МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

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 Микляева А.С.

(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

**Оглавление**

[**Введение** 3](#_Toc70551118)

[**1. Аналитический обзор литературы** 4](#_Toc70551119)

[**1.1 Описание симметричного алгоритма шифрования TEA** 4](#_Toc70551120)

[**1.1.1 Генерация ключа** 4](#_Toc70551121)

[**1.1.2 Шифрование** 4](#_Toc70551122)

[**1.1.3 Расшифрование** 5](#_Toc70551123)

[1.1.4 5](#_Toc70551124)

[**1.2 Обзор аналогов** 5](#_Toc70551125)

[**2. Разработка архитектуры проекта** 7](#_Toc70551126)

[**3. Тестирование** 9](#_Toc70551127)

[**4. Руководство пользователя** 11](#_Toc70551128)

[**Заключение** 14](#_Toc70551129)

[**Список используемых источников** 15](#_Toc70551130)

[**Приложение А** 16](#_Toc70551131)

[**Приложение Б** 18](#_Toc70551132)

# **Введение**

1976 год считается годом рождения несимметричной криптографии. В этом году американскими математиками Вайтфилдом Диффи, Мартином Хеллманом и Ральфом Меркле была представлена идеология криптосистемы с открытым ключом

Кардинальное отличие криптосистемы с открытым ключом (по другой терминологии, несимметричной системы) от симметричной системы состоит в том, что в криптосистемах с открытым ключом процедура зашифровывания становится общедоступной. Это, однако, не означает как в традиционных системах шифрования, что общедоступной является и процедура расшифровывания. Понятие ключа разбивается на две части (включает теперь два понятия): ключ открытый, и ключ секретный. Общедоступный открытый ключ используется для зашифровывания, но расшифровывание может осуществить только тот, кто владеет секретным ключом.

Именно как раз в допущении того, что нахождение ключа расшифровывания по известному ключу зашифровывания может быть сложно-вычислимой задачей, и заключается идея, которая определила дальнейшее направление развития криптографии.

Но наравне с криптографией шло развитие и криптоанализа – другого противоположного раздела криптологии, предметом которого является разработка методов взлома новых криптографических алгоритмов с целью выявления их надежности. Результатом возникновения каждого нового метода криптоанализа является пересмотр оценок безопасности шифров, что в свою очередь, влечет необходимость создания более стойких шифров.

Целью курсовой работы является анализ надежности алгоритмов ассиметричных методов шифрования.

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.

**1. Аналитический обзор литературы**

## **1.1 Описание криптосистем с открытым ключом**

Криптографическая система с открытым ключом – система шифрования, при которой открытый ключ передаётся по открытому каналу связи, и используется для шифрования сообщения, а для расшифрования сообщения используется секретный ключ.

Идея криптографии с открытым ключом очень тесно связана с идеей односторонних функций, то есть таких функций f(x), что по известному x довольно просто найти значение f(x), тогда как определение x из f(x) сложно в смысле теории.

Но сама односторонняя функция бесполезна в применении: ею можно зашифровать сообщение, но расшифровать нельзя. Поэтому криптография с открытым ключом использует односторонние функции с лазейкой. Лазейка — это некий секрет, который помогает расшифровать. То есть существует такой y, что зная f(x), можно вычислить x. К примеру, если разобрать часы на множество составных частей, то очень сложно собрать вновь работающие часы. Но если есть инструкция по сборке (лазейка), то можно легко решить эту проблему.

Алгоритмы шифрования с открытым ключом разрабатывались для того, чтобы решить две наиболее трудные задачи, возникшие при использовании симметричного шифрования.

Первой задачей является распределение ключа. При симметричном шифровании требуется, чтобы обе стороны уже имели общий ключ, который каким-то образом должен быть им заранее передан.

Второй задачей является необходимость создания таких механизмов, при использовании которых невозможно было бы подменить кого-либо из участников, т.е. нужна цифровая подпись. При использовании коммуникаций для решения широкого круга задач, например в коммерческих и частных целях, электронные сообщения и документы должны иметь эквивалент подписи, содержащейся в бумажных документах. Необходимо создать метод, при использовании которого все участники будут убеждены, что электронное сообщение было послано конкретным участником. Это более сильное требование, чем аутентификация.

Рассмотрим требования, сформулированные Диффи и Хеллманом, которым должен удовлетворять алгоритм шифрования с открытым ключом.

* Вычислительно легко создавать пару (открытый ключ, закрытый ключ);
* Вычислительно легко, имея открытый ключ и незашифрованное сообщение , создать соответствующее зашифрованное сообщение;
* Вычислительно легко дешифровать сообщение, используя закрытый ключ;
* Вычислительно невозможно, зная открытый ключ, определить закрытый ключ;
* Вычислительно невозможно, зная открытый ключ и зашифрованное сообщение, восстановить исходное сообщение;
* Шифрующие и дешифрующие функции могут применяться в любом порядке (это требование не выполняется для всех алгоритмов с открытым ключом)

Основными способами использования алгоритмов с открытым ключом являются шифрование/дешифрование, создание и проверка подписи.

### **1.1.1 Генерация ключа**

Ключ К длиной 128 бит делится на четыре 32-битных подключа K0, K1, K2 и K3. В нечетных раундах используются подключи K0 и K1, в четных – 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**

Шифрование с открытым ключом состоит из следующих шагов рисунок 1.1.2.1.

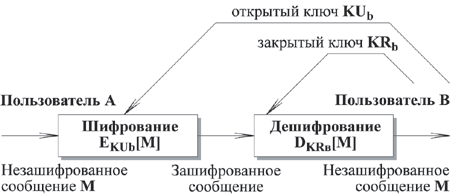


Рисунок 1.1.2.1 – Схема шифрования с открытым ключом

Пользователь В создает пару ключей KUb и KRb, используемых для шифрования и дешифрования передаваемых сообщений;

Пользователь В делает доступным некоторым надежным способом свой ключ шифрования, т.е. открытый ключ KUb. Составляющий пару закрытый ключ KRb держится в секрете;

Если А хочет послать сообщение В, он шифрует сообщение, используя открытый ключ В KUb;

Когда В получает сообщение, он дешифрует его, используя свой закрытый ключ KRb. Никто другой не сможет дешифровать сообщение, так как этот закрытый ключ знает только В.

Если пользователь (конечная система) надежно хранит свой закрытый ключ, никто не сможет подсмотреть передаваемые сообщения.

### **1.1.3 Расшифрование**

Создание и проверка подписи состоит из следующих шагов (рис. 2):

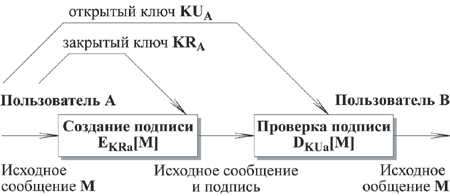


Рисунок 2. Создание и проверка подписи

1. Пользователь А создает пару ключей KRA и KUA, используемых для создания и проверки подписи передаваемых сообщений;
2. Пользователь А делает доступным некоторым надежным способом свой ключ проверки, т.е. открытый ключ KUA. Составляющий пару закрытый ключ KRA держится в секрете;
3. Если А хочет послать подписанное сообщение В, он создает подпись EKRa[M] для этого сообщения, используя свой закрытый ключ KRA;
4. Когда В получает подписанное сообщение, он проверяет подпись DKUa[M], используя открытый ключ А KUA. Никто другой не может подписать сообщение, так как этот закрытый ключ знает только А.

До тех пор, пока пользователь или прикладная система надежно хранит свой закрытый ключ, их подписи достоверны. Кроме того, невозможно изменить сообщение, не имея доступа к закрытому ключу А; тем самым обеспечивается аутентификация и целостность данных.

В этой схеме все сообщение подписывается, причем для подтверждения целостности сообщения требуется много памяти. Каждое сообщение должно храниться в незашифрованном виде для использования в практических целях. Кроме того, копия сообщения также должна храниться в зашифрованном виде, чтобы можно было проверить в случае необходимости подпись. Более эффективным способом является шифрование небольшого блока битов, который является функцией от сообщения. Такой блок, называемый аутентификатором, должен обладать свойством невозможности изменения сообщения без изменения аутентификатора. Если аутентификатор зашифрован закрытым ключом отправителя, он является цифровой подписью, с помощью которой можно проверить исходное сообщение. Далее эта технология будет рассматриваться в деталях.

Важно подчеркнуть, что описанный процесс создания подписи не обеспечивает конфиденциальность. Это означает, что сообщение, посланное таким способом, невозможно изменить, но можно подсмотреть. Это очевидно в том случае, если подпись основана на аутентификаторе, так как само сообщение передается в явном виде. Но даже если осуществляется шифрование всего сообщения, конфиденциальность не обеспечивается, так как любой может расшифровать сообщение, используя открытый ключ отправителя.

## **1.2 Обзор аналогов**

Первой системой с открытым ключом стал **метод экспоненциального ключевого обмена Диффи - Хеллмана,** разработанный в 1976 году. Метод предназначен для передачи секретного ключа симметричного шифрования. В обмене задействованы два участника А и Б (рис.3). Сначала они выбирают большие простые числа *n* и *g*<*n* (эти числа секретными не являются). Затем участник A выбирает большое целое число *х,*  вычисляет *Х*=*gx* mod *n* и передает *Х* участнику Б. Б в свою очередь выбирает большое целое число *y*, вычисляет *Y*=*gy* mod *n* и передает *Y* участнику А. Б вычисляет *K’*=*Xy* mod *n,*  А вычисляет *K’’*=*Yx* mod *n*. Легко заметить, что *K’*=*K’*’=*gxy* mod *n,* и это значение оба участника могут использовать в качестве ключа симметричного шифрования.

Криптостойкость этого метода определяется трудоемкостью вычисления дискретного логарифма в конечном поле. Действительно, злоумышленник может узнать такие параметры алгоритма, как *n, g, X, Y,* но вычислить по ним значения *x* или *y* – задача, требующая очень больших вычислительных мощностей и времени (последнее утверждение верно при использовании сверхбольших чисел, размером более 768 бит). Метод легко можно обобщить на случай ключевого обмена для большего количества участников.

Пользователь А

Пользователь Б

Открытый канал

1. Выбрать случайное число *x*

2. Вычислить

*Х*=*g x* mod *n*

3. Передать X пользователю Б

4. Получить Y.

Вычислить

*K’’*=*Y x* mod *n*

1. Выбрать случайное число *y*

2. Вычислить

*Y*=*g y* mod *n*

3. Передать Y пользователю А

4. Получить X.

Вычислить

*K’*=*X y* mod *n*

X

Y

*g, n* – открытые

параметры

Ключ *K’*=*K’*’=*gxy* mod *n* используется в симметричном алгоритме

**Получатель**

**Отправитель**

**Генерация ключей**

Выбрать простое *p* и 2 случайных числа *g* – первообразный корень по модулю *p* и *x* в диапазоне (1,*p*), такое, что (*x,p*)=1

Вычислить *y=gx* mod p

**Открытый ключ** (*p,g,y*)

**Закрытый ключ** *x*

**Сообщение** *m*

**Шифрование**

Выбрать случайное *k*:

(*k,p-1*)=1

Вычислить

*a*=*gk* mod *p*

*b*=*ykm*  mod *p*

**Шифротекст**

(*a,b*)

**Дешифрование**

*m*=(*b / ax*) mod *p*

**Алгоритм Эль-Гамаля** – асимметричный алгоритм шифрования, основанный на проблеме дискретного логарифмирования, разработан в 1985 г. Последовательность действий при генерации ключей, шифровании и дешифрации представлена на рис. 4.

Необходимо пояснить процедуру дешифрования. Так как *ax≡gkx* mod *p,* то имеем:



Таким образом, кодируемое сообщение М разбивается на части, каждая из которых *m* интерпретируется как число в диапазоне [0 .. *p*-1], и выполняется операция шифрования согласно схеме на рис.4. На практике при использовании данного алгоритма рекомендуется выбирать ключи размером 768, 1024 и 1536 бит.

**Алгоритм RSA** был разработан в 1978 году. Алгоритм назван в честь авторов (Rivest, Shamir, Adleman). В основу криптостойкости RSA положена задача факторизации (разложения на множители) больших целых чисел. Процедуры генерации ключей, шифрования и дешифрования для этого алгоритма представлены на рис. 5.

**Получатель**

**Отправитель**

**Генерация ключей**

Выбрать простые *p* и *q*

Вычислить *n=p\*q*

Вычислить функцию Эйлера

*f(p,q)=(p-1)(q-1)*

Выбрать случайное d взаимно простое с *f(p,q)*

Подобрать такое e, чтобы:

*e\*d ≡ 1 mod f(p,q)*

**Открытый ключ** (*e,n*)

**Закрытый ключ** (*d,n)*

**Сообщение** *m*

**Шифрование**

*c*=*me* mod *n*

**Шифротекст**

*c*

**Дешифрование**

*m*=(*cd*) mod *n*

**Рис.5. Схема шифрования и дешифрации алгоритма RSA**

На этапе генерации ключей формируется пара ключей: закрытый *d* и открытый *e*. Шифрование данных должно начинаться с его разбиения на блоки *L* размером *k*=[log2 (*n*)] бит каждый, чтобы блок *L* можно было рассматривать как целое число в диапазоне [0.. *n-*1].

Основная задача криптоаналитика при взломе этого шифра – узнать закрытый ключ *d*. Для этого он должен выполнить те же действия, что и получатель при генерации ключа – решить в целых числах уравнение *ed* + *y* (*p*-1)(*q*-1) =1 относительно *d* и *y.* Однако, если получателю известны входящие в уравнение параметры *p* и *q,* то криптоаналитик знает только число *n –* произведение *p* и *q*. Следовательно, ему необходимо произвести факторизацию числа *n,* то есть разложить его на множители. Для решения задачи факторизации к настоящему времени разработано множество алгоритмов, наиболее известными из них являются метод квадратичного решета и метод эллиптических кривых. Но для чисел большой размерности (около 1024 бит и более) это очень трудоемкая задача.

**DSA (Digital Signature Algorithm)** - алгоритм цифровой подписи, принятый в 1994 году в качестве стандарта США на ЭЦП и действовавший до 2001 г., представляет собой один из вариантов схемы Эль-Гамаля.

Национальный институт стандартов и технологии NIST, вместе с АНБ, разработал алгоритм **SHA** (Secure Hash Algorithm – алгоритм стойкого хеширования), требуемый для обеспечения стойкости алгоритма цифровой подписи DSA. По предположениям Microsoft, алгоритм **SHA-2**, предусматривающий генерацию хеш-значений длиной 256, 384 и 512 разрядов, еще на протяжение нескольких лет будет оставаться стойким, но в конечном счете будет заменен.

Ряд успешных атак на системы, основанные на сложности дискретного логарифмирования в конечных полях, привел к тому, что стандарты ЭЦП России и США в 2001 году были обновлены: переведены на **эллиптические кривые**. Алгоритмы ЭЦП при этом не изменились, однако вместо элементов конечного поля они оперируют эллиптическими числами, т.е. решениями уравнения эллиптических кривых над указанными конечными полями, а роль

операции возведения в степень в конечном поле выполняет операция взятиякратной точки эллиптической кривой.

Специальный выбор типа эллиптической кривой позволяет не только во много раз усложнить задачу взлома схемы ЭЦП, но и уменьшить рабочий размер блоков данных. Старый российский стандарт ЭЦП оперирует 1024-битовыми блоками, а новый, основанный на эллиптических кривых, — 256-битовыми, и при этом обладает большей стойкостью.

**EC-DSA (Elliptic Curve Digital Signature Algorithm)** – это классический алгоритм DSA, «переведенный» на эллиптические кривые. Стандарт FIPS 186-2 предусматривает 256- и 284-битный рабочий размер блоков данных, но Microsoft также поддерживает 521-разрядные ключи.

Как известно, далеко не все присутствующие на рынке криптографические средства обеспечивают обещанный уровень защиты. Системы и средства защиты информации (СЗИ) характеризуются тем, что для них не существует простых и однозначных тестов, позволяющих убедиться в надежной защите информации. Например, чтобы проверить работоспособность системы связи, достаточно провести ее испытания. Тем не менее, успешное завершение этих испытаний не дает возможность сделать вывод, что встроенная подсистема защиты информации тоже является работоспособной. Часто задача определения эффективности СЗИ при использовании криптографических методов защиты оказывается даже более трудоемкой, чем разработка СЗИ, т.к. требует наличия специальных знаний и более высокой квалификации, чем задача разработки. Как правило, анализ нового криптографического алгоритма является новой научной, а не инженерной задачей.

Повышение производительности вычислительной техники и появление новых видов атак на шифры ведет к понижению стойкости известных криптографических алгоритмов. Для уменьшения возможного ущерба, вызванного несвоевременной заменой потерявшего свою стойкость криптоалгоритма, необходима периодическая перепроверка стойкости криптоалгоритмов, которая включает в себя как разработку новых методов криптоанализа, так и повышение эффективности существующих методов.

То обстоятельство, что любую задачу отыскания способа раскрытия некоторой конкретной криптосистемы можно переформулировать как привлекательную математическую задачу, при решении которой удается использовать многие методы той же теории сложности, теории чисел и алгебры, привело к раскрытию многих криптосистем.

Практически все используемые алгоритмы асимметричной криптографии основаны задачах факторизации и дискретного логарифмирования в различных алгебраических структурах.

# **2. Разработка архитектуры проекта**

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.

Рассмотрим решение проекта «TEA», имеющим структуру, представленную на рисунке 2.1.

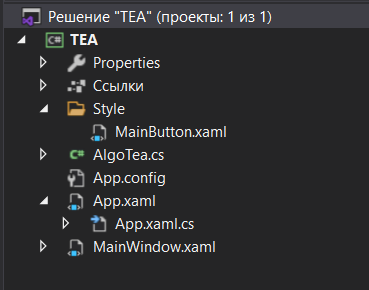


Рисунок 2.1 – Структура проекта

В директории Style находится файл MainBtton.xaml со стилями для кнопок приложения. Помимо этого в основе структуры содержатся файлы App.xaml и Mainwindow.xaml, которые отвечаю за настройку и отображение непосредственно самого окна приложения.

В файле AlgoTea.cs реализованы функции алгоритмов шифрования, расшифрования и генерации ключа. Функции представлены на рисунке 2.2. Программная реализация всех функций представлена в приложении А.

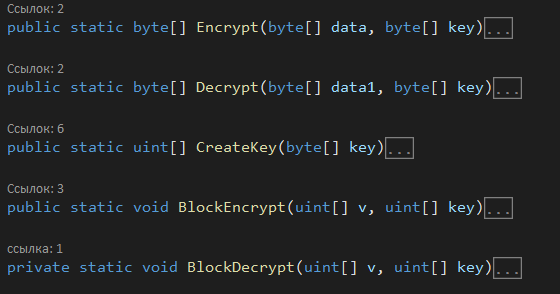


Рисунок 2.2 – Функции алгоритма

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. В четных раундах будут использоваться ключи K0 и K1, в нечетных - ключи K2 и K3.

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.

# **3. 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 3.1).

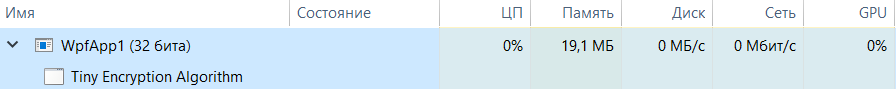


Рисунок 3.1 – Потребляемые ресурсы компьютера

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 3.2). Кнопки станут активны после того как окна «Ключ» и «Исходный текст» будут заполнены.

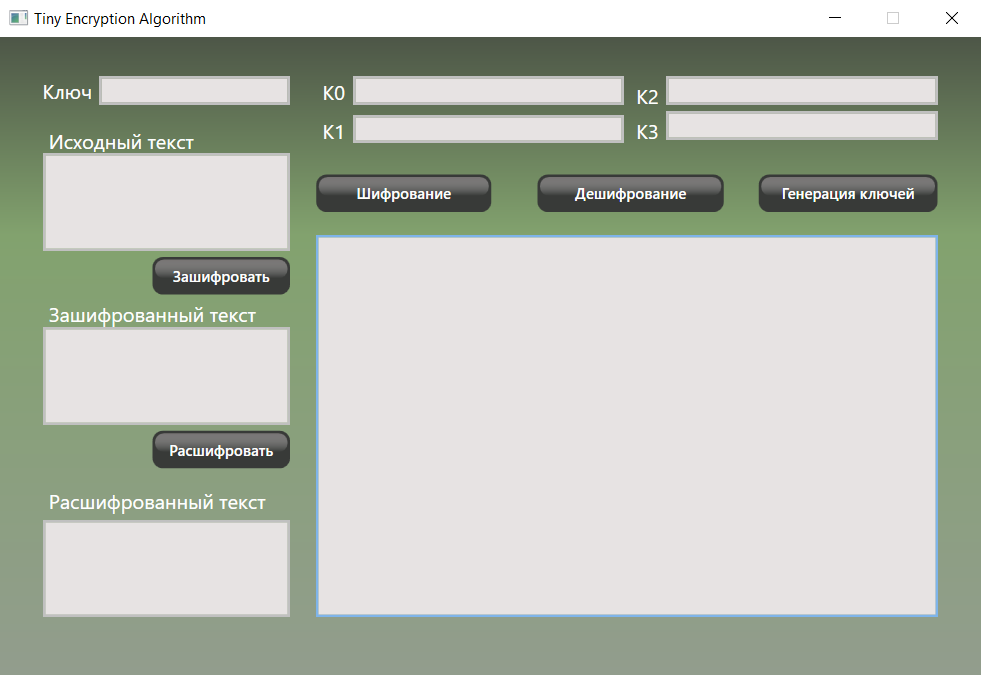


Рисунок 3.2 – Неактивные окна и кнопки

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 3.3. 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.

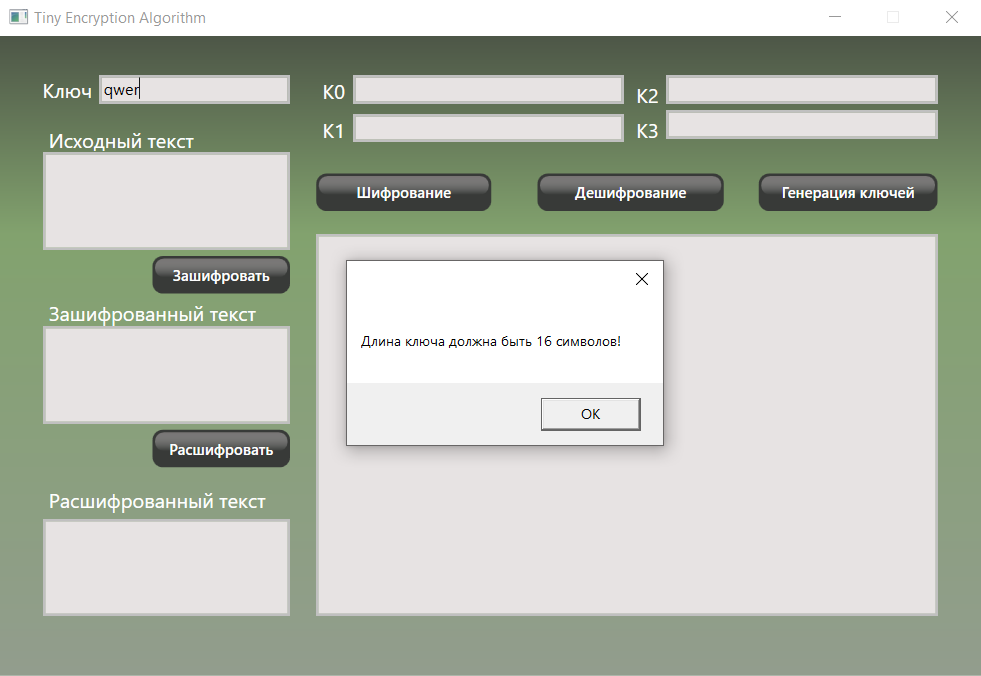


Рисунок 3.3 – Неверная длина ключа

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 3.4).

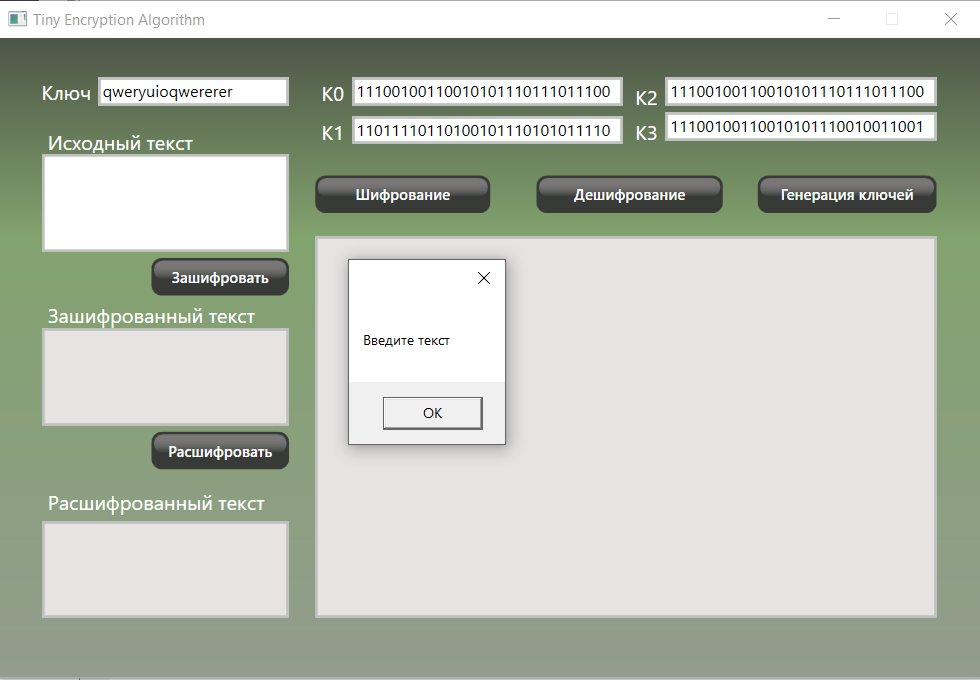


Рисунок 3.4 – Шифрование пустого сообщения

# **4. Руководство пользователя**

Чтобы запустить приложение нужно запустить exe-файл – Tiny Encryption Algorithm.exe. При запуске открывается окно приложения (рисунок 4.1).



Рисунок 4.1 – Окно приложения

Окно содержит окна для ввода и вывода данных и кнопки для совершения операций шифрования и расшифрования.

В данном окне можно:

* ввести данные для шифрования и расшифрования сообщения;
* посмотреть как генерируется ключ для шифрования;
* посмотреть как поэтапно шифруется сообщение;
* просмотреть как поэтапно расшифровывается сообщение.

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 4.2).

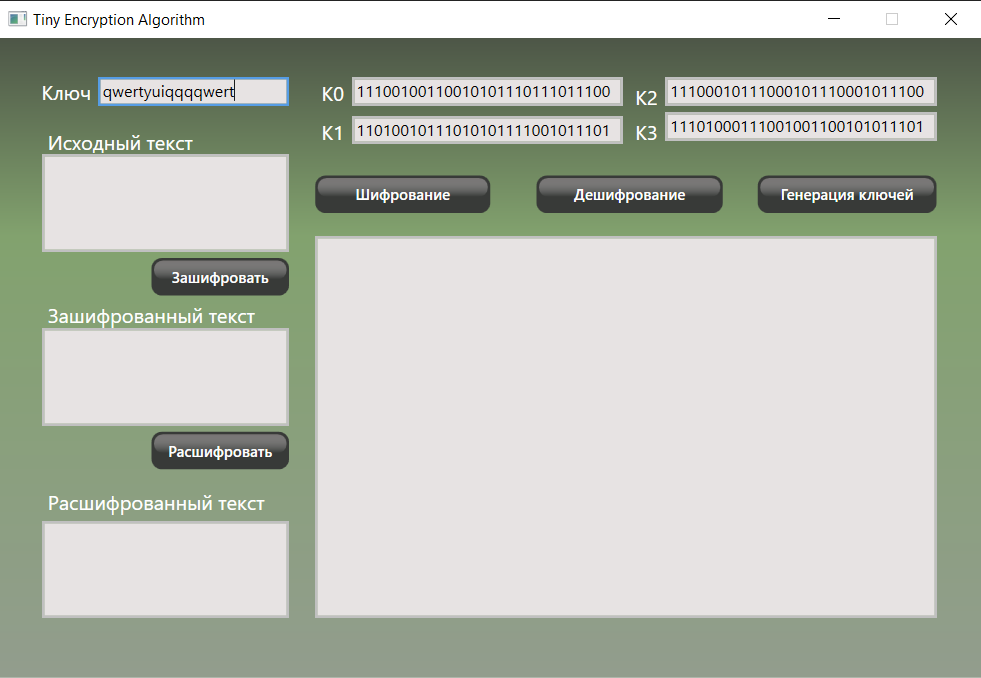


Рисунок 4.2 – Генерация подключей

В окно «Исходный текст» нужно ввести сообщение, а затем нажать кнопку «Зашифровать». В окно «Зашифрованный текст» будет выведено зашифрованное сообщение (рисунок 4.3).

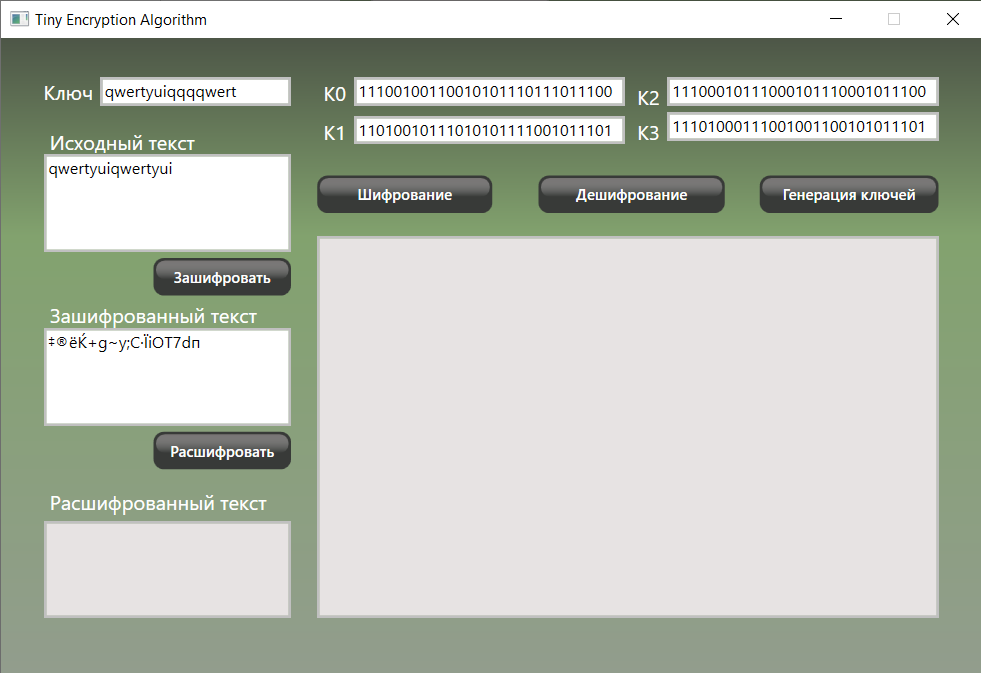


Рисунок 4.3 – Зашифрованное сообщение

После того как получили зашифрованное сообщение, расшифруем его, нажатием на кнопку «Расшифровать». Расшифрованное сообщение представлена на рисунке 4.4.

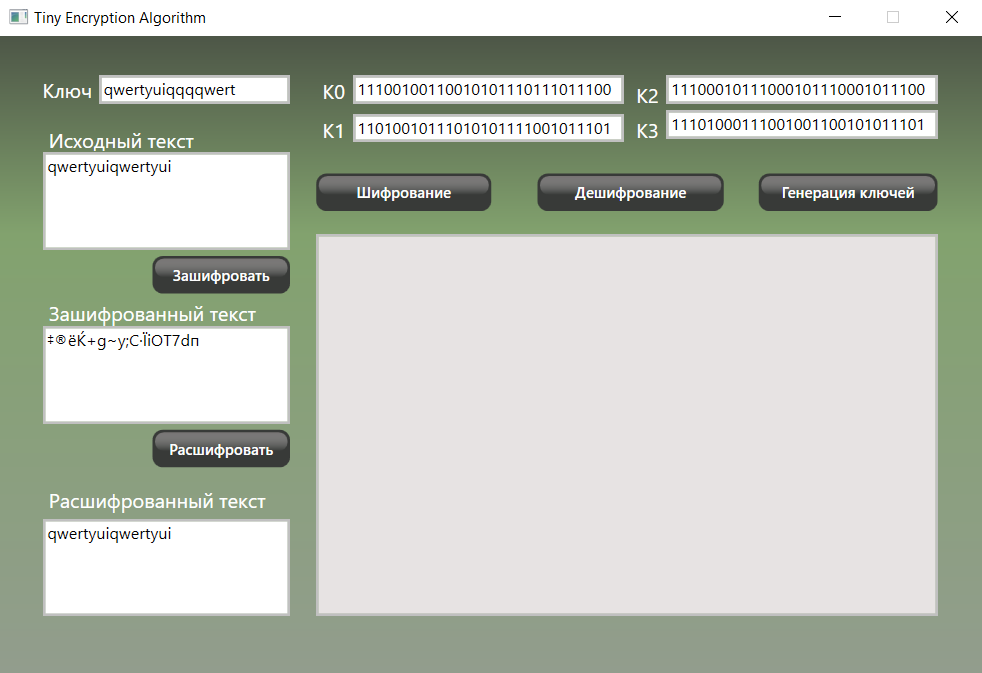


Рисунок 4.4 – Расшифрованное сообщение

После получения зашифрованного и расшифрованного сообщения, можно просмотреть последовательные шаги работы алгоритма. Программная реализация функций визуализации представлена в приложении Б.

Нажав на кнопку «Генерация ключей» в окне будет отображаться алгоритм получения подключей из основного ключа. Визуализация генерации ключей показана на рисунке 4.5.

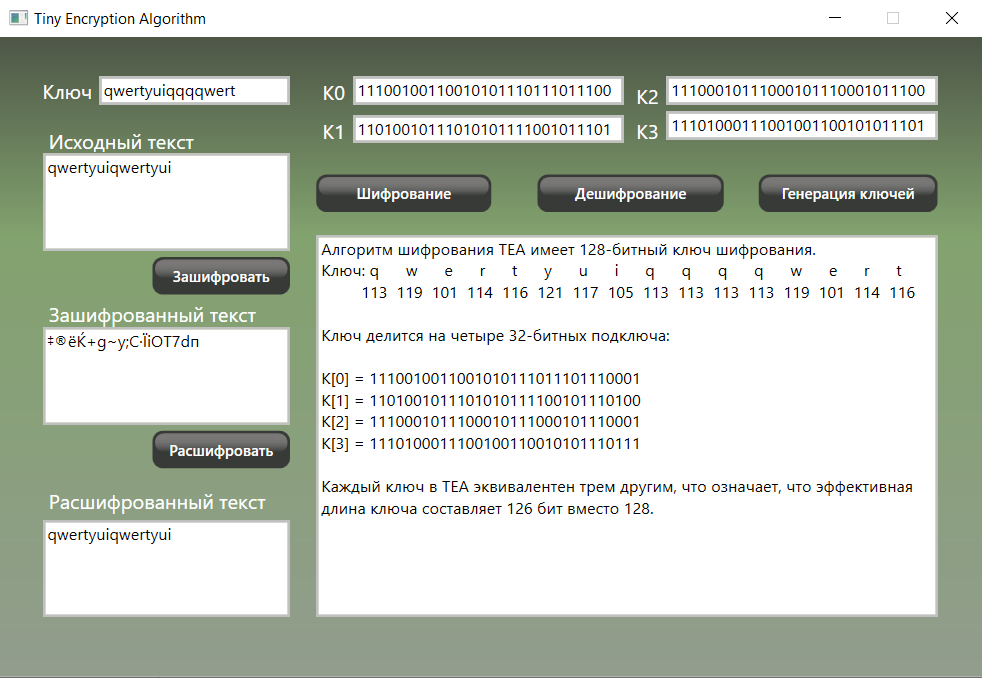


Рисунок 4.5 – Визуализация алгоритма генерации ключей

По кнопке «Шифрование» можно посмотреть последовательные действия алгоритм шифрование. В информационном окне будет отображаться шифрование одного блока данных. Визуализация алгоритма шифрования представлена на рисунке 4.6.

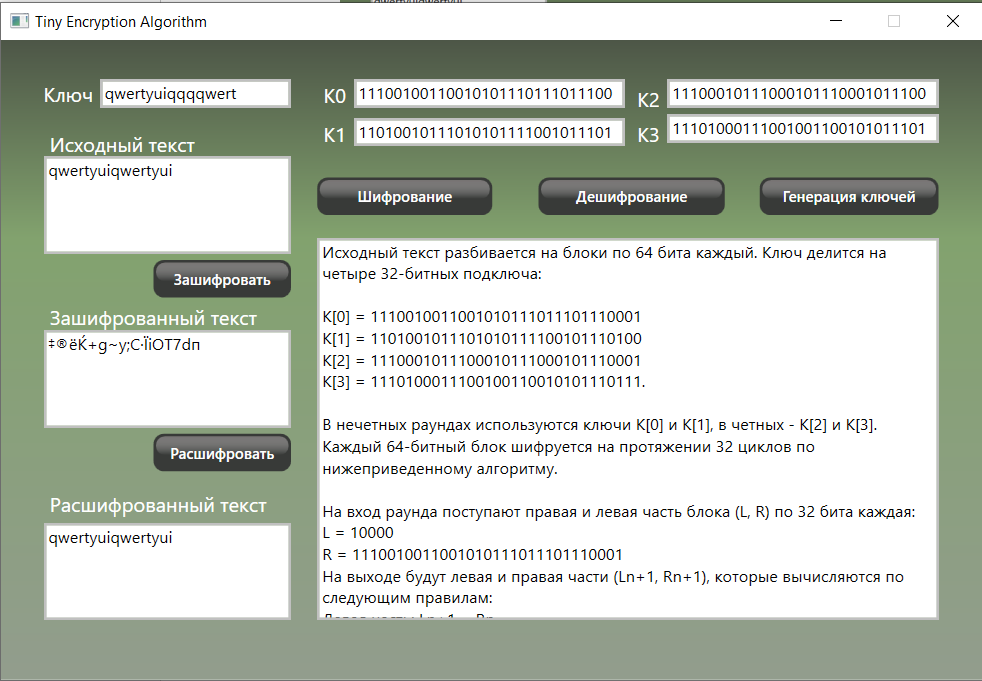


Рисунок 4.6 – Визуализация алгоритма шифрования

# **Заключение**

Данный проект был разработан для 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. Также был проработан ряд исключительных ситуаций с выводом сообщений, в случае некорректной работы пользователя.

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**

**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;

}

**Функции 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**

**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

{ }