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: «Сравнительный анализ криптостойкости ассиметриных алгоритмов шифрования»

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](#_Toc71758472)

[**1.Аналитический обзор литературы** 4](#_Toc71758473)

[**1.1.Генерация ключа** 5](#_Toc71758474)

[**1.2.Шифрование** 5](#_Toc71758475)

[**1.3.Расшифрование** 5](#_Toc71758476)

[**2.Практические основы криптосистем с открытым ключом** 7](#_Toc71758477)

[**2.1 RSA** 7](#_Toc71758478)

[**2.2 El-Gamal** 8](#_Toc71758479)

[**2.3 Diffie-Helman** 8](#_Toc71758480)

[**2.4 Blum-Goldwasser** 9](#_Toc71758481)

[**3. Сравнительный анализ криптосистем с открытым ключом** 11](#_Toc71758482)

[**3.1. Описание метода сравнения** 11](#_Toc71758483)

[**3.2. Анализ криптосистем** 14](#_Toc71758484)

[**4.Проектирование программного средства** 17](#_Toc71758485)

[**4.1 Blum** 18](#_Toc71758486)

[**4.2.Эль-Гамаль** 19](#_Toc71758487)

[**4.3 RSA** 20](#_Toc71758488)

[**4.4 Diffie-Hellman** 21](#_Toc71758489)

[**5.Тестирование программного средства** 21](#_Toc71758490)

[**6.Руководство пользователя** 24](#_Toc71758491)

[**Заключение** 30](#_Toc71758492)

[**Список используемых источников** 31](#_Toc71758493)

[**Приложение А** 32](#_Toc71758494)

# **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.

1. **Аналитический обзор литературы**

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.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].

**1.2.Шифрование**

Шифрование с открытым ключом состоит из следующих шагов рисунок 1.2.1.

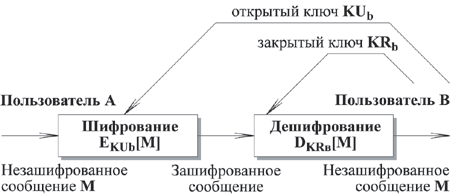


Рисунок 1.2.1 – Схема шифрования с открытым ключом

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.

**1.3.Расшифрование**

Создание и проверка подписи состоит из следующих шагов представлено на рисунке 1.3.1:

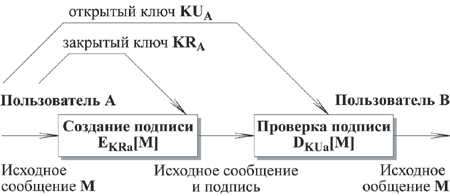


Рисунок1.3.1. Создание и проверка подписи

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.

# **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.

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].

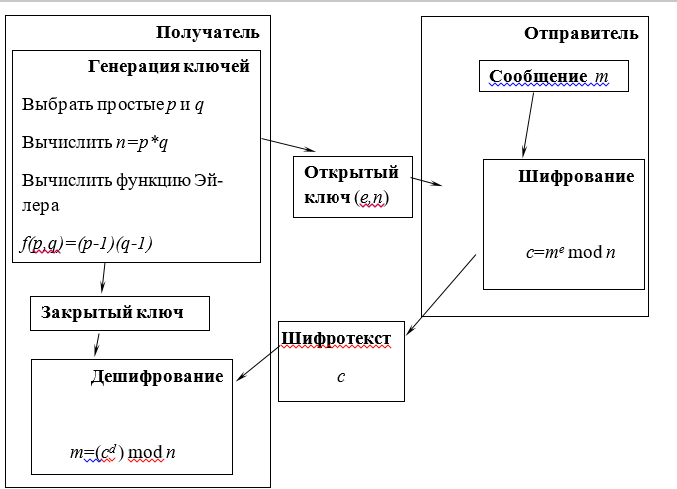


Рисунок 2.1.1. Схема шифрования и дешифрации алгоритма RSA

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 выражение как на рисунке 2.2.1

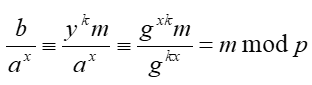


Рисунок 2.2.1— Отношение b к ax

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-Hellman**

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 .

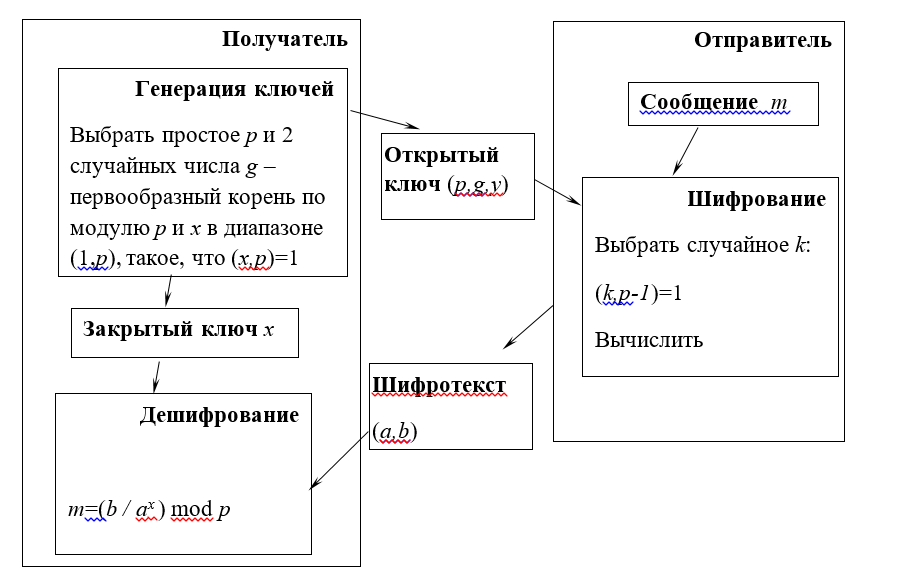


Рисунок 2.3.1 Принцип работы алгоритма

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  *p w q* 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 значения представленные на рисунке 2.4.2.

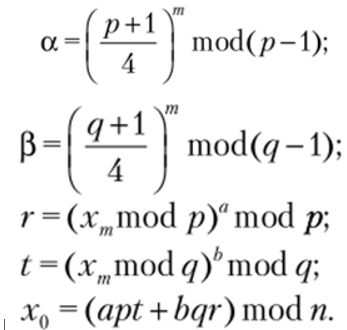


Рисунок 2.4.2 – Значения для расчётов

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. 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. Сравнительный анализ криптосистем с открытым ключом**

**3.1. Описание метода сравнения**

Метод анализа иерархий был предложен в конце 1970-х гг. американским математиком Т. Саати. Метод состоит в декомпозиции проблемы на более простые составляющие части и поэтапном установлении приоритетов оцениваемых компоненты с использованием парных (попарных) сравнений.

На первом этапе выявляются наиболее важные элементы проблемы. Па втором — наилучший способ поверки наблюдений, испытания и оценки элементов. На третьем — осуществляется выработка способа применения решения и оценка его качества.

Весь процесс подвергается проверке и переосмыслению до тех пор, пока не будет уверенности, что процесс охватил все важные характеристики, необходимые для представления и решения проблемы.

Процесс может быть проведен над последовательностью иерархий. При этом результаты, полученные в одной из них, используются в качестве входных данных при изучении следующей.

В наиболее простой иерархии, называемой Саати доминантной, он определяет три уровня: верхний уровень цели (или целей), средний — критерии, нижний — перечень альтернатив рисунок 3.1.1.

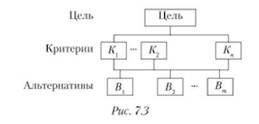


Рисунок 3.1.1 – иерархия Саати

В структуре между целью и альтернативами может быть несколько промежуточных уровней (рисунок. 7.4). Например, уровень проблем, акторов (уровень действующих сил, в качестве которых могут выступать административные власти, жители и т.п.). Каждый из критериев может разделяться на субкритерии.



Рисунок 3.1.2. – Промежуточные уровни

Иерархия считается полной, если каждый элемент заданного уровня функционирует как критерий для всех элементов нижележащего уровня. Иерархия может быть разделена па подиерархии.

Связи между уровнями часто изображают так, как показано на рис. 3.1.2.

Для реализации метода введен закон иерархической непрерывности, в соответствии с которым требуется, чтобы элементы каждого уровня были сравнимы по отношению к элементам вышестоящего уровня.

Между уровнями строятся матрицы. Для структуры, приведенной на рис. 3.1.1, матрицы строятся следующим образом: одна матрица для сравнения относительной важности критериев по отношению к цели и матрицы для оценки относительной значимости альтернатив относительно каждого из критериев второго уровня. Число матриц между уровнем критериев и альтернатив равно числу критериев. Общее число матриц равно числу критериев плюс одна для оценки критериев относительно цели.

Используемые в методе Саати попарные сравнения приводят к квадратным матрицам вида как на рисунке 3.1.3.

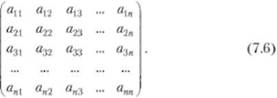


Рисунок 3.1.3 – Квадратная матрица саравнения

Эта матрица имеет свойство обратной симметрии, из выражения на рисунке 3.1.4 .

https://studme.org/imag/manag/vol_msistpr/image468.jpg

Рисунок 3.1.4 – Свойство обратной симметрии

Из рисунка индексы j и i относятся к строке и к столбцу соответственно. Обратные числа использованы в дальнейшем при обработке матрицы.

В матрицах элементы нижележащего уровня (альтернативы, варианты) сравниваются попарно по отношению к критериям, а критерии — по отношению к цели.

Эти оценки могут получаться различными способами. Но в методе Саати для оценки компонент рекомендуется специальная шкала от 1 до 9, в которой компонентам равной важности ставится в соответствие единица, при умеренном превосходстве — 3, при существенном превосходстве — 5, значительном превосходстве — 7 и очень сильном превосходстве — 9. Значения 2, 4, 6, 8 используются как промежуточные между двумя соседними компонентами.

Относительная важность любого элемента, сравниваемого с самим собой, равна 1, т.е. диагональ матрицы состоит из единиц. При заполнении матрицы используется свойство обратной симметрии: симметричные клетки заполняются обратными величинами.

**3.2. Анализ криптосистем**

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 — Скорость генерации 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.

Правила заполнения матрицы парных сравнений представлены в таблице 3.2.1.

Таблица 3.2.1. 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 3.2.2.

Таблица 3.2.2. 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 3.2.1 изображена созданная на основании данных таблицы 2 диаграмма весовых коэффициентов для критериев A1, A2, A3, A4 и A5.



Рисунок 3.2.1 -- Весовые коэффициенты критериев качества

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

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 .

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 :

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 |  | шифросистема Эль-Гамаля | обмен ключами Диффи-Хеллмана | Система Блюма — Гольдвассер |
| Криптостойкость (MIPS) | 0,49 | 7 |  | 7 | 5 | 5 | 6,2 |
| Размер генерируемого ключа (до 4096 бит) | 0,29 | 5 |  | 7 | 3 | 5 | 4,6 |
| Скорость генерации ключей | 0,11 | 7 |  | 7 | 3 | 3 | 5,45 |
| Скорость шифрования (при длине модуля в 1024 бита) | 0,06 | 7 |  | 7 | 5 | 3 | 4,95 |
| Скорость дешифрования (при длине модуля в 1024 бита) | 0,06 | 3 |  | 3 | 3 | 3 | 3,8 |
| Интегральные показатель качества Q | | 6,25 |  | 6,83 | 4,13 | 4,59 | 5,49 |

где Qi *—* 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 для схемы Эль-Гамаля наблюдаются скачки, 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.При схеме Эль-Гамаля шифрограмма имеет заметно больший размер, чем при шифровании другими схемами. Диффи-Хеллман ведёт себя достаточно нейтрально относительно других алгоритмов.

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.

**4.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 «СourseProgect», имеющим структуру, представленную на рисунке 4.1. Описание структуры проекта описано в таблице 4.1.

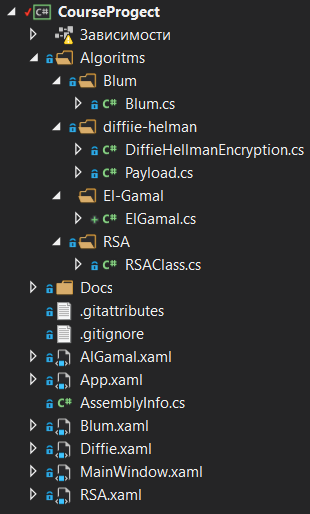


Рисунок 4.1 – Структура проекта

Таблица 4.1 – 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.

**4.1 Blum**

На рисунке 4.1.1 показан алгоритм Блюма-Гольдвассер

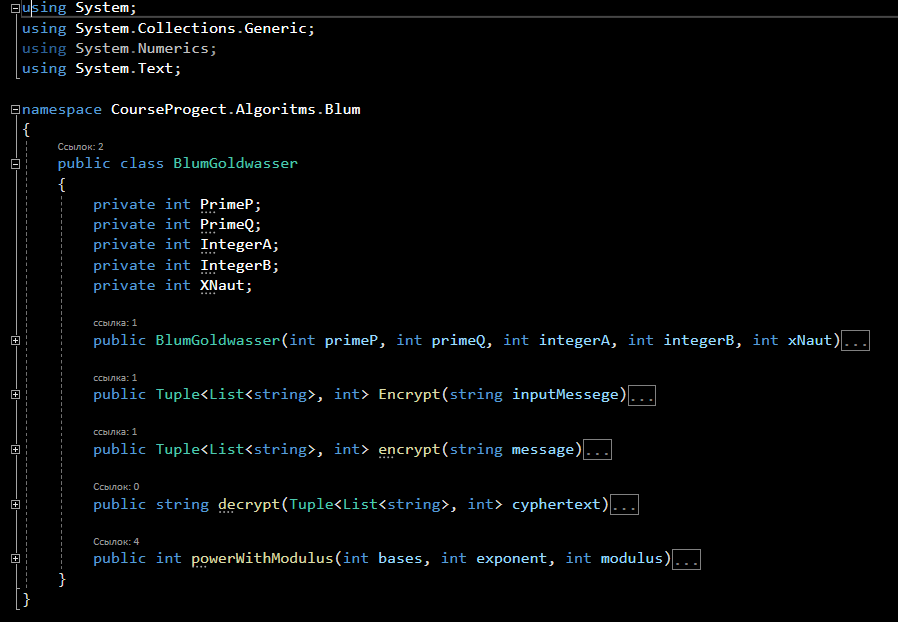


Рисунок 4.1.1 – Функции алгоритма

Функция Encrypt предназначена для принятия сообщения в качестве параметров и вызова функции encrypt. На выходе из функции получается сообщение, которое затем передаётся в функцию encrypt .

В свою очередь функция encrypt предназначена для шифрования сообщения.

Генерируется ключи генератором Блюма-Гольдвассер и затем идёт шифрование.

Функция Decrypt декодирует входящую последовательность и на выходе мы получаем и сходное сообщение.

* 1. **Эль-Гамаль**

На рисунке 4.2.1 показана реализация алгоритма Эль-Гамаля.

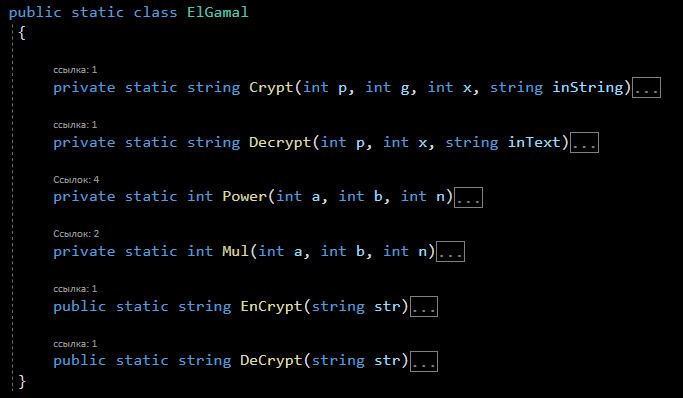
****

Рисунок 4.2.1 Функции алгоритма

В функции EnCrypt мы вызываем функцию Crypt(), в которую предаём x, g и p для генерации открытого и закрытого ключей. Так же мы передаём исходное сообщение, которое потом будет шифроваться.

Функция Crypt шифрует каждый символ исходного сообщения, предварительно сгенерировав закрытый ключ. Так же эта функция содержит вызов функций Power и Mul, которые возводят а в степень b по модулю n и умножают a на b по модулю n.

Функция DeCrypt предназначена для передачи сообщения в Decrypt, а также значений p и x.

Функция Decrypt дешифрует полученное сообщение и возвращает строку.

**4.3 RSA**

На рисунке 4.3.1 показана реализация алгоритма RSA.

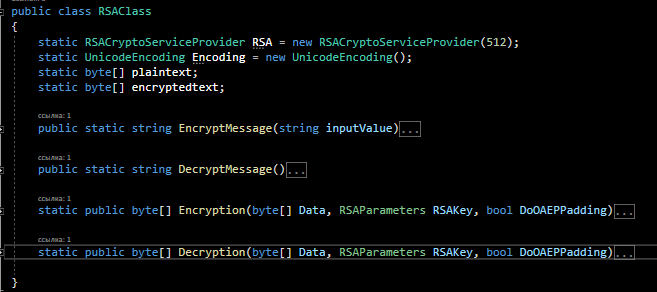


Рисунок 4.3.1 Функция алгоритма

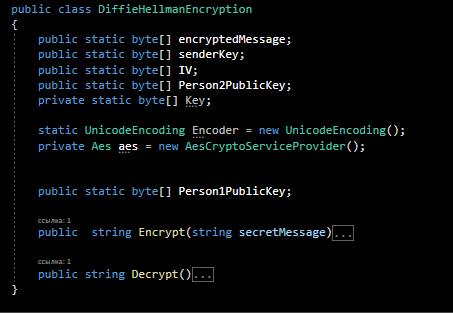
Функция EncryptMessage принимает в качестве параметра сообщение, которое конвертируется в байты и затем передаётся в функцию Encryption.

Encryption реализовывает шифрование сообщения в массив байт. Данная функция реализовывает библиотеку System.Sequrity.Cryptograthy.

Dencryption реализовывает расшифрование сообщения в массив байт. Данная функция реализовывает библиотеку System.Sequrity.Cryptograthy.

**4.4 Diffie-Hellman**

На рисунке 4.4.1 показана реализация алгоритма RSA.



4.4.1 Функции алгоритма

Encrypt реализовывает шифрование сообщения в массив байт. Данная функция реализовывает библиотеку System.Sequrity.Cryptograthy.

Dencrypt реализовывает расшифрование сообщения в массив байт. Данная функция реализовывает библиотеку System.Sequrity.Cryptograthy.

**5.Тестирование программного средства**

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).

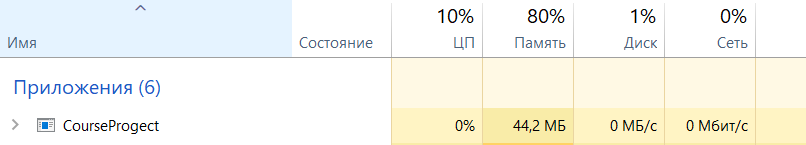


Рисунок 5.1 – Потребляемые ресурсы компьютера

Ниже приведены основные исключительные ситуации.

Перед началом работы приложения все кнопки и окна приложения находятся в активном состоянии (рисунок 5.2). Кнопки станут активны после того как окна «Ключ» и «Исходный текст» будут заполнены.

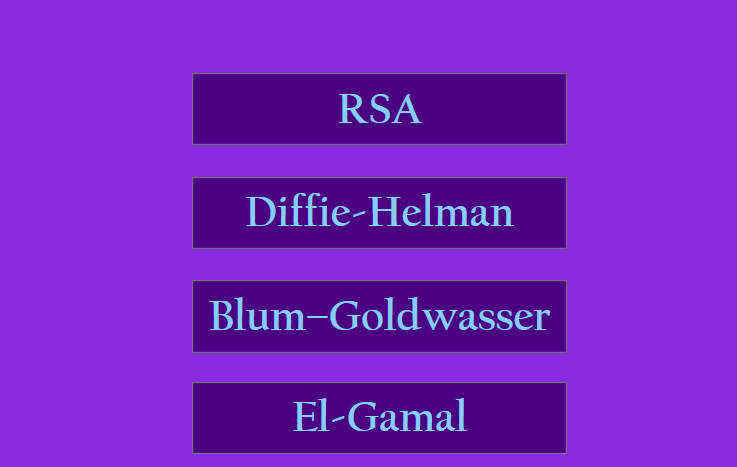


Рисунок 5.2 – Активные окна и кнопки

Для шифрования необходимо ввести исходное сообщение. Для этого необходимо кликнуть по полю ввода и оно станет активным. При попытке ввести пустое сообщение пользователь будет оповещен сообщением, которое представлено на рисунке 5.3. И в случае, исходное сообщение пустое, пользователь должен заполнить поле ввода либо выбрать файл

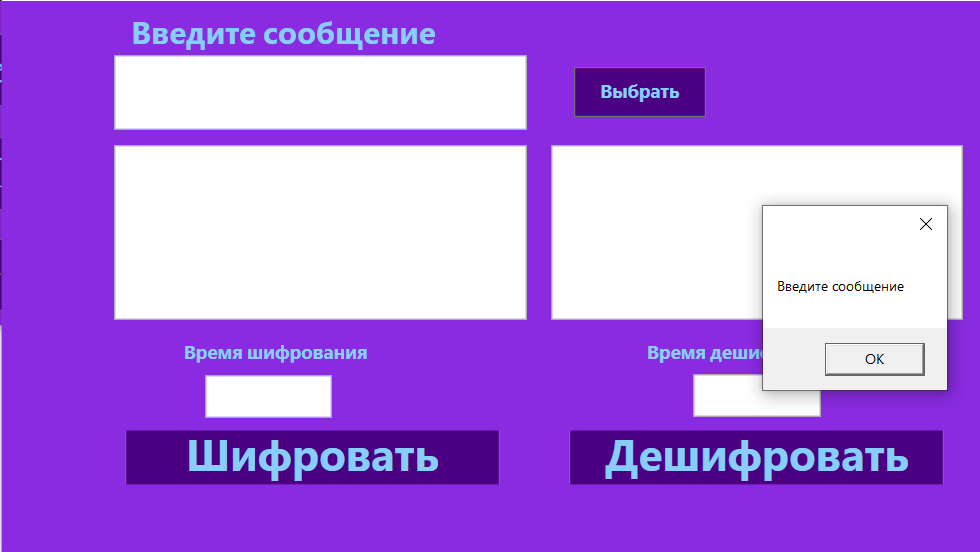


Рисунок 5.3 – Пустое сообщение

После корректного ввода сообщения пользователь уже может нажать кнопку Decrypt и произойдёт шифрование (рисунок 5.4).

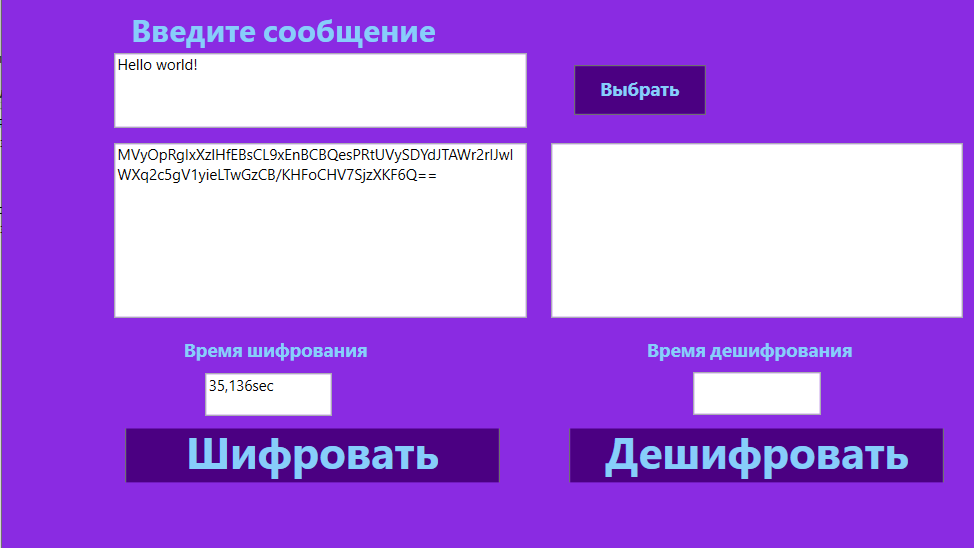


Рисунок 5.4 – Шифрование сообщения

Для дешифрования необходимо сначала зашифровать сообщение, чтобы поле Chiphered Text было непустым. В случае, если поле будет пустым, а кнопка Decrypt будет нажата, появится окошко как на рисунке 5.5

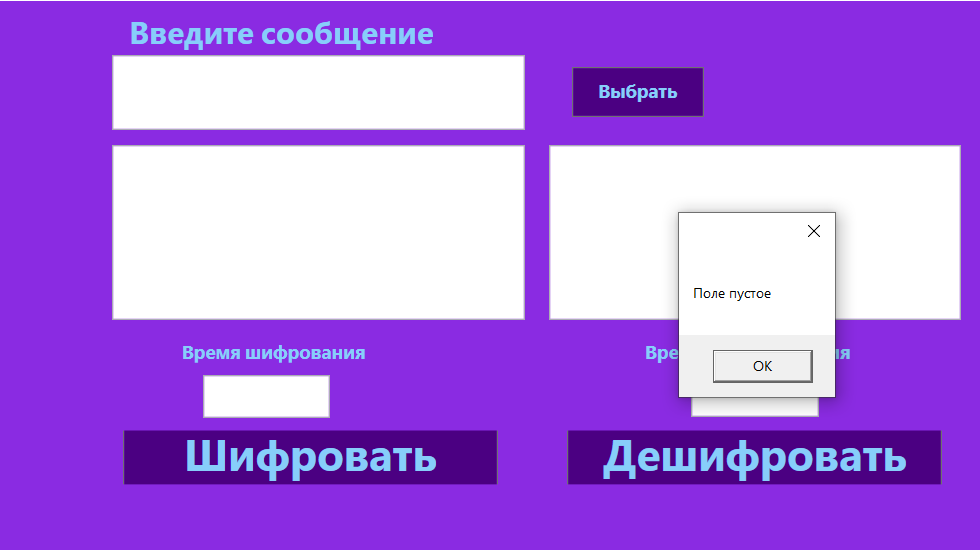


Рисунок 5.4 – Пустое поле для дешифрования

**6.Руководство пользователя**

Чтобы запустить приложение нужно запустить exe-файл – CourseProgect.exe. При запуске открывается окно приложения (рисунок 6.1).



Рисунок 6.1 – Окно приложения

Окно содержит кнопки алгоритмов, исследуемых в рамках моего курсового проекта.

В данном окне можно выбрать алгоритм.

После запуска приложения необходимо выбрать исследуемый алгоритм (рисунок 6.1). Я буду показывать функциональность приложения на примере алгоритма RSA. После выбора алгоритма откроется окно как на рисунке 6.2.

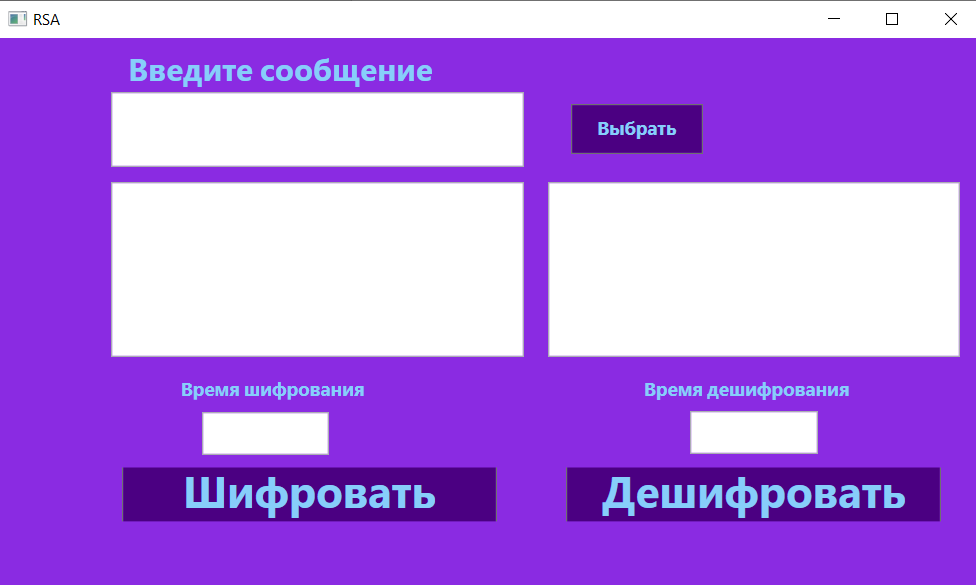


Рисунок 6.2 – Окно RSA

В окно «Введите сообщение» нужно ввести сообщение на английском языке, а затем нажать кнопку «Шифровать». В окно над кнопкой будет выведено зашифрованное сообщение и время, потраченное на шифрование(рисунок 6.3).

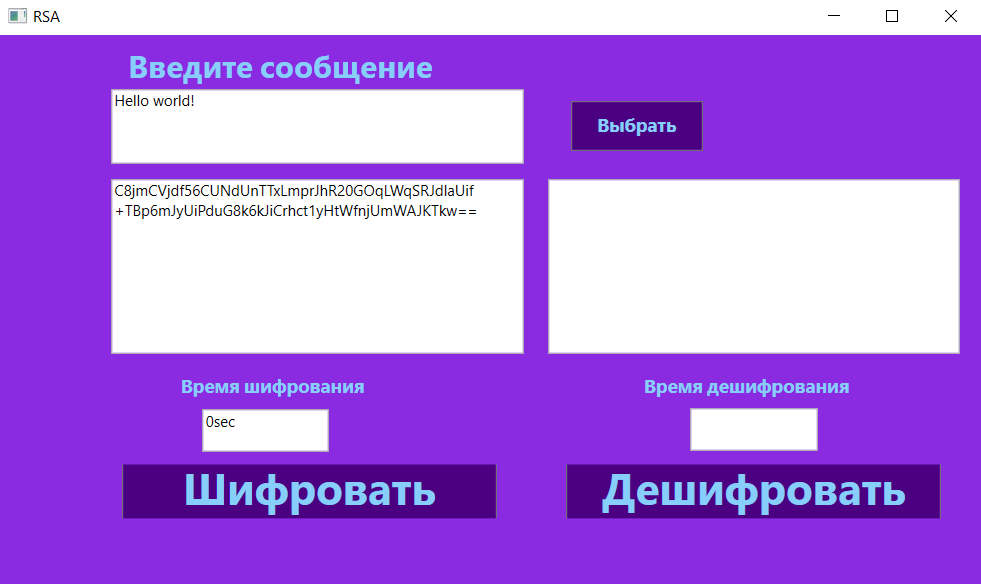


Рисунок 6.3 – Зашифрованное сообщение

В данной программе пользователь может не только вводить данные в поле для ввода сообщения, но и выбирать файл с форматом \*.txt. Для начала нужно нажать на кнопку “Выбрать” как показано на рисунке 6.4, затем выбрать файл \*.txt и нажать “Открыть” ,как показано на рисунке 6.5.

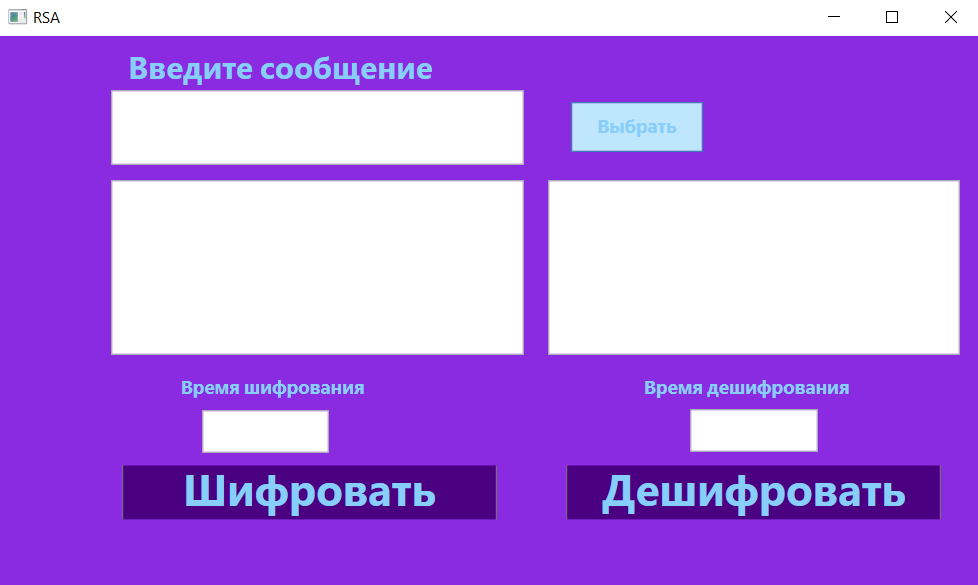


Рисунок 6.4 – Открытие

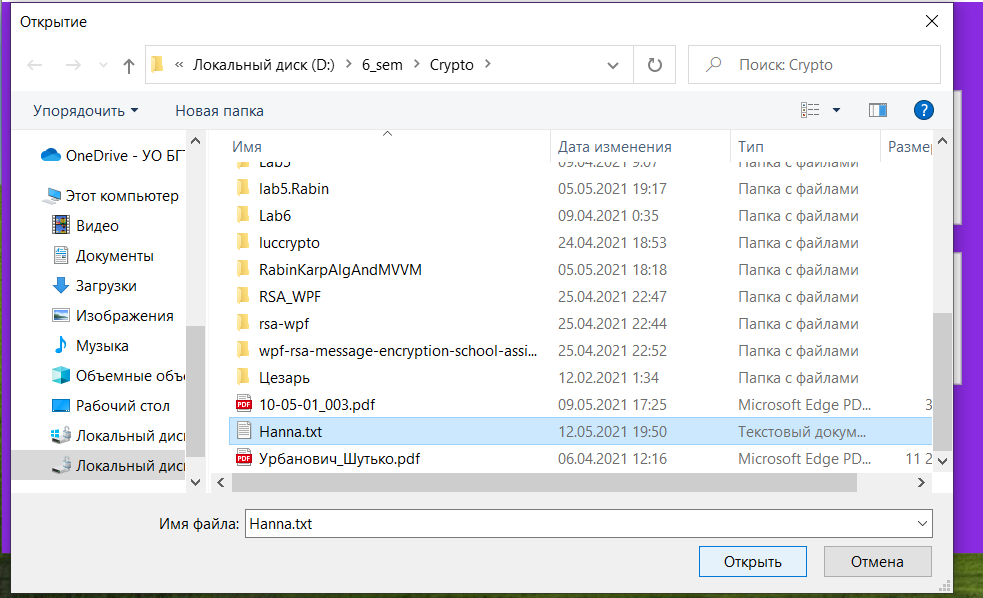
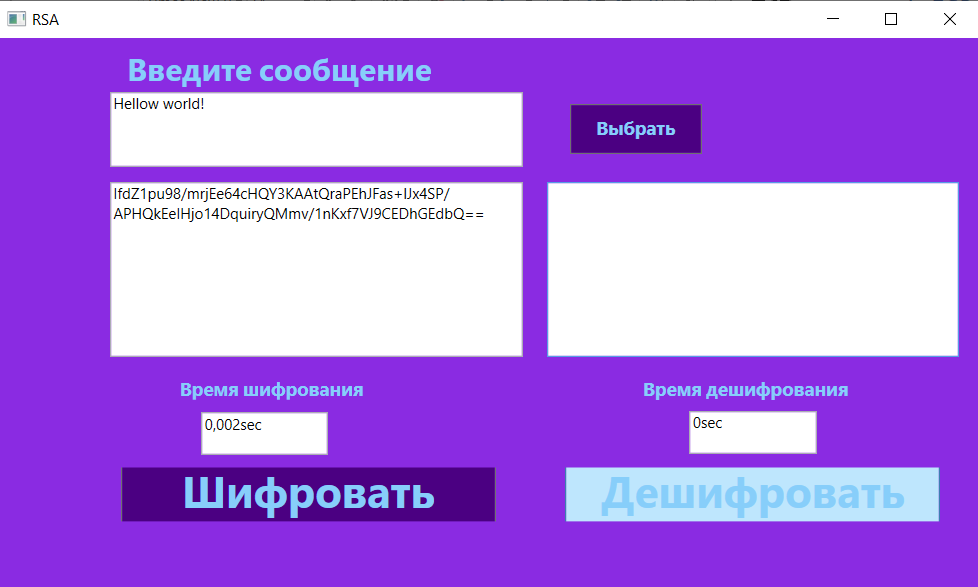


Рисунок 6.5 Выбор файла для шифрования

После того как получили зашифрованное сообщение, расшифруем его, нажатием на кнопку «Дешифровать» (рисунок 6.6). Расшифрованное сообщение и время, потраченное на дешифрование представлена на рисунке 6.7.



6.6 – Нажатие кнопки «Дешифрование»

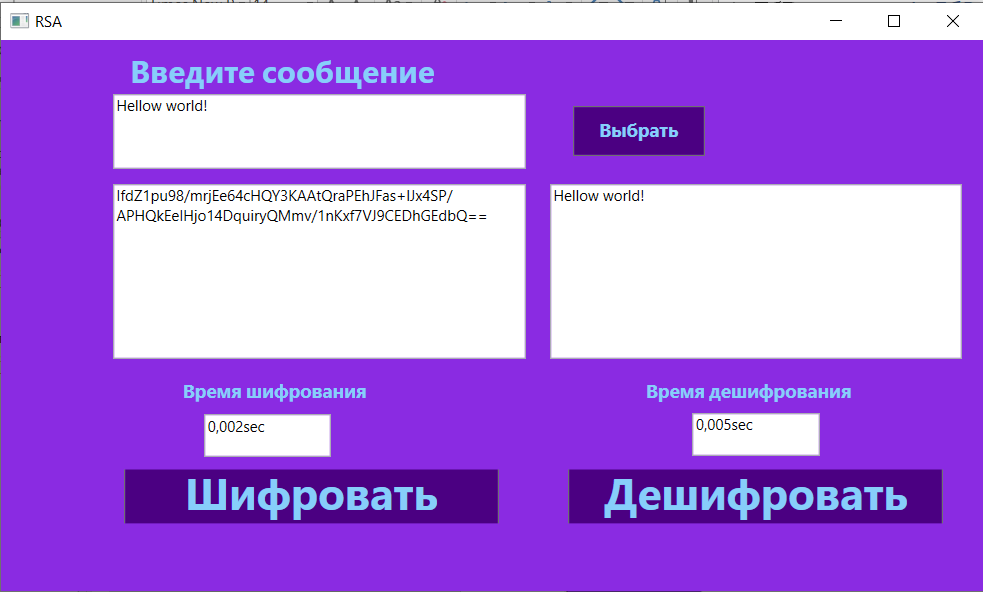


Рисунок 6.7 – Дешифрованное сообщение

В данном руководстве был пример использования алгоритма RSA. Функциональность остальных окон выполняются по аналогии с данным руководством.

**Заключение**

Данный проект был разработан для сравнения работы ассиметричных криптосистем по заданным критериям. Главной задачей было сделать максимально простое и функциональное приложение, а так же анализ криптосистем по методу Саати. 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.

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

Схемы RSA и Эль-Гамаля показывают одинаковую скорость при дешифровании. Хорошо видно, что при генерации ключа для схемы Эль-Гамаля наблюдаются скачки, 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. При схеме Эль-Гамаля шифрограмма имеет заметно больший размер, чем при шифровании другими схемами.

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

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. Кондрацкий Д.Е., Рыбанов А.А. Исследование методов и алгоритмов автоматизированной системы оценки альтернативных вариантов методом Т. Саати // NovaInfo.Ru. 2016. Т. 3. № 46. С. 107-116.
2. Рыбанов А. Определение весовых коэффициентов сложности тем учебного курса на основе алгоритма Cаати // Педагогические измерения. 2014. № 4. С. 21-28.
3. Рыбанов А.А., Макушкина Л.А. Технология определения весовых коэффициентов сложности тем дистанционного курса на основе алгоритма Саати // Открытое и дистанционное образование. 2016. № 1 (61). С. 69-79.
4. Саломаа, А. Р. Криптография с открытым ключом / А. Р. Саломаа. – М.: Мир, 2014. – 318 с.
5. Фомичёв, В. М. Дискретная математика и криптология / В. М. Фомичёв. – М.: ДИАЛОГ-МИФИ, 2013. – 400 с.
6. Агибалов, Г. П. Конечные автоматы в криптографии / Г. П. Агибалов. – М.: РАГС, 2014. – 78 с.
7. Каров, А. П. Шрифтовые технологии. Описание и инструментарий / А. П. Каров. – М.: Мир, 2015, – 454 с.
8. Ростовцев, А. Г. Методы криптоанализа классических шифров / А. Г. Ростовцев, Н. В. Михайлова. – М.: Наука, 2012. – 208 с.

# **Приложение А**

**Алгоритм Блюма-Гольдвассер**

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 алгоритма Диффи-Хеллмана**

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;

}

}

}

}

**Функции алгоритма RSA**

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 Convert.ToBase64String(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;

}

}

}

}

**Функции алгоритма Эль-Гамаля**

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;

}

}

}

}