситуаций, будет составлено подробное руководство пользователя. Оно будет содержать описание выполнения важнейших операций и примеры в виде рисунков.

# **1 Аналитический обзор литературы**

## **1.1 Описание симметричного алгоритма шифрования TEA**

**TEA (Tiny Encryption Algorithm)** –блочный алгоритм шифрования типа «[Сеть Фейстеля](https://intellect.icu/set-konstruktsiya-fejstelya-9351#term-set-fejstelya)». Алгоритм был разработан на факультете компьютерных наук Кембриджского университета Дэвидом Уилером и Роджером Нидхэмом.

Алгоритм шифрования TEA основан на битовых операциях с 64-битным блоком, имеет 128-битный ключ шифрования. Стандартное количество раундов равно 32.

Достоинствами шифра являются его простота в реализации, небольшой размер кода и довольно высокая скорость выполнения. Алгоритм имеет отличную устойчивость к линейному криптоанализу и довольно хорошую к дифференциальному криптоанализу. Не обошлось и без недостатков в виде уязвимости к некоторым типам криптографических атак.  По причине простого расписания ключей главной уязвимостью алгоритма является возможность проведения «атак на связанных ключах».

Атака на связанных ключах представляет собой вид криптографической атаки, в рамках которой номинальный злоумышленник имеет возможность следить за шифрованием или расшифрованием, использующими совокупность закрытых ключей. Эта атака не осуществляется простым перебором значений возможных ключей. В самом начале точные значения ключей неизвестны злоумышленнику, но предполагается, что криптоаналитик знает некое математическое соотношение, которое связывает ключи между собой [1].

### **1.1.1 Генерация ключа**

Ключ К длиной 128 бит делится на четыре 32-битных подключа K0, K1, K2 и K3. Из-за простого расписания ключей каждый ключ имеет 3 эквивалентных ключа. Это означает, что эффективная длина ключа составляет всего 126 бит [2].

### **1.1.2 Шифрование**

Исходный текст разбивается на блоки по 64 бита каждый. При шифровании используются ранее сгенерированные четыре 32-битных подключа K0, K1, K2 и K3. Каждый 64-битный блок шифруется на протяжении 32 раундов по нижеприведенному алгоритму.

На вход раунда поступают правая и левая часть блока (L, R) по 32 бита каждая. На выходе будут левая и правая части (Ln+1, Rn+1), которые вычисляются по следующим правилам:

Левая часть: Ln+1 = Rn. Правая часть зависит от четности раунда.

Если нечётные раунды: Rn+1 = Ln + (( [ Rn << 4 ] + K0 ) XOR ( Rn + i \* δ ) XOR ( [ Rn >> 5 ] + K1 )).

Если чётные раунды: Rn+1 = Ln + (( [ Rn << 4 ] + K2 ) XOR ( Rn + i \* δ ) XOR ( [ Rn >> 5 ] + K3 )).

В процессе преобразования блока исходной информации используются три действия:

* побитовое исключающее или (XOR);
* сложение по модулю 232 (+);
* побитовые сдвиги на переменное число бит («<<», «>>»).

Также используется константа δ = 2654435769, которая была выведена из Золотого сечения.

Схема шифрования одного блока исходного сообщения показана на рисунке 1.1 [3].

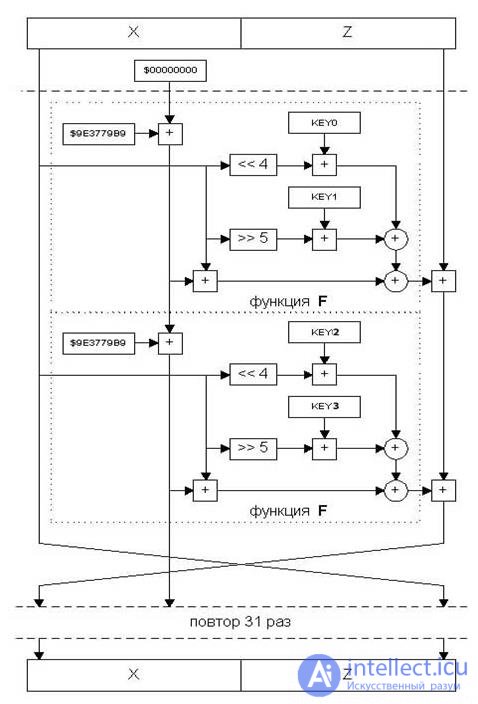


Рисунок 1.1 – Схема шифрования одного блока данных

### **1.1.3 Расшифрование**

Для расшифрования используюется аналогичный алгоритм. Единственным отличием является использование ключей в обратном порядке.

# **2 Обзор аналогов**

Выявление ряда серьезных уязвимостей и слабых мест в исходном алгоритме TEA привело к скорому созданию его расширений. Основными отличиями всех этих алгоритмов являются усовершенствованное расписание ключей, динамическая зависимость ключа от текста, а также другой размер ключа, входного блока и/или количество раундов сети Фейстеля [4].

1. XTEA

XTEA имеет размер блока, равный 64 битам, размер ключа – 128 битам, количество раундов сети Фейстеля равно 64. Алгоритм был разработан Дэвидом Уилером и Роджером Нидхэмом и опубликован в 1997 году. Главное отличие от исходного алгоритма TEA – наличие алгоритма расписания ключей, что позволило устранить критическую уязвимость для «атак на связанных ключах», но привело к ухудшению стойкости к дифференциальному криптоанализу.

1. XXTEA

В 1998 году было опубликовано следующее расширение алгоритма, получившее название XXTEA. Размер ключа – 128 бит. Отличительной особенностью является возможность шифрования любых блоков, длина которых кратна 64 битам, количество раундов равно 52 + 6\*(количество 32-битных слов в блоке) или 52 + 12\*M при длине блока 64\*M бит. Практическая эффективность опубликованной анонимно дифференциальной атаки не доказана.

1. RTEA

Существует так же альтернативная модификация алгоритма TEA, получившая наименование RTEA, разработанная в 2007 году*.* Размер блока – 64 бита; для 128-битного ключа число раундов сети Фейстеля равно 48, для 256-битного – 64. По заявлениям разработчиков этот алгоритм производительнее и более устойчив к криптоанализу, чем XTEA, однако и на этот алгоритм уже существует «атака на связанных ключах».

1. Raiden

С использованием механизмов генетического программирования в 2006 году командой разработчиков во главе с Хулио Кастро был создан алгоритм Raiden, призванный устранить уязвимости шифра TEA. Он практически в точности повторяет структуру алгоритма TEA, за исключением того, что у алгоритма Raiden есть расширенный алгоритм расписания ключей. Стандартное число раундов сети Фейстеля равно 32 (16 циклов). Raiden использует ключевое расписание, трансформирует ключ и генерирует подключи для каждого раунда. Шифр успешно проходит тесты Diehard, Sexton и ENT.

В настоящее время, в реальных условиях алгоритм TEA на практике используется редко. Одним из немногих программных средств, использующих данный алгоритм является SSuite Agnot StrongBox Security. Он использует любой из девятнадцати алгоритмов шифрования для защиты вашей информации. В них входят BlowFish, TwoFish, CAST-128, Cast-256, DES , 3DES, ICE, IDEA, MARS, RC4, Rijndael (AES) и, наконец, TEA. Уникальные пароли создаются, используя внутренний генератор случайных ключей. Поддерживаются различные размеры паролей, например, длина строки 8, 16, 24, 32 символа.

Также рассмотрим программные средства, использующие другие алгоритмы шифрования.

**AxCrypt** – простая в использовании [программа для шифровки данных](https://spy-soft.net/my-secret-folder-ios/), которая поддерживает алгоритмы шифрования AES-128 и SHA-1. Программа интегрируется с Windows, для осуществления сжатия и шифрования, дешифрования, удаления, отправки, работы с конкретными файлами.

Программное обеспечение для шифрования данных CryptoForge позволяет защитить конфиденциальные данные везде, где они используются с надежным шифрованием AES. Он имеет модуль шифрования файлов, который позволяет шифровать файлы и папки любого типа и размера на любых носителях, включая внешние USB-накопители, мобильные устройства, флэш-память и сетевые папки. Он также включает в себя уничтожитель файлов для безопасного удаления данных. Шифрование электронной почты также очень простое, поскольку получатели зашифрованных файлов не должны устанавливать программное обеспечение, чтобы иметь возможность расшифровывать файлы. Модуль шифрования текста преобразует все что угодно в зашифрованный текст. Это приложение для шифрования включает в себя четыре алгоритма шифрования: Rijndael, Blowfish, TripleDES и Gost.

CrococryptFile – это инструмент шифрования файлов, который создает зашифрованные архивы произвольных файлов и папок. Функции шифрования CrococryptFile можно сравнить с утилитой ZIP, которая использует шифрование AES ZIP. Тем не менее, есть значительные различия. CrococryptFile шифрует всю информацию о файлах и папках, включая имена файлов, время / дату и информацию о размере файлов, маскирует любую информацию о содержимом архива. Расширение .croco исключительно показывает, что это действительно архив CrococryptFile и какой криптографический набор (например, шифрование AES-256 на основе пароля) используется.

# **3 Проектирование**

Рассмотрим решение проекта «TEA», имеющим структуру, представленную на рисунке 3.1.

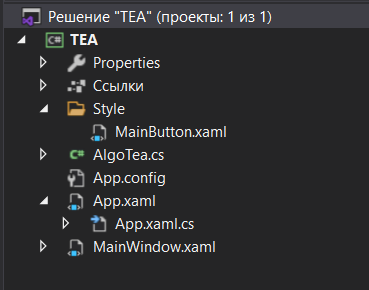


Рисунок 3.1 – Структура проекта

В директории Style находится файл MainBtton.xaml со стилями для кнопок приложения. Помимо этого в основе структуры содержатся файлы App.xaml и Mainwindow.xaml, и связанные с ними файлы кода App.xaml.cs и Mainwindow.xaml.cs соответственно. Данные файлы отвечаю за настройку и отображение непосредственно самого окна приложения.

Основная задача файла App.xaml состоит в определении ресурсов, общих для приложения. Здесь указывается то окно, которое будет запускаться при вызове приложения. В нашем случае это окно MainWindow.

Файл MainWindow.xaml представляет определение окна приложение, которое появится при запуске. В файле отделенного кода MainWindow.xaml.cs находится логика приложения, связанная с разметкой из MainWindow.xaml.

В файле AlgoTea.cs реализованы функции алгоритмов шифрования, расшифрования и генерации ключа.

# **4 Разработка программного средства**

Программное обеспечение для визуализации алгоритма шифрования и расшифрование алгоритма TEA реализовано на объектно-ориентированном языке программирования С# на платформе Microsoft .NET Framework версии 4.5 в среде разработки Microsoft Visual Studio 2019.

Для реализации программного средства использован API – интерфейс WPF для создания настольных графических программ имеющих понятный и интерактивный интерфейс.

Для разделения разработки графического интерфейса от логики приложения использовался расширяемый язык разметки – eXtensible Application Markup Language (XAML).

Визуализация алгоритмов позволит просмотреть последовательную работу алгоритмов шифрования и расшифрования, а также генерацию ключа.

Само приложение должно:

* принимать сообщение, введенное пользователем;
* принимать ключ, введенный пользователем и генерировать подключи;
* шифровать и расшифровывать сообщение, введенное пользователем;
* предоставлять визуализацию каждой функции: шифрования, расшифрования, генерации ключа.

Функции алгоритмов шифрования, расшифрования и генерации ключа представлены на рисунке 4.1. Программная реализация всех функций класса AlgoTea представлена в приложении А.

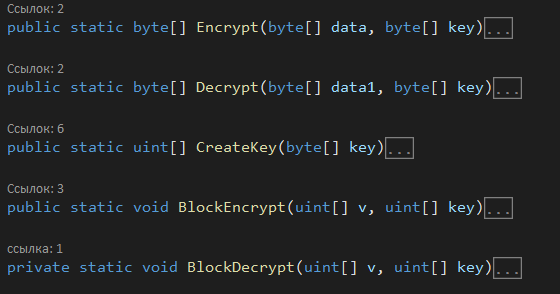


Рисунок 4.1 – Функции алгоритма

Функция Encrypt предназначена для шифрования сообщения и принимает в качестве параметров массивы байт сообщения и ключа. На выходе из функции получается зашифрованное сообщение.

В свою очередь функция Encrypt вызывает функцию BlockEncrypt, которая предназначена для шифрования отдельного блока сообщения по определенному алгоритму. Каждый блок проходит 32 раунда.

Также функция Encrypt вызывает функцию генерации подключей CreateKey.

Функция генерации 32-битных подключей CreateKey представляет собой разбиение 128-битного ключа на 4 части. Эти подключи будут использоваться в шифровании блоков сообщения.

Для расшифрования сообщения используется функция Decrypt, которая также как и функция Encrypt принимает в качестве параметров массивы байт сообщения и ключа. Данная функция вызывает функцию BlockDecrypt, для расшифрования сообщения по блокам.

Для визуализации работы алгоритма используется класс MainWindow. Функции, реализованные в нем описаны в файле MainWindow.xaml.cs, структура которого представлена на рисунке 4.2.

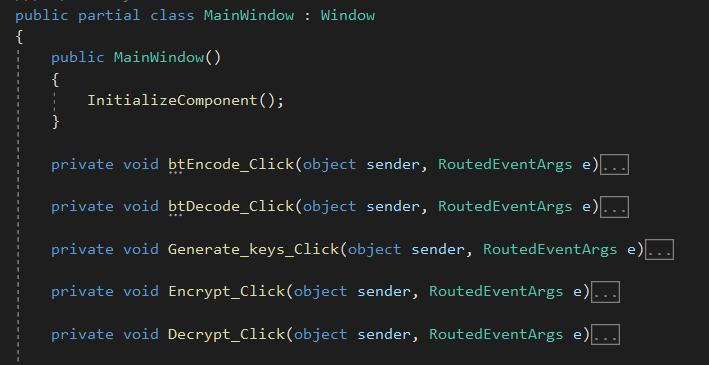


Рисунок 4.2 – Функции класса MainWindow

Функция btEncode\_Click является обработчиком события нажатия кнопки для зашифрования сообщения. Она вызывает функцию шифрования сообщения, введенного в текстовый блок с использованием подключей, сгенерированных функцией CreateKey. Функция btDecode\_Click вызывает функцию расшифрования сообщения, которое было зашифровано с помощью подключей. Функции Generate\_keys\_Click, Encrypt\_Click и Decrypt\_Click вызываются по клику соответствующих кнопок для подробного описания того, как проходит генерация ключа, шифрование и расшифрование данных. Программная реализация функций класса MainWindow представлена в приложении Б.

# **5 Тестирование**

Для корректной работы программы необходимо обеспечить защиту работы пользователя от ошибок и сбоев. Для этого используются конструкции типа «if-else» или «try-catch». Они служат для «отлавливания» исключений с посредствующей их обработкой. Это необходимо, чтобы при вводе пользователем некорректной информации, при вводе ключа неверного формата, изменение данных на непредполагаемые типы данных, программа не получала сбоев.

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

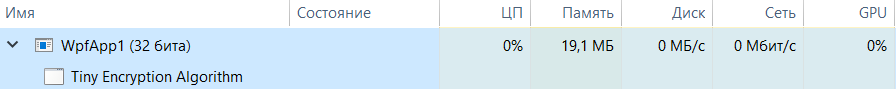


Рисунок 5.1 – Потребляемые ресурсы компьютера

Ниже приведены основные исключительные ситуации.

Перед началом работы приложения все кнопки и окна приложения находятся в неактивном состоянии, так как данные для шифрования ещё не введены (рисунок 5.2). Кнопки станут активны после того как окна «Ключ» и «Исходный текст» будут заполнены.

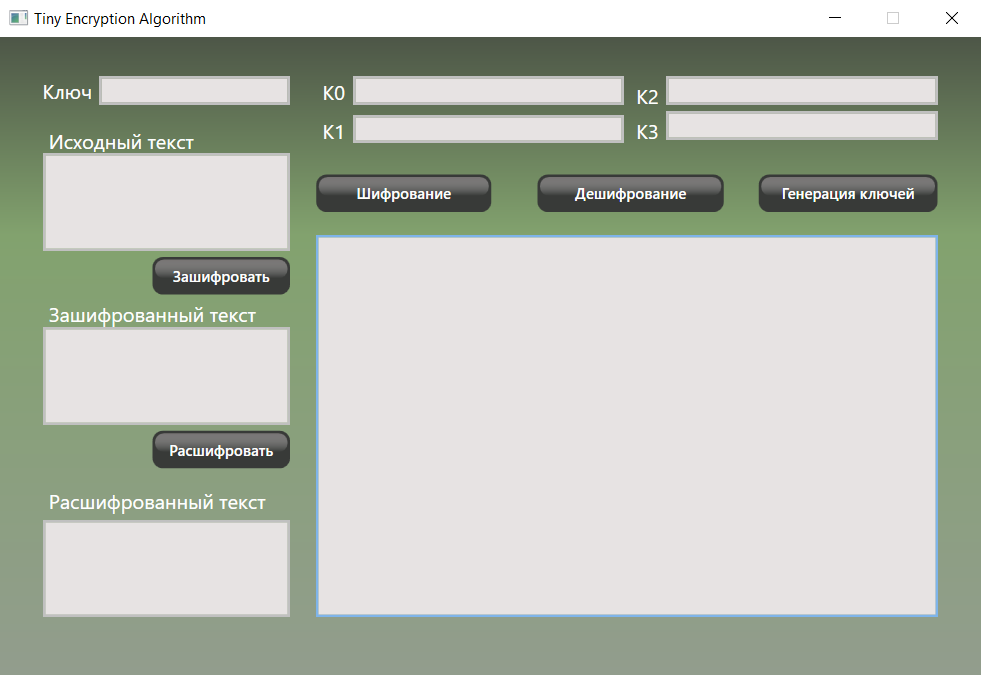


Рисунок 5.2 – Неактивные окна и кнопки

Для шифрования необходимо ввести ключ длиной 16 символов. Для этого необходимо кликнуть по полю «Ключ» и оно станет активным. При попытке ввести ключ большей или меньшей длины пользователь будет оповещен сообщение, которое представлено на рисунке 5.3. И в случае, если ключ меньше 16 символов, пользователь должен дополнить ключ недостающими знаками. Если же ключ больше 16 символов, то он будет обрезан до необходимой длины.

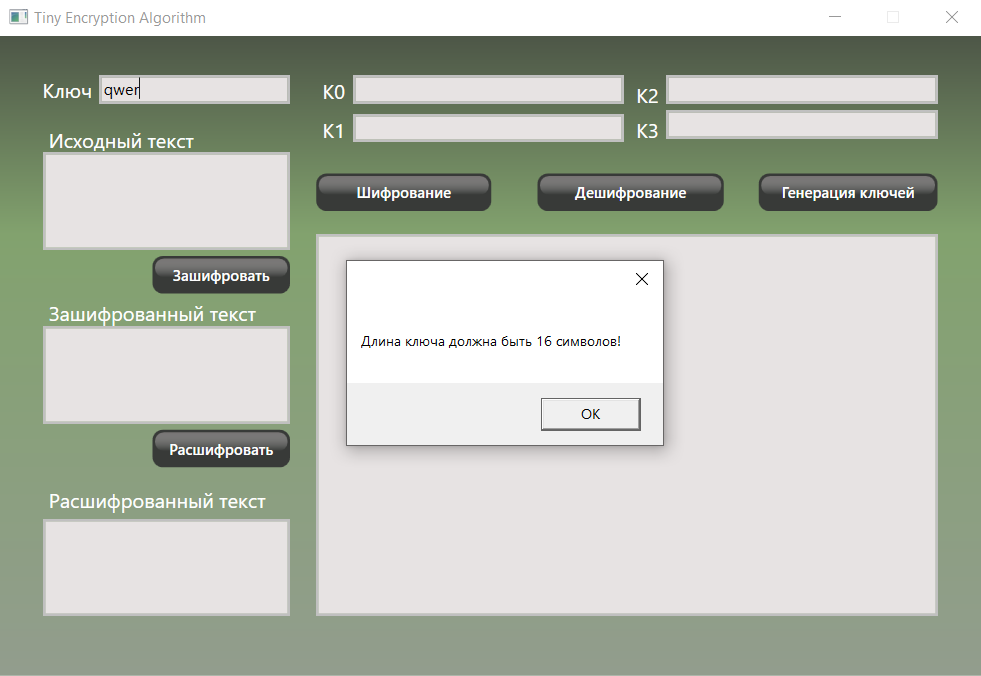


Рисунок 5.3 – Неверная длина ключа

После корректного ввода ключа окно для ввода сообщения «Исходный текст» станет активным. При этом пользователь должен ввести сообщение для последующего его шифрования. При попытке зашифровать пустое сообщение пользователь будет оповещен сообщением «Введите текст» (рисунок 5.4).

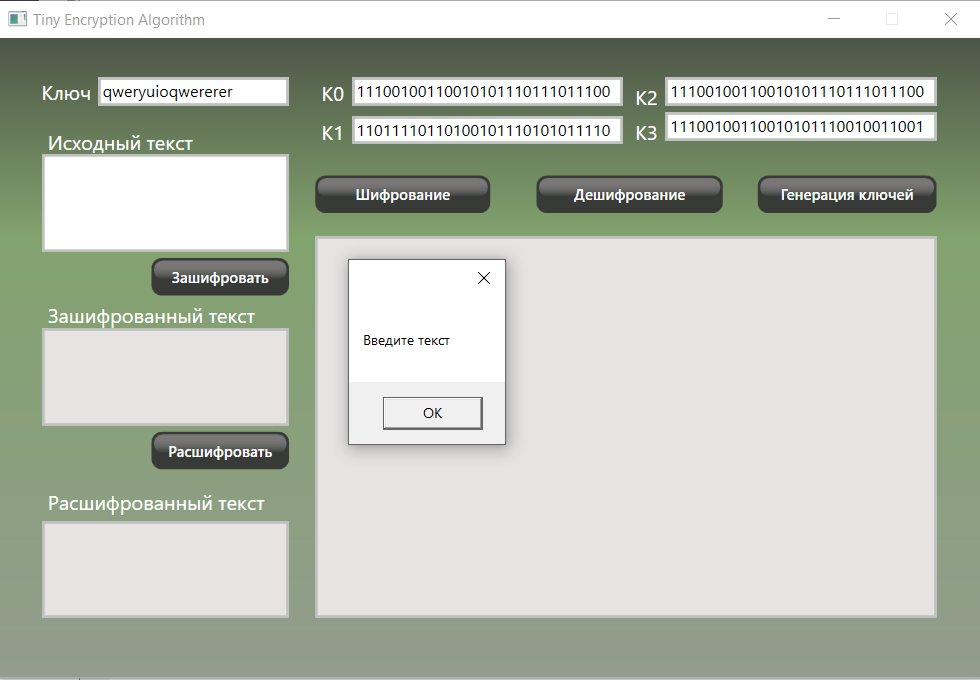


Рисунок 5.4 – Шифрование пустого сообщения

Следует отметить, что приложение разработано таким образом, чтобы минимизировать возможность появления пользовательских ошибок.

# **6 Руководство пользователя**

Чтобы запустить приложение нужно запустить exe-файл – Tiny Encryption Algorithm.exe. При запуске открывается окно приложения (рисунок 6.1). Окно содержит окна для ввода и вывода данных и кнопки для совершения операций шифрования и расшифрования.



Рисунок 6.1 – Окно приложения

После запуска приложения необходимо ввести данные в поле ключа. При этом в отдельные окна будут выведены сгенерированные 32-битные ключи, которые используются для шифрования и расшифрования (рисунок 6.2).

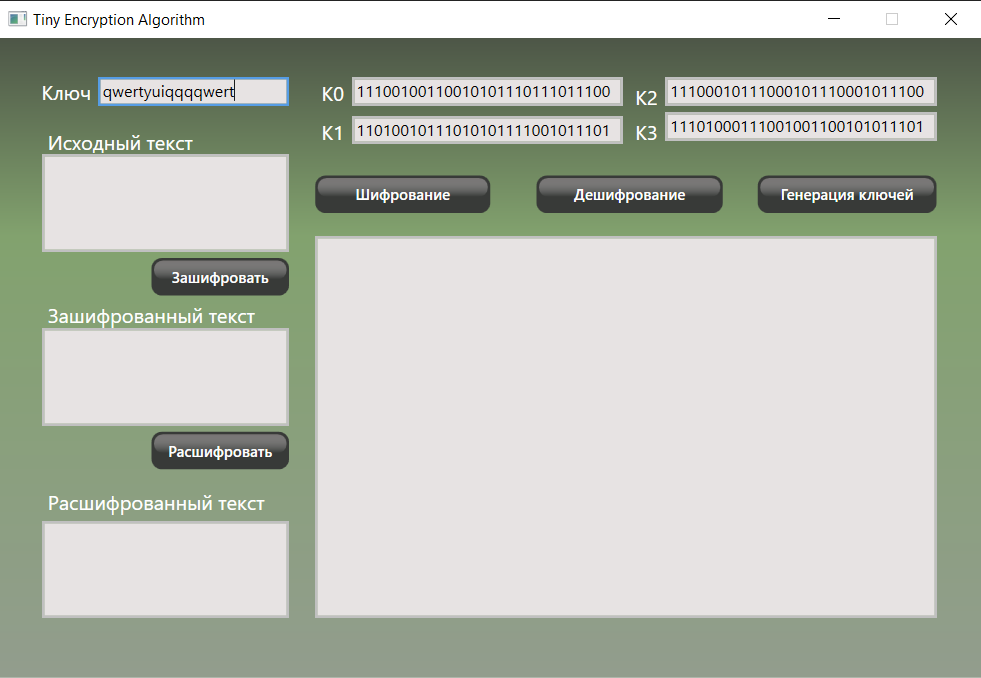


Рисунок 6.2 – Генерация подключей

В окно «Исходный текст» нужно ввести сообщение, а затем нажать кнопку «Зашифровать». При условии, что ранее был введен ключ и были сгенерированы 4 подключа, в окно «Зашифрованный текст» будет выведено зашифрованное сообщение (рисунок 6.3).

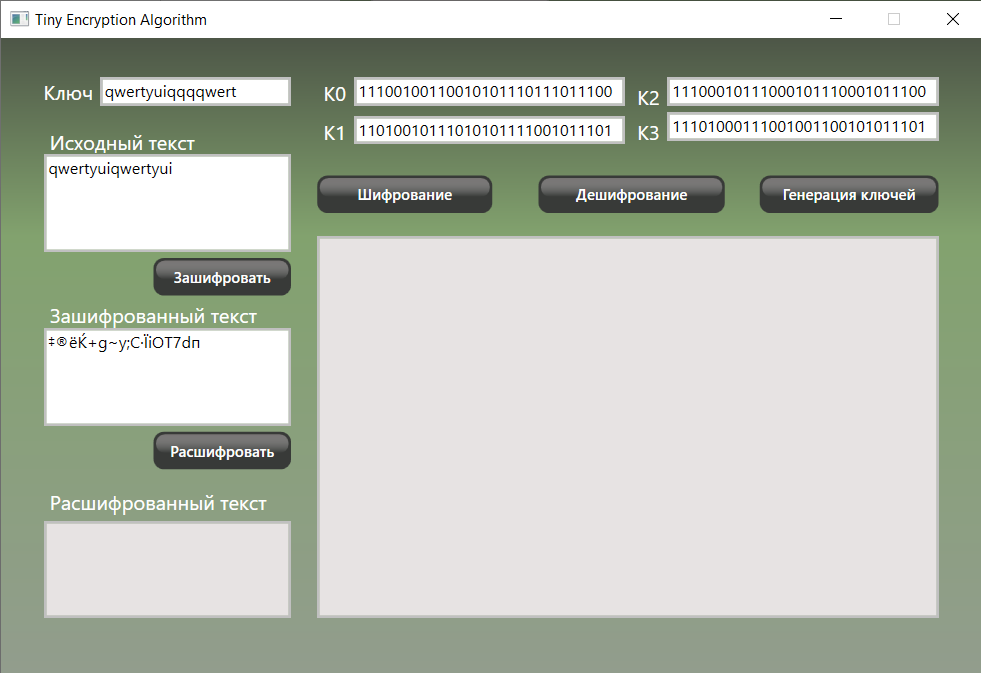


Рисунок 6.3 – Зашифрованное сообщение

После того как получили зашифрованное сообщение, расшифруем его, нажатием на кнопку «Расшифровать». Расшифрованное сообщение представлена на рисунке 6.4.

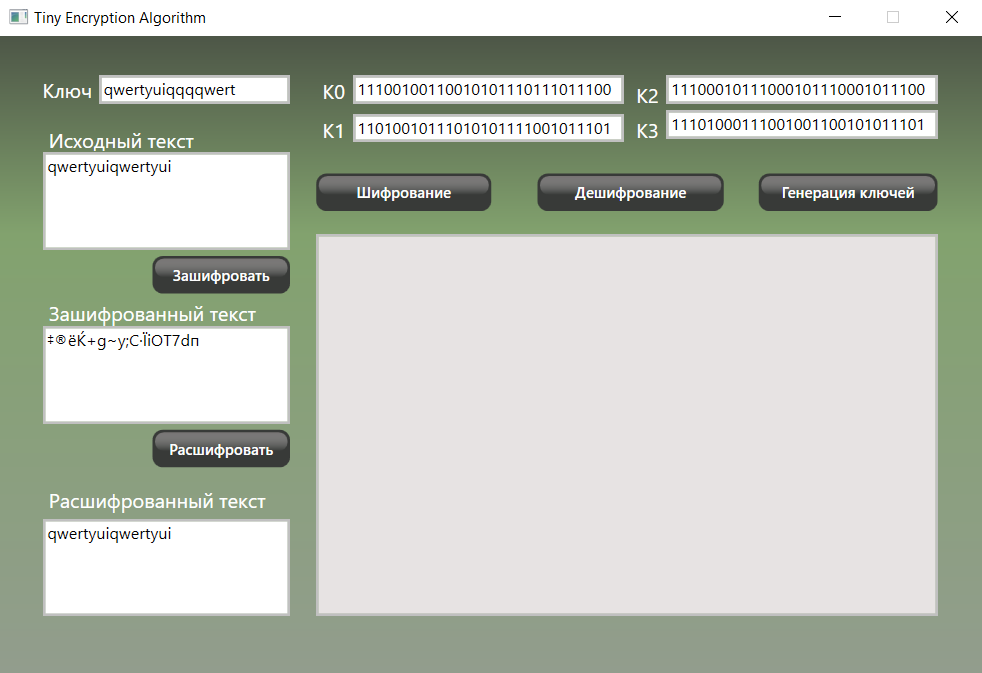


Рисунок 6.4 – Расшифрованное сообщение

После получения зашифрованного и расшифрованного сообщения, можно просмотреть последовательные шаги работы алгоритма.

Нажав на кнопку «Генерация ключей» в окне будет отображаться алгоритм получения подключей из основного ключа. Визуализация генерации ключей показана на рисунке 6.5.

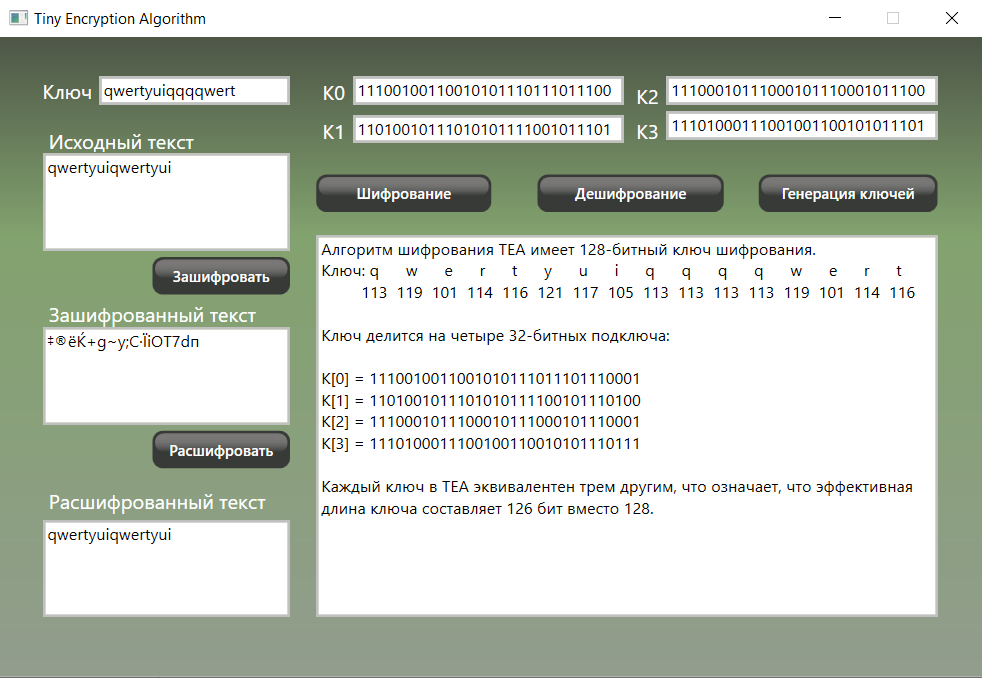


Рисунок 6.5 – Визуализация алгоритма генерации ключей

По кнопке «Шифрование» можно посмотреть последовательные действия алгоритм шифрование. В информационном окне будет отображаться шифрование одного блока данных. Визуализация алгоритма шифрования представлена на рисунке 6.6.

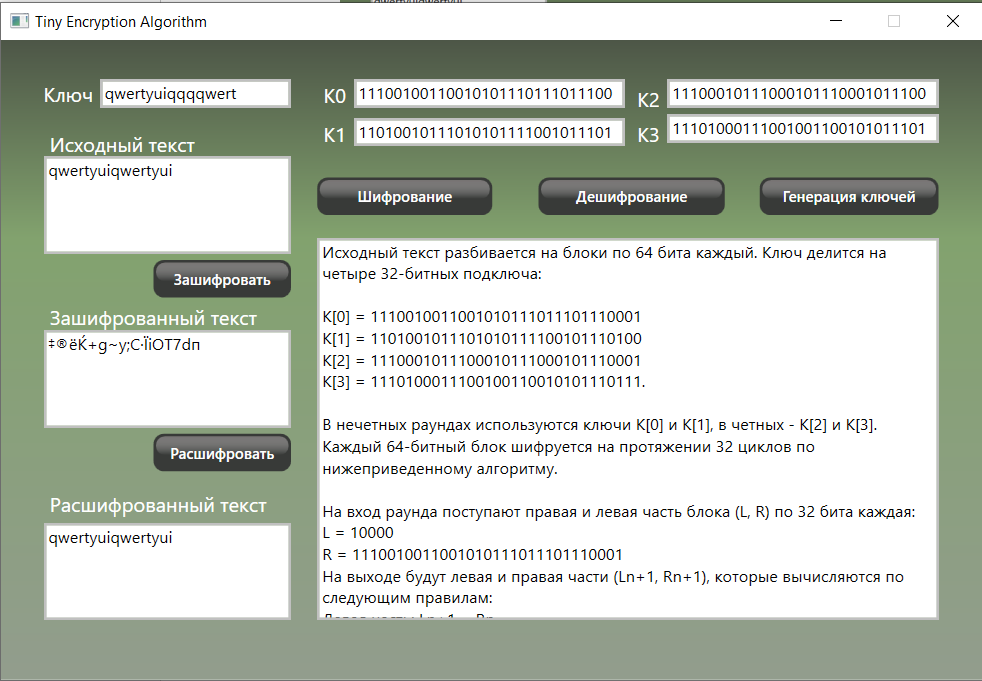


Рисунок 6.6 – Визуализация алгоритма шифрования

Данное руководство составлено для более легкого и понятного ознакомления пользователя с приложением.

**Заключение**

Данный проект был разработан для визуализации шифрования информации с помощью блочного алгоритма TEA. Главной задачей было сделать максимально простое и функциональное приложение. Для этого был использован API-интерфейс WPF и среда разработки Visual Studio 2019.

Перед началом разработки приложения были изучены модификации данного алгоритма и был произведен обзор аналогов программных средств шифрования. Также был проработан ряд исключительных ситуаций с выводом сообщений, в случае некорректной работы пользователя.

Для ознакомления пользователя с приложением было составлено руководство пользователя. Данное руководство описывает необходимую последовательность действий для зашифрования и расшифрования сообщения.

Выбранный алгоритм имеет как преимущества, так и недостатки. Достоинствами шифра являются его простота в реализации, небольшой размер кода и довольно высокая скорость выполнения, а также возможность оптимизации выполнения на стандартных 32-битных процессорах.

Алгоритм имеет отличную устойчивость к линейному криптоанализу и довольно хорошую к дифференциальному криптоанализу. Главным недостатком этого алгоритма шифрования является его уязвимость для атак «на связанных ключах». Из-за простого расписания ключей каждый ключ имеет 3 эквивалентных ключа. Это означает, что эффективная длина ключа составляет всего 126 бит, поэтому данный алгоритм не следует использовать в качестве хеш-функции [5].

Программа получила максимально удобный интерфейс и является понятной даже для человека, который мало знаком с компьютером и почти его не использует в повседневной жизни.

# **Список используемых источников**

1. Брюс Шнайер. Прикладная криптография. – М.: 2016. – 610с.
2. Введение в криптографию/ Под общ. ред. В.В. Ященко. – 3-е изд., доп. – М.: 2000. – 288с.
3. Файловый архив студентов [Электронный ресурс] / Foundation Inc. https://studfile.net/preview/2262041/page:5/ - Дата доступа: 20.04.2020
4. Энциклопедия [Электронный ресурс] / Foundation, Inc. https://ru.wikipedia.org/wiki/TEA Дата доступа: 20.04.2021
5. Чмора А.Л. Современная прикладная криптография. 2-е изд., стер. – М.: Гелиос АРВ, 2004. – 256с.

# **Приложение А**

**Функции класса AlgoTea**

//функция шифрования сообщения

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

{

uint[] key1 = AlgoTea.CreateKey(key);

uint[] v = new uint[2];

byte[] buffer = new byte[AlgoTea.NextMultipleOf8(data.Length + 4)];

byte[] bytes = BitConverter.GetBytes(data.Length);

Array.Copy((Array)bytes, (Array)buffer, bytes.Length);

Array.Copy((Array)data, 0, (Array)buffer, bytes.Length, data.Length);

using (MemoryStream memoryStream = new MemoryStream(buffer))

{

using (BinaryWriter binaryWriter = new BinaryWriter((Stream)memoryStream))

{

for (int startIndex = 0; startIndex < buffer.Length; startIndex += 8)

{

v[0] = BitConverter.ToUInt32(buffer, startIndex);

v[1] = BitConverter.ToUInt32(buffer, startIndex + 4);

AlgoTea.BlockEncrypt(v, key1);

binaryWriter.Write(v[0]);

binaryWriter.Write(v[1]);

}}}

return buffer;

}

//функция расшифрования сообщения

public static byte[] Decrypt(byte[] data1, byte[] key)

{

byte[] numArray1 = AlgoTea.Encrypt(data1, key);

if ((uint)(numArray1.Length % 8) > 0U)

throw new ArgumentException("Длина должна быть кратна 8 байтам");

uint[] key1 = AlgoTea.CreateKey(key);

uint[] v = new uint[2];

byte[] buffer = new byte[numArray1.Length];

Array.Copy((Array)numArray1, (Array)buffer, numArray1.Length);

using (MemoryStream memoryStream = new MemoryStream(buffer))

{ using (BinaryWriter binaryWriter = new BinaryWriter((Stream)memoryStream))

{ for (int startIndex = 0; startIndex < buffer.Length; startIndex += 8)

{

v[0] = BitConverter.ToUInt32(buffer, startIndex);

v[1] = BitConverter.ToUInt32(buffer, startIndex + 4);

AlgoTea.BlockDecrypt(v, key1);

binaryWriter.Write(v[0]);

binaryWriter.Write(v[1]);

} }}

uint uint32 = BitConverter.ToUInt32(buffer, 0);

if ((long)uint32 > (long)(buffer.Length - 4))

throw new ArgumentException("Зашифрованные данные разрушены");

byte[] numArray2 = new byte[(int)uint32];

Array.Copy((Array)buffer, 4L, (Array)numArray2, 0L, (long)uint32);

return numArray2;

}

//функция генерации ключей

public static uint[] CreateKey(byte[] key)

{

byte[] numArray1 = new byte[16];

byte[] numArray2 = key;

return new uint[4]

{ BitConverter.ToUInt32(numArray2, 0),

BitConverter.ToUInt32(numArray2, 4),

BitConverter.ToUInt32(numArray2, 8),

BitConverter.ToUInt32(numArray2, 12)

};}

//функция шифрования блока сообщения

public static void BlockEncrypt(uint[] v, uint[] key)

{ uint num1 = v[0];

uint num2 = v[1];

uint num3 = 0;

uint num4 = key[0];

uint num5 = key[1];

uint num6 = key[2];

uint num7 = key[3];

for (uint index = 0; index < 32U; ++index)

{ num3 += 2654435769U;

num1 += (uint)(((int)num2 << 4) + (int)num4 ^ (int)num2 + (int)num3 ^ (int)(num2 >> 5) + (int)num5);

num2 += (uint)(((int)num1 << 4) + (int)num6 ^ (int)num1 + (int)num3 ^ (int)(num1 >> 5) + (int)num7);

}

v[0] = num1; v[1] = num2; }

//функция расшифрования блока сообщения

private static void BlockDecrypt(uint[] v, uint[] key)

{ uint num1 = v[0];

uint num2 = v[1];

uint num3 = 3337565984;

uint num4 = key[0];

uint num5 = key[1];

uint num6 = key[2];

uint num7 = key[3];

for (uint index = 0; index < 32U; ++index)

{

num2 -= (uint)(((int)num1 << 4) + (int)num6 ^ (int)num1 + (int)num3 ^ (int)(num1 >> 5) + (int)num7);

num1 -= (uint)(((int)num2 << 4) + (int)num4 ^ (int)num2 + (int)num3 ^ (int)(num2 >> 5) + (int)num5);

num3 -= 2654435769U;

}

v[0] = num1;

v[1] = num2;}

# **Приложение Б**

**Функции класса MainWindow**

//функция визуализации алгоритма шифрования

private void Encrypt\_Click(object sender, RoutedEventArgs e)

{

Inf.Text = "";

Inf.Background = Brushes.White;

uint[] key = AlgoTea.CreateKey(Encoding.Default.GetBytes(tbPassword.Text));

byte[] bytes1 = Encoding.Default.GetBytes(tbSource.Text);

uint[] v = new uint[2];

byte[] numArray = new byte[(bytes1.Length + 11) / 8 \* 8];

byte[] bytes2 = BitConverter.GetBytes(bytes1.Length);

Array.Copy((Array)bytes2, (Array)numArray, bytes2.Length);

Array.Copy((Array)bytes1, 0, (Array)numArray, bytes2.Length, bytes1.Length);

Inf.Text = "Исходный текст разбивается на блоки по 64 бита каждый. Ключ делится на четыре 32-битных подключа: \n\nK[0] = " + Convert.ToString((long)key[0], 2) + " \nK[1] = " + Convert.ToString((long)key[1], 2) + " \nK[2] = " + Convert.ToString((long)key[2], 2) + " \nK[3] = " + Convert.ToString((long)key[3], 2) + ". \n\nВ нечетных раундах используются ключи K[0] и K[1], в четных - K[2] и K[3]. Каждый 64-битный блок шифруется на протяжении 32 циклов по нижеприведенному алгоритму.\n\n";

Inf.Text += "На вход раунда поступают правая и левая часть блока (L, R) по 32 бита каждая:";

using (MemoryStream memoryStream = new MemoryStream(numArray))

{

using (BinaryWriter binaryWriter = new BinaryWriter((Stream)memoryStream))

{

v[0] = BitConverter.ToUInt32(numArray, 0);

v[1] = BitConverter.ToUInt32(numArray, 4);

Inf.Text += "\nL = " + Convert.ToString(v[0], 2) + "\nR = " + Convert.ToString(v[1], 2) + " ";

Inf.Text += "\nНа выходе будут левая и правая части (Ln+1, Rn+1), которые вычисляются по следующим правилам:\nЛевая часть: Ln+1 = Rn\nПравая часть зависит от четности раунда:\n 1)Если нечётные раунды: \n\tRn+1 = Ln + (( [ Rn << 4 ] + K[0] ) xor ( Rn + i \* δ ) xor ( [ Rn >> 5 ] + K[1] ))\n 2)Если чётные раунды: \n\tRn+1 = Ln + (( [ Rn << 4 ] + K[2] ) xor ( Rn + i \* δ ) xor ( [ Rn >> 5 ] + K[3] )).\nЗдесь:\n\t+ - операция сложения чисел по модулю 2^32\n\txor - побитовое исключающее «ИЛИ»\n\tX << Y и X >> Y – операции побитового сдвига числа X на Y бит влево и вправо соответственно\n\tКонстанта δ была выведена из Золотого сечения δ = 2654435769\nТаким образом выполняется 32 раунда\n";

Inf.Text += "\nНа выходе получаем первый зашифрованный блок сообщения:\n";

AlgoTea.BlockEncrypt(v, key);

binaryWriter.Write(v[0]);

binaryWriter.Write(v[1]);

Inf.Text += "\t1) " + Encoding.Default.GetString(numArray).Trim().Substring(0, 8) + "\nОставшиеся зашифрованные блоки: \n";

for(int startIndex1 = 8, startIndex2 = 8, num = 1; startIndex1 < numArray.Length; startIndex1 += 8, startIndex2 += 8, num++)

{

v[0] = BitConverter.ToUInt32(numArray, startIndex1);

v[1] = BitConverter.ToUInt32(numArray, startIndex1 + 4);

AlgoTea.BlockEncrypt(v, key);

binaryWriter.Write(v[0]);

binaryWriter.Write(v[1]);

Inf.Text += $"\t{num}) {Encoding.Default.GetString(numArray).Trim().Substring(startIndex2, 8)}\n";

}

}

Inf.Text += "\nЗашифрованное сообщение целиком: " + Encoding.Default.GetString(numArray);

}

}

//функция визуализации генерации ключей

private void Generate\_keys\_Click(object sender, RoutedEventArgs e)

{

try

{

Inf.Text = "";

Inf.Background = Brushes.White;

uint[] key = AlgoTea.CreateKey(Encoding.Default.GetBytes(tbPassword.Text));

char[] charArray = tbPassword.Text.ToCharArray();

Inf.Text = "Алгоритм шифрования TEA имеет 128-битный ключ шифрования. \nКлюч: ";

for (int index = 0; index < charArray.Length; ++index)

{

Inf.Text += charArray[index].ToString() + " ";

}

Inf.Text += "\n ";

for (int index = 0; index < charArray.Length; ++index)

{

Inf.Text += ((int)charArray[index]).ToString() + " ";

}

Inf.Text += "\n\nКлюч делится на четыре 32-битных подключа: \n\nK[0] = " + Convert.ToString(key[0], 2) + " \nK[1] = " + Convert.ToString(key[1], 2) + " \nK[2] = " + Convert.ToString(key[2], 2) + " \nK[3] = " + Convert.ToString(key[3], 2);

Inf.Text += "\n\nКаждый ключ в TEA эквивалентен трем другим, что означает, что эффективная длина ключа составляет 126 бит вместо 128.";

}

catch

{ }