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](#_Toc71476554)

[**1.1 Описание криптосистем с открытым ключом** 4](#_Toc71476555)

[**1.1.1 Генерация ключа** 5](#_Toc71476556)

[**1.1.2 Шифрование** 5](#_Toc71476557)

[**1.1.3 Расшифрование** 5](#_Toc71476558)

[**2. Практические основы криптосистем с открытым ключом** 7](#_Toc71476559)

[**2.1 RSA** 7](#_Toc71476560)

[**2.2 El-Gamal** 8](#_Toc71476561)

[**2.3 Diffie-Helman** 8](#_Toc71476562)

[**2.3 Blum-Goldwasser** 9](#_Toc71476563)

[**3. Сравнительный анализ криптосистем с открытым ключом** 12](#_Toc71476564)

[**4. Проектирование программного средства** 12](#_Toc71476565)

[**5.Тестирование программного средства** 12](#_Toc71476566)

[**6.Руководство пользователя** 12](#_Toc71476567)

[**Вывод** 12](#_Toc71476568)

[**4. Руководство пользователя** 15](#_Toc71476569)

[**Заключение** 19](#_Toc71476570)

[**Список используемых источников** 20](#_Toc71476571)

[**Приложение А** 21](#_Toc71476572)

[**Приложение Б** 24](#_Toc71476573)

# **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.

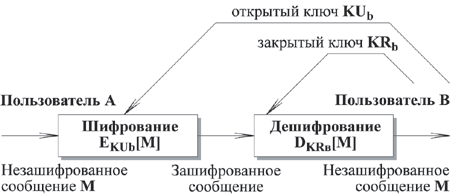


Рисунок 1.1.2.1 – Схема шифрования с открытым ключом

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 представлено на рисунке 1.1.3.1:

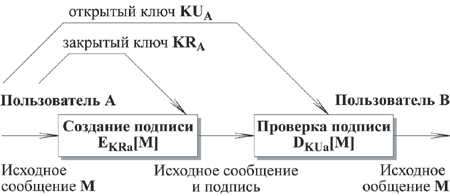


Рисунок 1.1.3.1. Создание и проверка подписи

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.

# **2. Практические основы криптосистем с открытым ключом**

**2.1 RSA**

**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 рисунке 2.1.1.

**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*

Рисунок 2.1.1. 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.

**2.2 El-Gamal**

**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.

**2.3 Diffie-Helman**

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 рисунок 2.3.1. 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 .

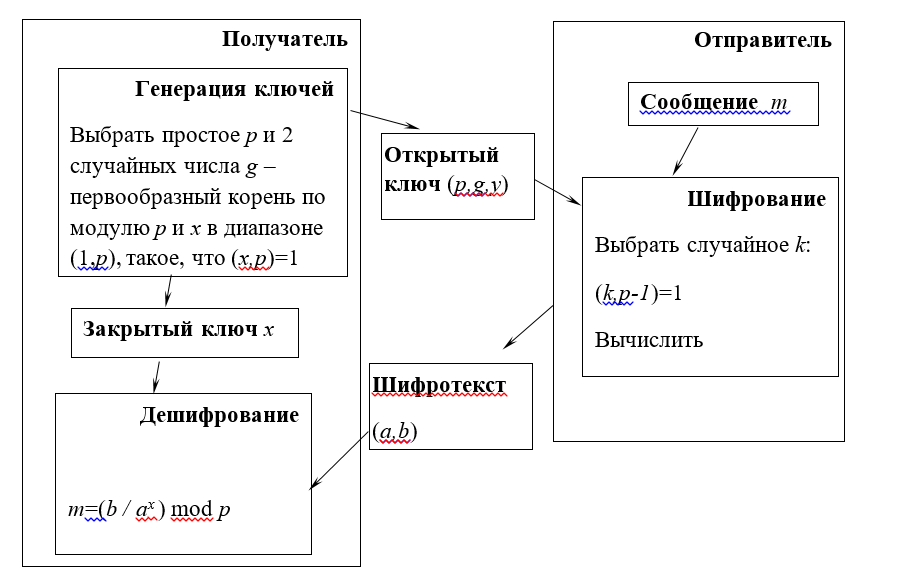


Рисунок 2.3.1 Принцип работы алгоритма

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.

**2.4 Blum-Goldwasser**

Система Блюма — Гольдвассер основана на применении криптографически сильного BBS-генератора, который был рассмотрен в параграфе 2.13. Напомним, что параметрами генератора являются два случайных больших простых числа*р* и *q,* таких, что *p = q =* 3 (mod 4). Числа *pwq* являются личным ключом системы и держатся в секрете, а число Блюма *п = pq* является открытым ключом и может публиковаться свободно.

Отправитель сообщения *М* представляет его в виде битовой последовательности длины *т*, выбирает случайное число s, взаимно простое с *п,* 1 < *s < п,*и генерирует псевдослучайную битовую последовательность *ps = b0b{b2...b*m\_, длины *т,* пользуясь формулами рисунок 2.4.1

https://studme.org/htm/img/15/3044/325.png

Рисунок 2.4.1 Формулы генерации псевдослучайной последовательности

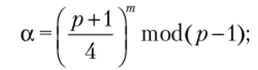
Полученная псевдослучайная битовая последовательность накладывается на сообщение (с помощью операции побитового XOR) и отправляется получателю вместе с подсказкой — значением следующего члена ряда.\*:,, т.е. *хт.* Таким образом, в качестве криптограммы выступает пара *(хт, М* © *ps*), причем *хт* не участвовал в формировании последовательности *ps.*

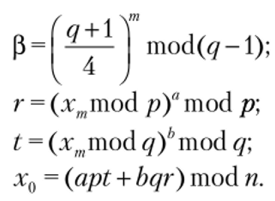
Получатель должен восстановить последовательность *ps,* а затем наложить ее на *М* © *ps* *(М* © *ps* © *ps* = *М).*

Стойкость криптосистемы базируется на непредсказуемости влево BBS генератора, определяемой однонаправленностью функции *f(pc)* = я2 mod *п, п —* число Блюма, и трудности факторизации числа *п.*

В самом деле, зная значение *хт* и открытый ключ *п*, противник может легко продолжить последовательность *ps,* но для того, чтобы узнать ее начало, он должен уметь эффективно вычислять значения *х-\_х = yfx^modn.*Сложность таких вычислений эквивалентна сложности факторизации числа *п.* В предположении, что факторизация числа *п* является трудной задачей, оказывается практически невозможно не только определить квадратный корень по модулю, но даже с вероятностью, большей 0,5, получить информацию о его первом бите.

В то же время законный получатель сообщения, зная значения/? и *q,* может легко восстановить последовательность *ps.* Для этого сначала с помощью расширенного алгоритма Евклида он получает целые числа *а* и *b*, такие, что *ар* + *bq* = 1. Затем вычисляет





Значение *х0* позволяет вычислить любой член последовательности *х-*с той же эффективностью, как это может сделать отправитель.

Временная и емкостная сложность вычислений системы Блюма — Гольдвассер сопоставима со сложностью криптосистем RSA и Эль-Гамаля. Криптосистема Эль-Гамаля также обладает семантической стойкостью при соответствующем выборе начальных параметров[[2]](https://studme.org/239585/informatika/veroyatnostnoe_shifrovanie_sistema_blyuma_goldvasser" \l "gads_btm). Таким образом, эффективность вероятностной системы Блюма — Гольдвассер сопоставима с эффективностью шифра RSA а за счет побитового шифрования может быть даже быстрее в случае длинных сообщений (которые RSA должен зашифровывать по частям).

Быстродействие вероятностной системы шифрования Блюма — Гольдвассер может быть повышено за счет использования не одного младшего бита, а не более log2r| младших битов чисел *xjy* где г| — количество двоичных разрядов числа Блюма *п* (т.е. примерно log2log*2п* младших битов). Это не ослабит результирующую последовательность *psf* а само шифрование будет эффективнее RSA в log2r| раз[[3]](https://studme.org/239585/informatika/veroyatnostnoe_shifrovanie_sistema_blyuma_goldvasser" \l "gads_btm).

# **3. Сравнительный анализ криптосистем с открытым ключом**

Базой количественной оценки программ является не только аналитическо-иерархическая процедура Саати, повсеместно эксплуатируемая для точного определения весовых коэффициентов критериев качества. Помимо неё, также нашёл применение метод экспертных оценок, задачей которого является получение количественных значений критериев качества.

Для оценочного сравнения выбранных криптоалгоритмов проведём их сравнительный анализ посредством метода Саати [1]. Чуть ниже отображены выбранные критерии, на основании которых будет проводиться оценочная процедура:

А1 — Криптостойкость (MIPS);

А2 — Размер генерируемого ключа (до 4096 бит);

А3 —Генерация ключа;

А4 — Скорость шифрования (при длине модуля в 1024 бита);

А5 — Скорость дешифрования (при длине модуля в 1024 бита).

Используя аналитическо-иерархическую процедуру Саати, установим для каждого критерия качества его вес [2].

Правила заполнения матрицы парных сравнений представлены в таблице 3.1.

Таблица 3.1. Значения коэффициентов матрицы парных сравнений

|  |  |
| --- | --- |
| Xij | Значение |
| 1 | i-ый критерий практически равноценен j-му |
| 3 | i-ый критерий в меньшей степени важнее j-го |
| 5 | i-ый критерий важнее j-го |
| 7 | i-ый критерий в большей степени важнее j-го |
| 9 | i-ый критерий намного важнее j-го |

Матрица парных сравнений, веса критериев и средние геометрические занесены в таблицу 3.2.

Таблица 3.2. Матрица парных сравнений, средние геометрические и веса критериев.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A1 | A2 | A3 | A4 | A5 | Среднее геометрическое | Веса критериев |
| A1 | 1 | 3/1 | 7/1 | 5/1 | 5/1 | 3,5 | 0,49 |
| A2 | 1/3 | 1 | 5/1 | 5/1 | 5/1 | 2,11 | 0,29 |
| A3 | 1/7 | 1/5 | 1 | 3/1 | 3/1 | 0,76 | 0,11 |
| A4 | 1/5 | 1/5 | 1/3 | 1 | 1 | 0,42 | 0,06 |
| A5 | 1/5 | 1/5 | 1/3 | 1 | 1 | 0,42 | 0,06 |

На рисунке 3.1 изображена созданная на основании данных таблицы 2 диаграмма весовых коэффициентов для критериев A1, A2, A3, A4 и A5.

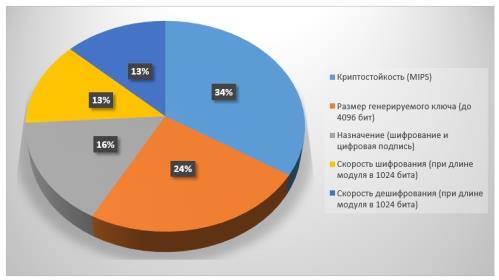


Рисунок 1. Весовые коэффициенты критериев качества

Чтобы проверить матрицу парных сравнений на непротиворечивость, произведём её проверку. Суммы столбцов матрицы парных сравнений [3]:

R1=1.88; R2=4.6; R3=13.67; R4=15; R5=15.

После вычислим дополнительную величину L, просуммировав весовые коэффициенты и произведения сумм столбцов матриц: L = 5,45.

Таким образом, индекс согласованности ИС = (L-N)/(N-1) = 0,113.

Следовательно, величина случайной согласованности для размерности матрицы парных сравнений: СлС = 1,24.

Отношение согласованности ОС=ИС/СлС = 0,09 не превышает 0,2, а значит, дополнительное уточнение матрицы парных сравнений не требуется [4].

Используя вычисленные коэффициенты, найдём интегральный показатель качества для следующих асимметричных алгоритмов шифрования данных: RSA, шифросистема Эль-Гамаля, обмен ключами Диффи—Хелмана, Система Блюма — Гольдвассер

Установим категориальную шкалу от нуля до семи (где 0 — качество не удовлетворительно, а 7 — предельно достижимый уровень качества) для установления функциональных возможностей выбранных криптоалгоритмов.

Значения весовых коэффициентов ai, соответствующие функциональным возможностям аналогов [5]:

1. Криптостойкость (MIPS): a1 = 0.34;
2. Размер генерируемого ключа (до 4096 бит): a2 = 0.24;
3. Генерация ключа: a3 = 0.16;
4. Скорость шифрования (при длине модуля в 1024 бита): a4 = 0.13;
5. Скорость дешифрования (при длине модуля в 1024 бита): a5 = 0.13;

где \sum a_i = 1 [6]. По выбранной шкале определим количественные значения функциональных возможностей Xij (таблица 3) и вычислим интегральные показатели качества для выбранных асимметричных алгоритмов шифрования: асимметричный криптоалгоритм шифрование

Таблица 3. Интегральные показатели качества

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Критерии** | **Весовые коэффициенты** | **Асимметричные алгоритмы** | | | | | **Базовые значения** |
| RSA | DSA | шифросистема Эль-Гамаля | обмен ключами Диффи-Хелмана | Система Блюма — Гольдвассер |
| Криптостойкость (MIPS) | 0,49 | 7 | 7 | 7 | 5 | 5 | 6,2 |
| Размер генерируемого ключа (до 4096 бит) | 0,29 | 5 | 3 | 7 | 3 | 5 | 4,6 |
| Генерация ключ | 0,11 | 7 | 7 | 7 | 3 | 3 | 5,45 |
| Скорость шифрования (при длине модуля в 1024 бита) | 0,06 | 7 | 7 | 3 | 5 | 3 | 4,95 |
| Скорость дешифрования (при длине модуля в 1024 бита) | 0,06 | 3 | 3 | 7 | 3 | 3 | 3,8 |
| Интегральные показатель качества Q | | 6,25 | 5,67 | 6,83 | 4,13 | 4,59 | 5,49 |

где Q_{j}=\sum a_i*X_{ij} *—* интегральный показатель качества для *j*-го криптоалгоритма.

На основе экспериментов было замечено, что RSA очень эффективен при шифровании, но медленен при расшифровании, это обусловлено тем, что открытая экспонента много меньше, чем закрытая экспонента, которая используется при расшифровании, при этом процедуры шифрования и расшифрования представлены в виде операции возведения в степень.

Схемы RSA и Эль-Гамаля показывают одинаковую скорость при расшифровании. Хорошо видно, что при генерации ключа (при 1 раунде) для схемы Эль-Гамаля наблюдаются скачки на графиках, это обусловлено тем, что алгоритм генерации зависит от генерации первообразного элемента, который в свою очередь зависит от факторизации чисел. Факторизация может работать в некоторых ситуациях быстрее, в некоторых дольше, но при 100 раундах график без явных скачков, но производительность хуже по той же причине.

При схеме Эль-Гамаля шифрограмма имеет заметно больший размер, чем при шифровании другими схемами.

Генерация ключа у данных схем шифрования приблизительно одинаковая. Таблицы видно, что схема Блюма-Гольдвассер имеет преимущество перед остальными криптосистемами по всем показателям. Наихудшим образом, с учетом всех операций, проявила себя схема Эль-Гамаля.

1. **Проектирование программного средства**

Программное обеспечение для визуализации криптосистем с открытым ключом, шифрование и расшифрование алгоритмами реализовано на объектно-ориентированном языке программирования С# на платформе Microsoft .NET Core версии 4.7 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;
* шифровать и расшифровывать сообщение, введенное пользователем;

Рассмотрим решение проекта «TEA», имеющим структуру, представленную на рисунке 5.1. Описание структуры проекта описано в таблице 5.1.

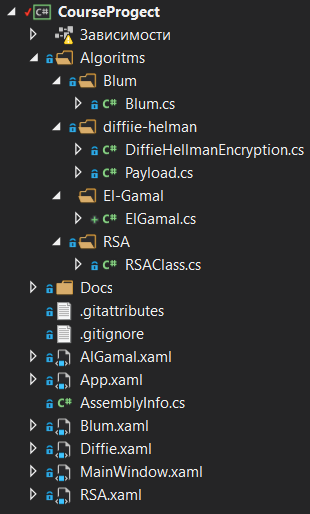


Рисунок 5.1 – Структура проекта

Таблица 5.1 – Описание структуры проекта

|  |  |
| --- | --- |
| Зависимости | Свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках |
| Algorithms | Директории с реализацией алгоритмов, используемых в курсовом проекте |
| Blum | Директория, с реализацией криптосистемы Блюма -Гольдвассер |
| RSA | Директория, с реализацией криптосистемы RSA |
| Diffie-Helman | Директория, с реализацией криптосистемы Диффи—Хелмана |
| El-Gamal | Директория, с реализацией криптосистемы Эль-Гамаля |

Программная реализация всех функций представлена в приложении А.

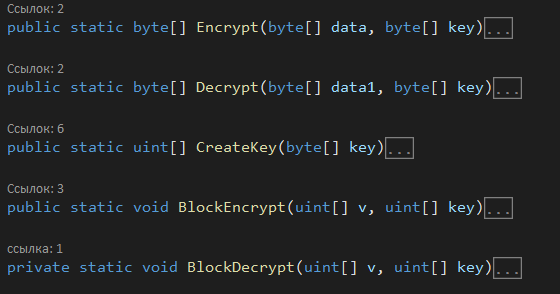


Рисунок 2.2 – Функции алгоритма

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.

**5.Тестирование программного средства**

**6.Руководство пользователя**

**Вывод**

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).

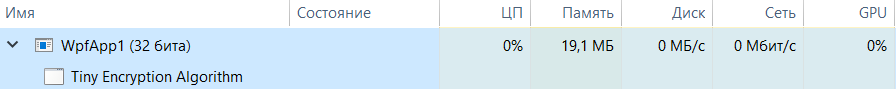


Рисунок 3.1 – Потребляемые ресурсы компьютера

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.

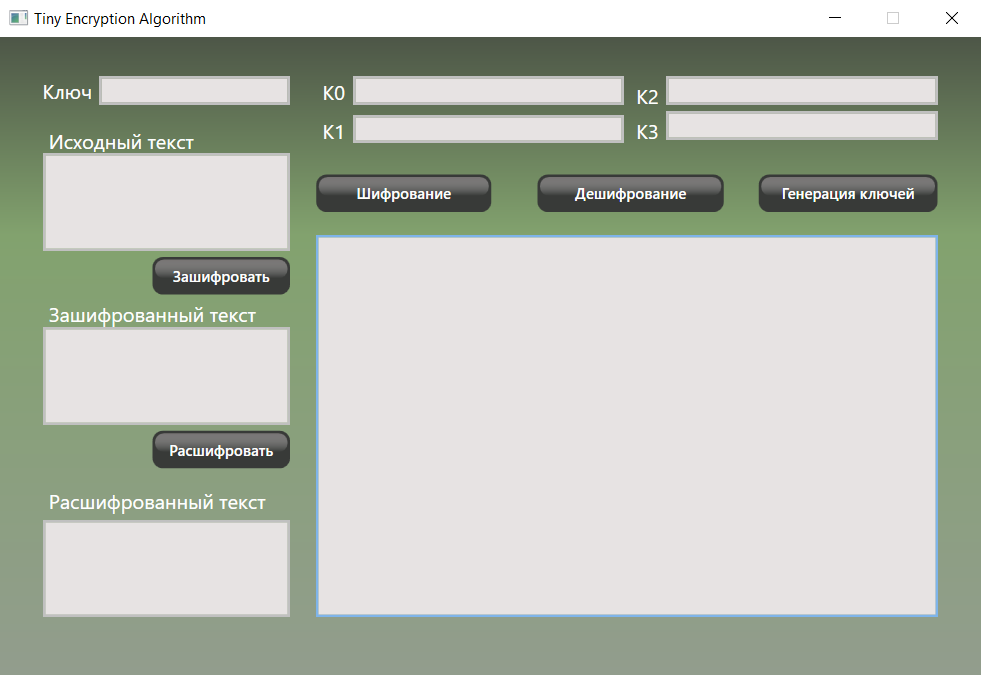


Рисунок 3.2 – Неактивные окна и кнопки

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.

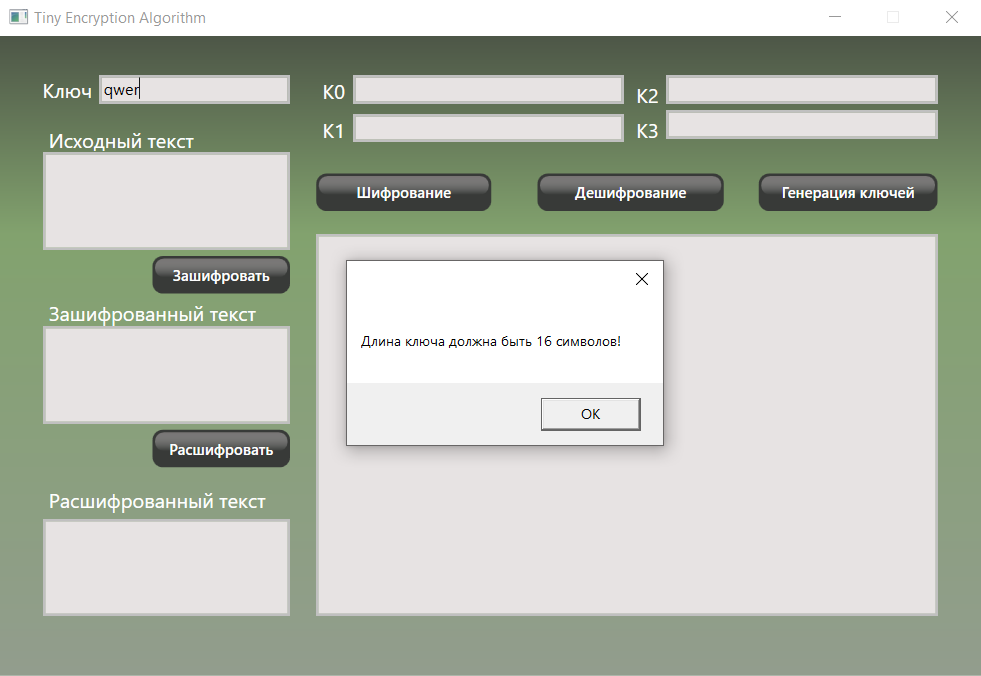


Рисунок 3.3 – Неверная длина ключа

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).

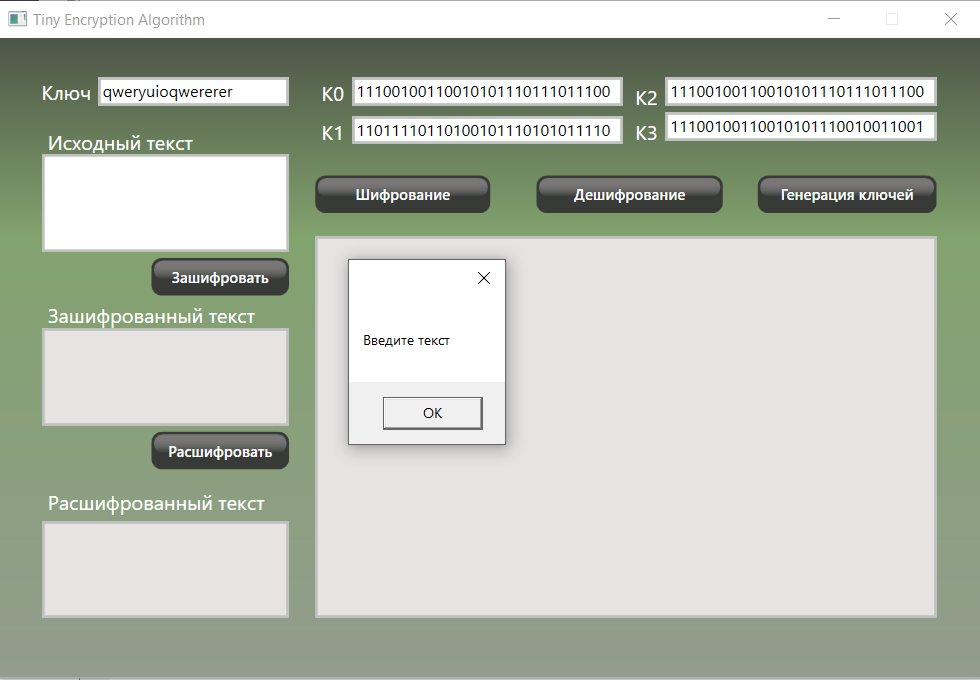


Рисунок 3.4 – Шифрование пустого сообщения

# **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).

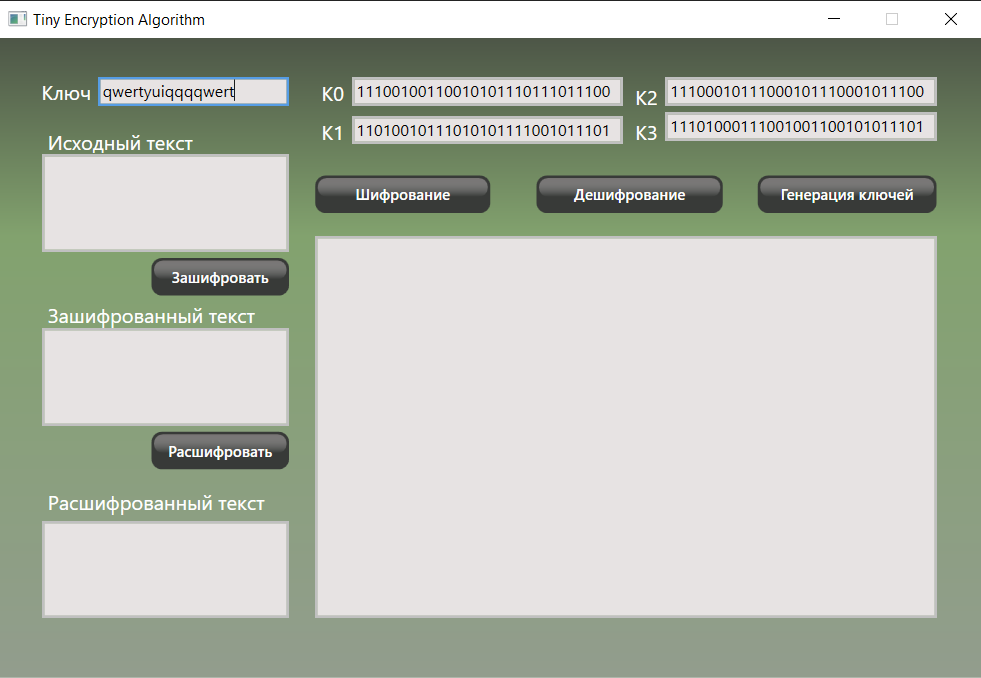


Рисунок 4.2 – Генерация подключей

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

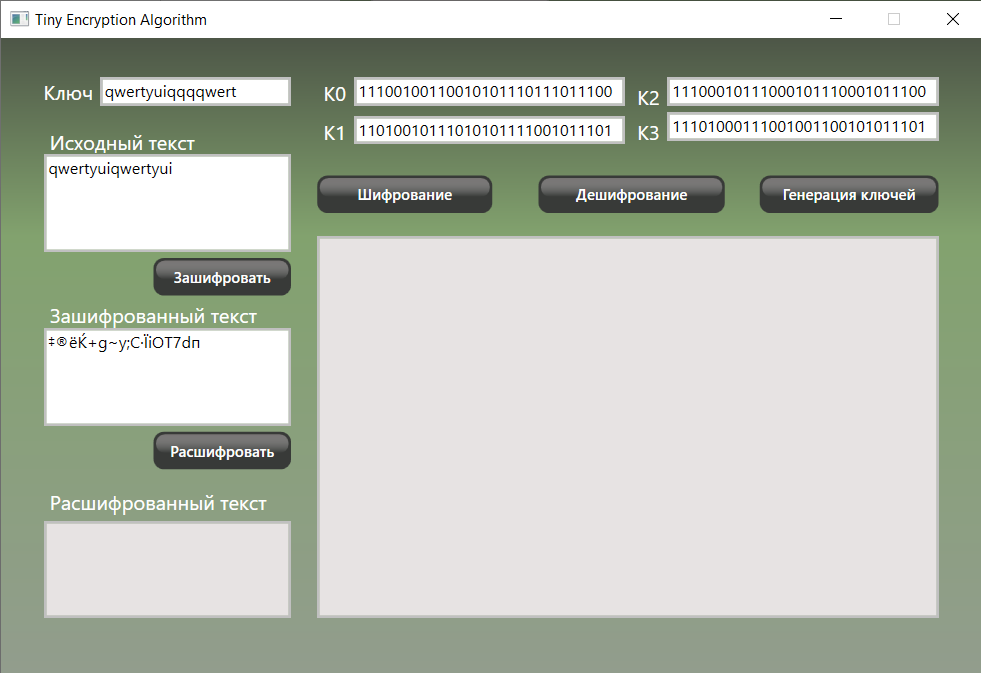


Рисунок 4.3 – Зашифрованное сообщение

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

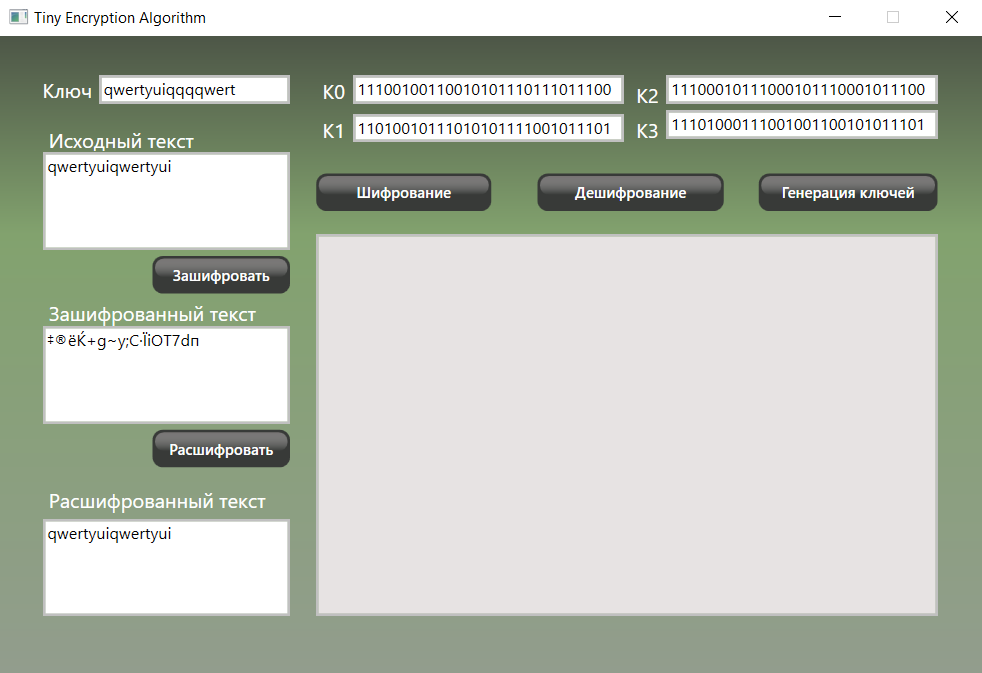


Рисунок 4.4 – Расшифрованное сообщение

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.

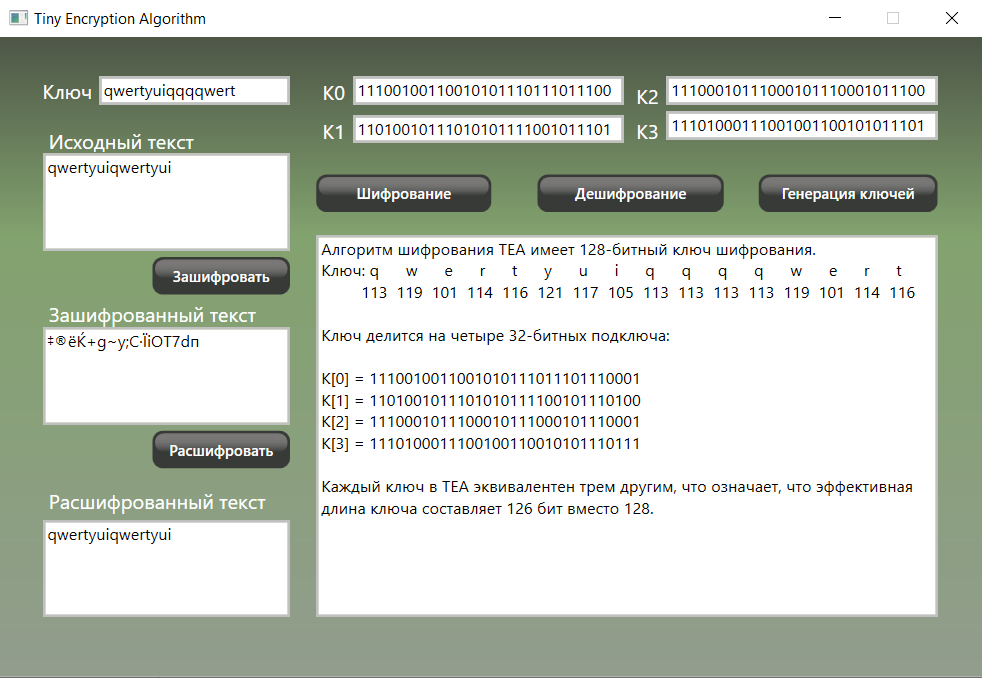


Рисунок 4.5 – Визуализация алгоритма генерации ключей

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.

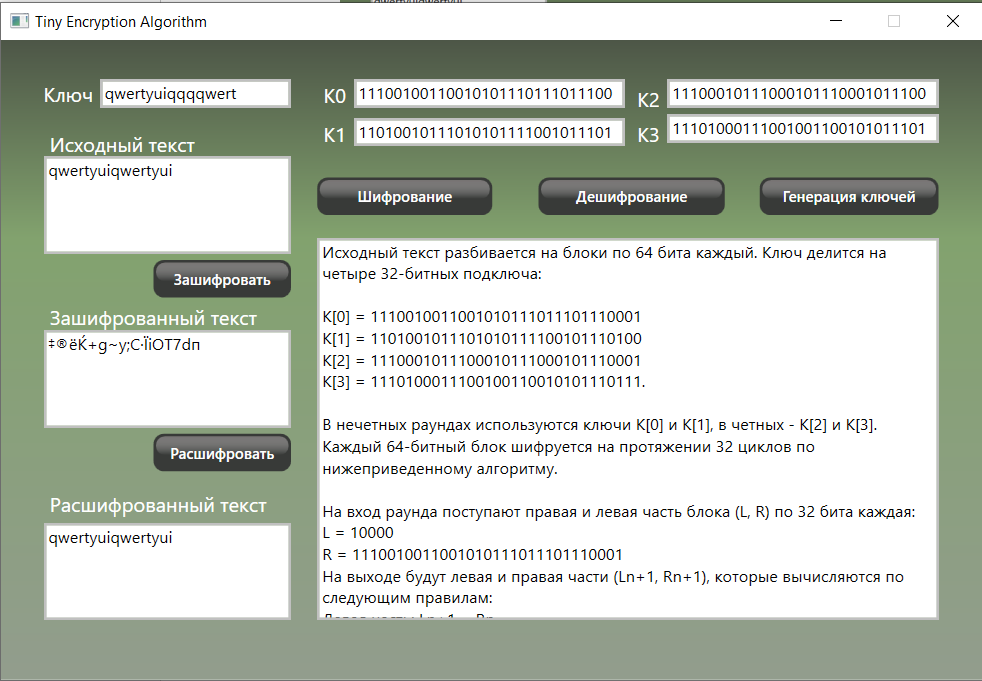


Рисунок 4.6 – Визуализация алгоритма шифрования

# **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**

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Text;

namespace CourseProgect.Algoritms.Blum

{

public class BlumGoldwasser

{

private int PrimeP;

private int PrimeQ;

private int IntegerA;

private int IntegerB;

private int XNaut;

public BlumGoldwasser(int primeP, int primeQ, int integerA, int integerB, int xNaut)

{

PrimeP = primeP;

PrimeQ = primeQ;

IntegerA = integerA;

IntegerB = integerB;

XNaut = xNaut;

}

public Tuple<List<string>, int> Encrypt(string inputMessege)

{

byte[] inputMessegeByte = Encoding.Unicode.GetBytes(inputMessege);

StringBuilder stringBuilder = new StringBuilder();

foreach (var item in inputMessegeByte)

{

stringBuilder.Append(Convert.ToString(item, 2));

}

string inputString = stringBuilder.ToString();

Tuple<List<string>, int> cyphertext = encrypt(inputString);

return cyphertext;

}

public Tuple<List<string>, int> encrypt(string message)

{

List<string> answer = new List<string>();

int N = PrimeP \* PrimeQ;

int k = (int)(Math.Log(N) / Math.Log(2));

int h = (int)(Math.Log(k) / Math.Log(2));

int t = message.Length / h;

if (message.Length % h != 0) { t++; }

List<string> choppedMessage = new List<string>();

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

{

choppedMessage.Add(message.Substring(i, h));

}

if (message.Length % h != 0)

{

choppedMessage.Add(message.Substring((message.Length / h) \* h));

}

long XsubI = XNaut;

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

{

string XsubIstring = Convert.ToString(XsubI, 2);

string leastSigBits = XsubIstring.Substring(XsubIstring.Length - h);

int PsubIasInt = int.Parse(leastSigBits);

int MsubIasInt = int.Parse(choppedMessage[i]);

int CsubIasInt = PsubIasInt ^ MsubIasInt;

string CsubI = CsubIasInt.ToString();

while (CsubI.Length < 4)

{

CsubI = "0" + CsubI;

}

answer.Add(CsubI);

XsubI = (XsubI \* XsubI) % N;

}

XsubI = (XsubI \* XsubI) % N;

return new Tuple<List<string>, int>(answer, (int)XsubI);

}

public string decrypt(Tuple<List<string>, int> cyphertext)

{

string answer = "";

int N = PrimeP \* PrimeQ;

int t = cyphertext.Item1.Count;

int valueD1 = powerWithModulus((PrimeP + 1) / 4, t + 1, PrimeP - 1);

int valueD2 = powerWithModulus((PrimeQ + 1) / 4, t + 1, PrimeQ - 1);

int valueU = powerWithModulus(cyphertext.Item2, valueD1, PrimeP);

int valueV = powerWithModulus(cyphertext.Item2, valueD2, PrimeQ);

long Vap = (long)valueV \* (long)IntegerA \* (long)PrimeP;

long ubq = (long)valueU \* (long)IntegerB \* (long)PrimeQ;

long Xnaut = (Vap + ubq) % N;

if (Xnaut < 0) { Xnaut = N - Xnaut; }

long XsubI = Xnaut;

int k = (int)(Math.Log(N) / Math.Log(2));

int h = (int)(Math.Log(k) / Math.Log(2));

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

{

string XsubIstring = Convert.ToString(XsubI, 2);

string leastSigBits = XsubIstring.Substring(XsubIstring.Length - h);

int PsubIasInt = int.Parse(leastSigBits);

int CsubIasInt = int.Parse(cyphertext.Item1[i]);

int MsubIasInt = PsubIasInt ^ CsubIasInt;

string MsubI = MsubIasInt.ToString();

while (MsubI.Length < 4)

{

MsubI = "0" + MsubI;

}

answer = answer + MsubI;

XsubI = (XsubI \* XsubI) % N;

}

return answer;

}

public int powerWithModulus(int bases, int exponent, int modulus)

{

if (exponent < 0)

{

}

long answer = 1;

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

{

answer = answer \* bases % modulus;

}

return (int)answer;

}

}

}

**Функции расшифрования сообщения**

using System;

using System.Collections.Generic;

using System.IO;

using System.Numerics;

using System.Security.Cryptography;

using System.Text;

namespace CourseProgect.Algoritms.diffiie\_helman

{

public class DiffieHellmanEncryption

{

public static byte[] encryptedMessage;

public static byte[] senderKey;

public static byte[] IV;

public static byte[] Person2PublicKey;

private static byte[] Key;

static UnicodeEncoding Encoder = new UnicodeEncoding();

private Aes aes = new AesCryptoServiceProvider();

public static byte[] Person1PublicKey;

public string Encrypt(string secretMessage)

{

using (ECDiffieHellmanCng ecd = new ECDiffieHellmanCng())

{

ecd.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

ecd.HashAlgorithm = CngAlgorithm.Sha256;

Person1PublicKey = ecd.PublicKey.ToByteArray();

CngKey k = CngKey.Import(Person1PublicKey, CngKeyBlobFormat.EccPublicBlob);

senderKey = ecd.DeriveKeyMaterial(CngKey.Import(Person1PublicKey, CngKeyBlobFormat.EccPublicBlob));

}

// Encrypt the message

using (MemoryStream ms = new MemoryStream())

using (CryptoStream cs = new CryptoStream(ms, aes.CreateEncryptor(), CryptoStreamMode.Write))

{

byte[] plainTextMessage = Encoding.UTF8.GetBytes(secretMessage);

cs.Write(plainTextMessage, 0, plainTextMessage.Length);

cs.Close();

encryptedMessage = ms.ToArray();

}

return Convert.ToBase64String(encryptedMessage);

}

public string Decrypt()

{

using (ECDiffieHellmanCng ecd = new ECDiffieHellmanCng())

{

ecd.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

ecd.HashAlgorithm = CngAlgorithm.Sha256;

Person2PublicKey = ecd.PublicKey.ToByteArray();

Key = ecd.DeriveKeyMaterial(CngKey.Import(Person1PublicKey, CngKeyBlobFormat.EccPublicBlob));

}

// Decrypt and show the message

using (MemoryStream ms = new MemoryStream())

using (CryptoStream cs = new CryptoStream(ms, aes.CreateDecryptor(), CryptoStreamMode.Write))

{

cs.Write(encryptedMessage, 0, encryptedMessage.Length);

cs.Close();

string message = Encoding.UTF8.GetString(ms.ToArray());

return message;

}

}

}

}

**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**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CourseProgect.Algoritms.Al\_Gamal

{

public static class ElGamal

{

private static string Crypt(int p, int g, int x, string inString)

{

var result = "";

var y = Power(g, x, p);

var rand = new Random();

Console.WriteLine($"Open key (p,g,y)=({p},{g},{y})");

Console.WriteLine($"Closed key x={x}");

foreach (int code in inString)

if (code > 0)

{

var k = rand.Next() % (p - 2) + 1; // 1 < k < (p-1)

var a = Power(g, k, p);

var b = Mul(Power(y, k, p), code, p);

result += a + " " + b + " ";

}

return result;

}

private static string Decrypt(int p, int x, string inText)

{

var result = "";

var arr = inText.Split(' ').Where(xx => xx != "").ToArray();

for (var i = 0; i < arr.Length; i += 2)

{

var a = int.Parse(arr[i]);

var b = int.Parse(arr[i + 1]);

if (a != 0 && b != 0)

{

var deM = Mul(b, Power(a, p - 1 - x, p), p);

// m=b\*(a^x)^(-1)mod p =b\*a^(p-1-x)mod p - т

var m = (char)deM;

result += m;

}

}

return result;

}

private static int Power(int a, int b, int n)

{

// a^b mod n

var tmp = a;

var sum = tmp;

for (var i = 1; i < b; i++)

{

for (var j = 1; j < a; j++)

{

sum += tmp;

if (sum >= n)

{

sum -= n;

}

}

tmp = sum;

}

return tmp;

}

private static int Mul(int a, int b, int n)

{

// a\*b mod n

var sum = 0;

for (var i = 0; i < b; i++)

{

sum += a;

if (sum >= n)

{

sum -= n;

}

}

return sum;

}

public static string EnCrypt(string str)

{

return Crypt(593, 123, 8, str);

}

public static string DeCrypt(string str)

{

return Decrypt(593, 8, str);

}

}

}

**Функция расшифрования блока сообщения**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Numerics;

using System.Security.Cryptography;

using System.Text;

namespace CourseProgect.Algoritms

{

public class RSAClass

{

static RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(512);

static UnicodeEncoding Encoding = new UnicodeEncoding();

static byte[] plaintext;

static byte[] encryptedtext;

public static string EncryptMessage(string inputValue)

{

plaintext = Encoding.GetBytes(inputValue);

encryptedtext = Encryption(plaintext, RSA.ExportParameters(false), false);

return Encoding.GetString(encryptedtext);

}

public static string DecryptMessage()

{

byte[] decryptedtex = Decryption(encryptedtext, RSA.ExportParameters(true), false);

return Encoding.GetString(decryptedtex);

}

static public byte[] Encryption(byte[] Data, RSAParameters RSAKey, bool DoOAEPPadding)

{

try

{

byte[] encryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKey);

encryptedData = RSA.Encrypt(Data, DoOAEPPadding);

}

return encryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

return null;

}

}

static public byte[] Decryption(byte[] Data, RSAParameters RSAKey, bool DoOAEPPadding)

{

try

{

byte[] decryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKey);

decryptedData = RSA.Decrypt(Data, DoOAEPPadding);

}

return decryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.ToString());

return null;

}

}

}

}

# **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

{ }