Введение

Человечество перешло в эру, когда денежные транзакции, любая переписка или обмен секретными документами осуществляется с помощью компьютерных коммуникаций, таких как Internet. Ведь этой возможностью можно воспользоваться практически из любой точки земного шара и совершить обмен в считанные секунды.

В связи с этим наиболее актуальной и перспективной темой в XXI веке – веке информационных технологий – является работа с информацией, а главным приоритетом становится её защита, а в частности защита потоков данных.

Применение криптографии позволяет эффективно решить проблему обмена информацией через открытые сети. Чаще всего встречающимися криптографическими средствами, обеспечивающими безопасность, являются шифрование, аутентификация паролем и цифровые подписи.

Современная криптография – широкая область знаний, которая сложилась в результате усиленных исследований на протяжение последних четырех десятилетий. Он включает в себя поиск решений основных задач информационной безопасности – целостности и контроля участников, конфиденциальности и аутентификации.

Криптоалгоритмы делятся на две большие группы: симметричные (AES, ГОСТ, …) и ассиметричные (RSA, El-Gamal). Широкое распространение получили ассиметричные алгоритмы (с открытым ключом), ведь без них было бы невозможно развитие основных криптографических протоколов, аутентификации и электронно-цифровой подписи.

Широкое распространение данные алгоритмы шифрования получили в связи с необходимостью иметь пару ключей – открытый для зашифрования и закрытый для расшифрования. В соответствии с этим, вводя понятие открытого ключа, то есть ключа, который находится в

свободном доступе, пропадает необходимость решения сложнейшей задачи обмена секретными ключами.

Самым известным и распространенным алгоритмом ассиметричного шифрования является алгоритм RSA, который основан на идеи использования односторонних функций при шифровании.

Алгоритм RSA используется в большинстве защитных интернет систем для шифрования паролей или ключей шифрования к симметричным алгоритмам. Он очень редко используется для шифрования текстов в связи с «медленной» скоростью шифровки.

Цель данной работы: использование параллельного алгоритма RSA для шифрования данных. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести теоретический обзор алгоритмов шифрования;
2) разработать параллельный алгоритм шифрования;
3) сделать выбор программной среды для реализации алгоритма;
4) произвести реализацию алгоритма в выбранной среде;
5) тестировать запрограммированный алгоритм шифрования;
6) построить иллюстративные примеры и провести численные эксперименты.

1. Введение в теорию шифрования.

1.1. Основные понятия теории шифрования

Криптология – наука, изучающая методы защиты информации путем её алгоритмического преобразования. Она подразделяется на два раздела – криптографию и криптоанализ.

Секретность – главное понятие криптографического раздела.

Шифрование – средство достижения секретности информации, состоящее из двух этапов: зашифрования и расшифрования исходных данных.

Зашифрование – процесс криптографического преобразования множества открытых сообщений в множество закрытых сообщений.

Расшифрование – процесс криптографического преобразования закрытых сообщений в открытые.

Дешифрование – это процесс взлома (расшифровки) шифртекста при неизвестном ключе или алгоритме, использующий методы криптоанализа.

Шифртекст – это данные, которые были подвергнуты процедуре зашифрования.

Конфиденциальность – свойство информации быть доступной только для определенного круга пользователей информационной системы, в которой данная информация циркулирует.

Множество открытых сообщений – битовый поток, файл, сетевой фрейм и т.д.

Закрытый ключ – это секретная информация, требуемая для расшифровки сообщений.

Открытый ключ - это открытая информация, требуемая для шифрования сообщений, результат шифрования и криптостойкость зависит от длины ключа.

Криптостойкость – способность алгоритма шифрования противостоять криптоанализу. Она является главным параметром любой криптосистемы, может определяться в следующем виде:

- количество времени;

- количество требуемой информации;
- количество итераций,

  требуемые для взлома секретного ключа[1].

1.2. Односторонние функции

Идея ассиметричных алгоритмов тесно связана с развитием теории односторонних функций, называемых так же необратимыми функциями, до сих пор не доказано существование данных функций, но современное ассиметричное шифрование основывается на предположение, что данный тип функций всё же существуют. Построение односторонних функций является очень сложной задачей.

Односторонней функцией называется функция $F(x): X \to Y, x$ из X, обладающая двумя свойствами:

1. существует полиномиальный алгоритм вычисления значения $y = F(x)$;
2. не существует полиномиального алгоритма инвертирования функции $F(x) = y$.

Полиномиальным называется алгоритм, выполнение которого заканчивается не более чем за $P(n)$ шагов, где n – размер задачи, подаваемой на вход, измеряемая, как правило количеством символов текста, описывающего эту задачу [2].

Односторонние функции являются фундаментом ассиметричной криптографии, многие из их числа являются частью большинства систем интернет-банкинга по всему миру.

1.3. Асимметричные криптосистемы

Ассиметричные алгоритмы шифрования используют пары ключей, один ключ находится в свободном доступе и используется для шифрования текста – открытый ключ, а второй – закрытый, находится у человека, сгенерировавшего парные ключи и используется для расшифровки сообщения.

Криптосистемы с открытым ключом определяется тремя взаимосвязанными алгоритмами: генерация ключей, шифрование и расшифрование. Алгоритм генерации ключей находится в свободном доступе, всякий пользователь может подать ему на вход данные и получить пару ключей[3].

Стойкость данных криптосистем основывается, на алгоритмической трудности решения задачи поиска секретного ключа различными методами.

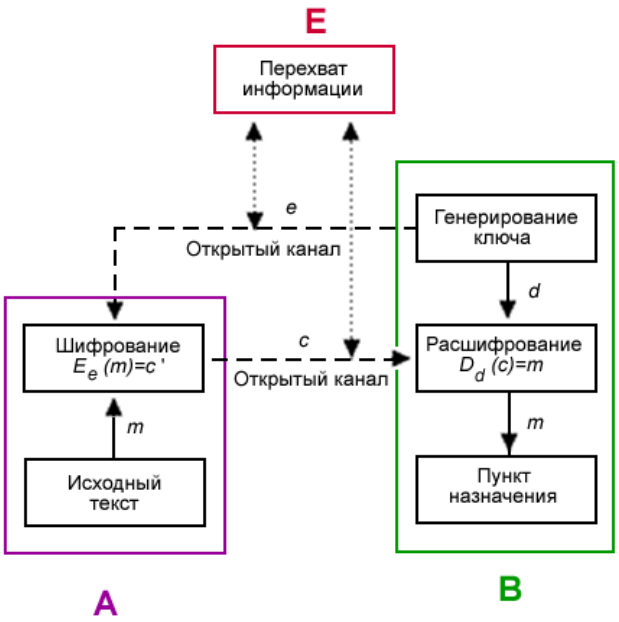

Рисунок 1 – схема движения информации при использовании ассиметричного алгоритма шифрования

1.4. Алгоритм шифрования RSA

Наиболее известно криптосистемой с открытым ключом является алгоритм шифрования RSA, который назван по первым буквам фамилий своих изобретателей – Ривеста (Rivest), Шамира (Samir) и Адлемана (Adleman). RSA – первая практическая реализация криптографии с открытым ключом, основывающаяся на понятии односторонних функций, которое предложили Диффи и Хеллман.

Стойкость алгоритма основывается на сложности факторизации больших простых чисел. Открытый и закрытый ключи являются функциями двух больших простых чисел с высокой разрядностью. Чтобы восстановить закрытый ключ по шифртексту и открытому ключу требуется разложить открытый ключа на два больших простых числа[4].

Для генерации пары ключей нужно выполнить следующий алгоритм[2,4]:

1. Выбрать два случайных простых числа p и q, удовлетворяющих условию |p|≈|q|.
2. Вычислить N=pq.
3. Вычислить функцию Эйлера φ(N)=(p-1)(q-1).
4. Выбрать случайное целое число 1 < e < φ(N), взаимно простое со значение функции Эйлера
5. Вычисляется число d, мультипликативно обратное к числу e по модулю φ(n), то есть число, удовлетворяющее сравнению:

$$e * d = 1(\mathrm{mod}(\varphi(n))$$

6. Ключ {e, N} публикуются в открытом доступе и являются открытым ключом шифрования RSA.
7. Ключ {d,N} используется в качестве закрытого ключа.

После генерации ключей настоятельно рекомендуется тщательно уничтожить числа p, q и φ(N) [4].

Чтобы зашифровать исходное сообщение нужно использовать следующую формулу:

$$C = M^e mod(N)$$

В свою очередь для расшифровки шифр текста и получения исходного текста используется формула:

$$M = E^d mod(N)$$

1.5. Параллельный алгоритм RSA

Так как алгоритм RSA относится к типу асимметричных, то скорость его работы намного ниже, чем у симметричных, но его криптостойкость кратно выше[1].

Основная идея параллельного шифрования заключается в разделении исходного текста на несколько частей и дальнейшем шифровании каждой из частей на узлах при помощи уникального ключа шифрования, где роль узлов выполняют вычислительные машины, входящие в сеть «Отправитель-

Получатель» и имеющие уникальный номер. Реализация данной идеи позволит значительно увеличить скорость шифрования и сделает возможным использование алгоритма RSA для передачи сообщений.

Параллельный алгоритм шифрования RSA:

1. перемешивание изначального текста при помощи SHUFFLE алгоритма;
2. разделение измененного текста на n частей, где n – количество узлов шифрования;
3. генерация n+1 парных ключей для каждого из узлов и одного для вычислительной машины отправителя и дальнейшая передача открытых его частей;
4. распределение текстовых частей по узлам и составление массива данных (SHUFFLE массив), который поможет восстановить исходный порядок измененного текста, последним значением передаётся параметр для обратного SHUFFLE преобразования;
5. шифрование полученных частей текста алгоритмом RSA и шифрование SHUFFLE массива;
6. передача зашифрованных сообщений;
7. дешифрация полученных шифртекстов и восстановление первоначальной информационной структуры.

Распараллеливание шифрования частей текста на нескольких узлах значительно повышает скорость шифрации, а также повышает защищенность передаваемых данных, ведь если злоумышленник получит даже несколько частей текста на одном из двух этапов, то не сможет получить цельного текста, несущего смысл для получателя [по данным автора].

Схематически алгоритм параллельного шифрования представлен на рисунке 2.

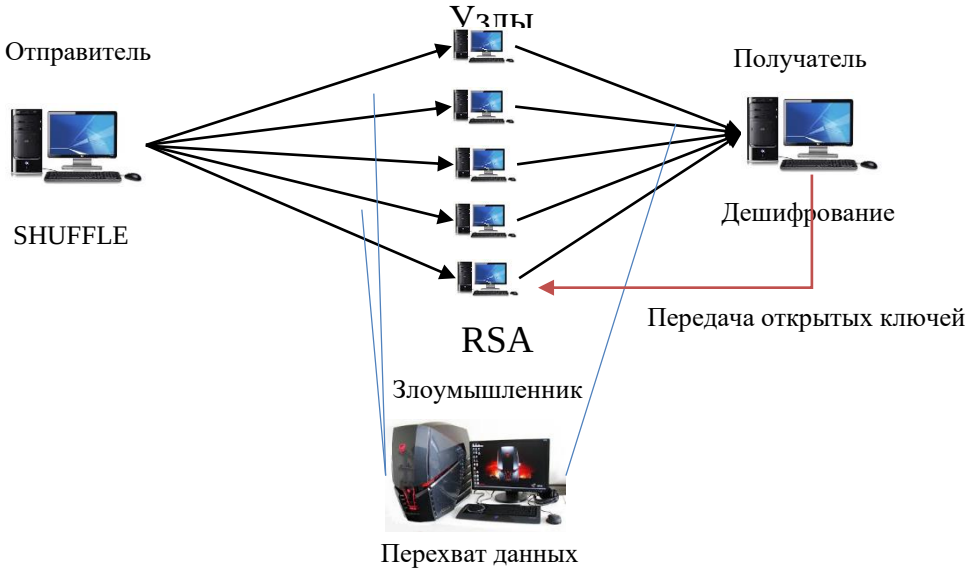

Рисунок 2 - Схема параллельного шифрования данных с возможностью перехвата информации [по данным автора]

При параллельной шифрации данных на n узлах скорость зависит от нескольких факторов:

1. длина текста;
2. используемые язык (английский, русский);
3. длина ключа;
4. количество узлов.

1.6. Идеальный алгоритм перемешивания для криптографии (Perfect Shuffle Algorithm)

Модель данного алгоритма базируется циклических функциях, перемешивания и замены. Функция перемешивания делит текст на две части, а затем равномерно перемешивает их между собой. Схема работы функции представлена на рисунке 3.

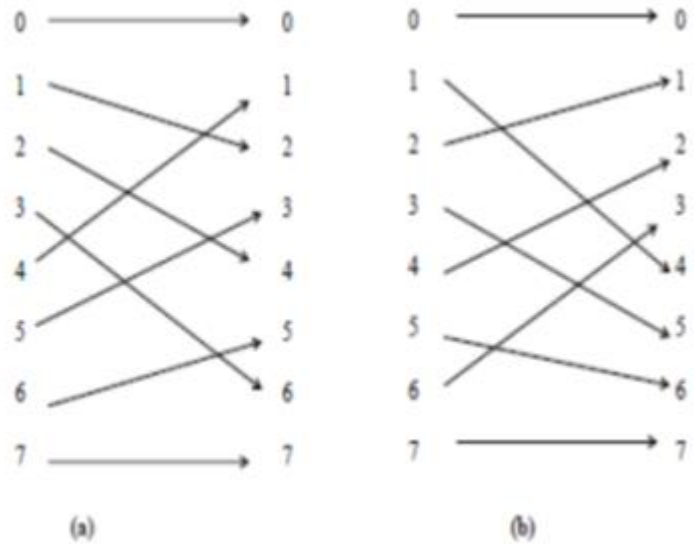

Рисунок 3 - (a) Идеальное перемешивание (b) Обратное идеальное перемешивание[5].

Связь между i-ым и j-ым элементом можно описать следующими уравнениями:

Таблица 1 – Уравнения связи между элементами

$j=2*i$	$0 \le i \le 2^n/2 - 1$
$j=2*1+1-2^n$	в других случаях

Функция замены выполняет перестановку соседних элементов. Для обратного преобразования требуется 2n операций перестановки и замены[5].

Данный алгоритм не производит вычисления выражений, поэтому не требует больших вычислительных мощностей и высокую скорость работы, поэтому идеально подходит для предварительной подготовки текста при применении параллельного алгоритма RSA.

1.7. Метод решения системы, содержащей операцию деления по модулю

Для решения систем вида:

$$x = r1mod(a1) \quad (1)$$
$$x = r2mod(a2)$$

существует метод, основанный на китайской теореме об остатках. Формула, для решения имеет следующий вид:

$$x = \sum_{i=1}^{n} r_i * M_i * M_i^{-1} mod(M) \quad (2)$$

Где:

$$M = \prod a_i \quad (3)$$

$$M_i = \frac{M}{a_i} \quad (4)$$

$$M_i^{-1} = \frac{1}{M_i} mod(a_i) \quad (5)$$

1.8. Общий метод решета числового поля

Криптостойкость алгоритма RSA основывается на сложности факторизации чисел. После анализа различных методов факторизации был выбран самый эффективный по скорости, общий метод решета числового поля.

Данный метод можно представить, как обновлённый метод квадратичного решета, он требуется нахождение гладких чисел порядка $n^{1/2}$. Размер данных чисел растёт с ростом разрядности числа. Метод решета числового поля требует поиск гладких чисел субэкспоненциального относительно n размера. Благодаря тому, что данные числа меньше, вероятность отношения чисел к гладким выше, это является основной причиной эффективности данного метода.

Конечная формула криптостойкости для факторизации чисел RSA имеет следующий вид:

$$e^{(c+o(1))k^{\frac{1}{3}}log^{\frac{2}{3}}k}, \quad (6)$$

$$o(1)=\log_2 e,$$

где c<2 (скорость факторизации k-битного числа), а e – часть открытого ключа [6].

2. Практическая часть

2.1. Программная реализация алгоритмов

Реализация алгоритмов выполнялась на языке C# в среде разработки Visual Studio. Для сопоставления буквам и символам числового эквивалента использовались значения кодировки UTF-32, после шифрования выполнялось обратное преобразование в Base64String, данная функция при конвертации использует сопоставление чисел с символами ASCII. Это

позволяет производить однозначное преобразование для шифрации и дешифрации. В отдельной программе были реализованы SHUFFLE алгоритм и стандарт ассиметричного шифрования RSA с генерацией ключей.

При разработке программного кода в основном использовались встроенные функции библиотек Math, включающая различные математические функции, String, которая содержит функции для работы с данными строкового типа, и класс, позволяющий эффективно работать с алгоритмом RSA, System.Security.Cryptography.RSA.

2.1.1. Реализация алгоритма RSA

Алгоритм реализован в виде консольного приложения с областью для ввода текста и возможностью генерации пар ключей заданной длины, где минимальным порогом является ключ длиной 384 бита. Интерфейс программы представлен на рисунке 4.

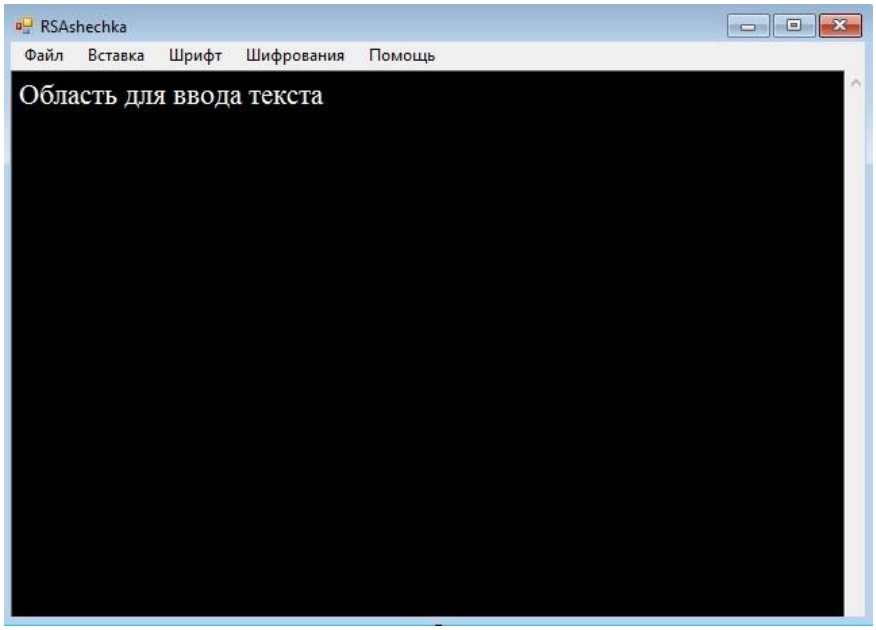

Рисунок 4 - Интерфейс программы шифрования

Первоочередным пунктом в работе программы является генерация ключей, кроме длины ключа другие параметры не задаются, для их выбора используется компьютерный датчик случайных чисел. Сгенерированные ключи сохраняются в отдельные файлы. Открытый ключ передаётся по

открытому каналу к узлу или будущему отправителю сообщения. Закрытый ключ хранится на компьютере получателя для дальнейшей расшифровки.

Для запуска генератора нужно перейти во вкладку «Шифрование-> Генерировать пару ключей» появится диалоговое окно, в котором нужно указать длину ключа.

Рисунок 5 - Интерфейс генератора ключей

Текст для шифрования можно загрузить из файла, вставить из буфера обмена, либо набрать вручную. После выгрузки текста нужно его зашифровать, для этого нужно выбрать пункт «Шифрование-> Зашифровать» и выбрать заранее сгенерированный ключ.

После шифрования будет выведено диалоговое окно со временем, затраченным на шифровку, а в области для ввода текста появится шифртекст, который можно либо скопировать в буфер обмена, либо сохранить в текстовый файл, для дальнейшей передачи.

В будущем планируется автоматизация данного процесса.

Программный код представлен в приложении А.

2.1.2. Реализация SHUFFLE алгоритма

Для применения алгоритма перемешивания к тексту нужно вставить его из буфера обмена или загрузить из файла. Далее нажать на «Файл-> SHUFFLE», появится диалоговое окно, в котором нужно указать количество итераций смешивания и количество разбиений текста. После выполнения данных действий создадутся n+1 файлов, где n-число разбиений.

Данный алгоритм имеет очень большую скорость, поэтому в дальнейшем время на его выполнение учитываться не будет.

2.2. Подготовка данных

В качестве данных для проведения тестирования были использованы четыре текста, два на русском и два на английском, в различной стилистике их основные параметры представлены в таблице 2.

Таблица 2 – Параметры тестовых текстов

Тип текста	Кол-во слов	Кол-во символов	Размер
Литературный (рус.)	1400	8760	35040 байт
Технический (рус.)	1387	8772	35088 байт
Литературный (англ.)	1600	8674	34696 байт
Технический (англ.)	1570	8671	34684 байт

Для шифрования было сгенерировано 32 пары ключей, 16 из них имеют длину 1024 бита, а 16 2048 бит. Ключи большей длины не рассматривались, потому что на сегодняшний день максимальная длина расшифрованного RSA ключа равна 700 битам, на его расшифровку потребовалось около 11 месяцев совместной работы 300-400 вычислительных машин (2007 год). Так как текст переводится в 32-битную кодировку, то при кодировании более точно сохраняется его структура, а также возможность передачи различных символов, входящих в данную кодировку.

Таблица 3 – Числовое обозначение заглавных букв английского алфавита

A	00000041	N	0000004e
B	00000042	O	0000004f
C	00000043	P	00000050
D	00000044	Q	00000051
E	00000045	R	00000052
F	00000046	S	00000053
G	00000047	T	00000054
H	00000048	U	00000055
I	00000049	V	00000056
J	0000004	W	00000057
K	0000004b	X	00000058
L	0000004c	Y	00000059
M	0000004d	Z	0000005a

Перевод текста в числа выполняется программно. На этом подготовку данных будем считать завершенными.

2.3.Использование параллельного алгоритма шифрования и анализ эффективности

2.3.1.Иллюстративный пример шифрования текста

Для шифрования алгоритмом RSA будем использовать ранее разработанную программу. Для демонстрации работоспособности программы зашифруем литературный текст длиной 131 символ.

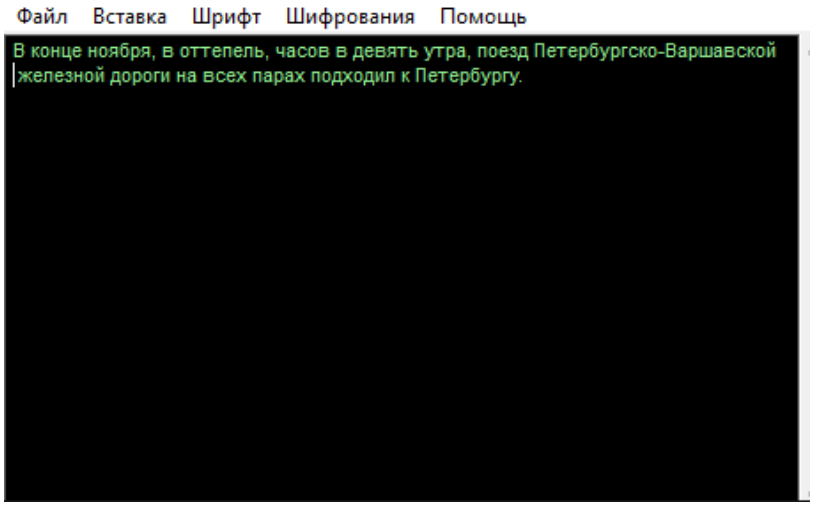

Рисунок 6 – Заполнение текста для шифровки

Производим шифрование при помощи ключа длиной 1024 бита.

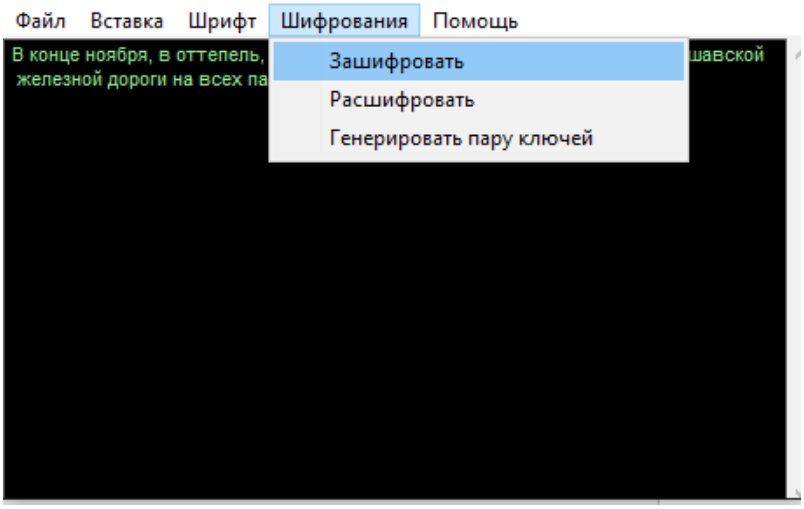

Рисунок 7 – Запуск алгоритма шифрования

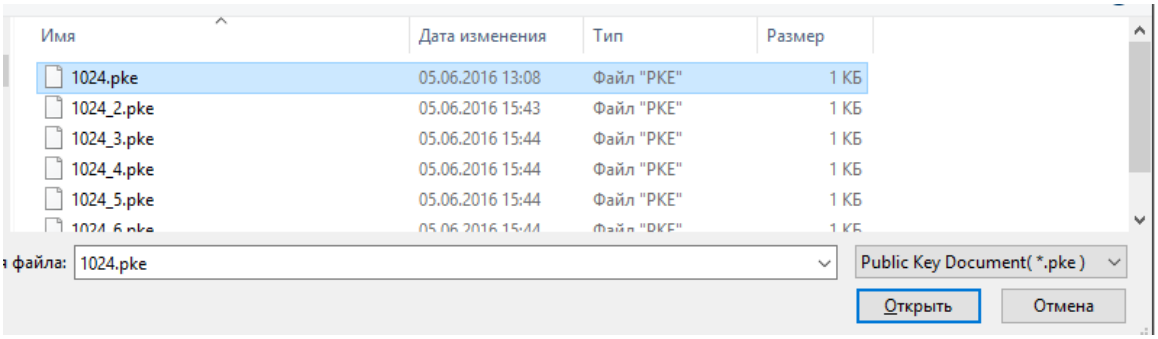

Рисунок 8 – Выбор открытого ключа

После выбора открытого ключа шифрования будет произведено шифрование и выведено время, затраченное на шифровку

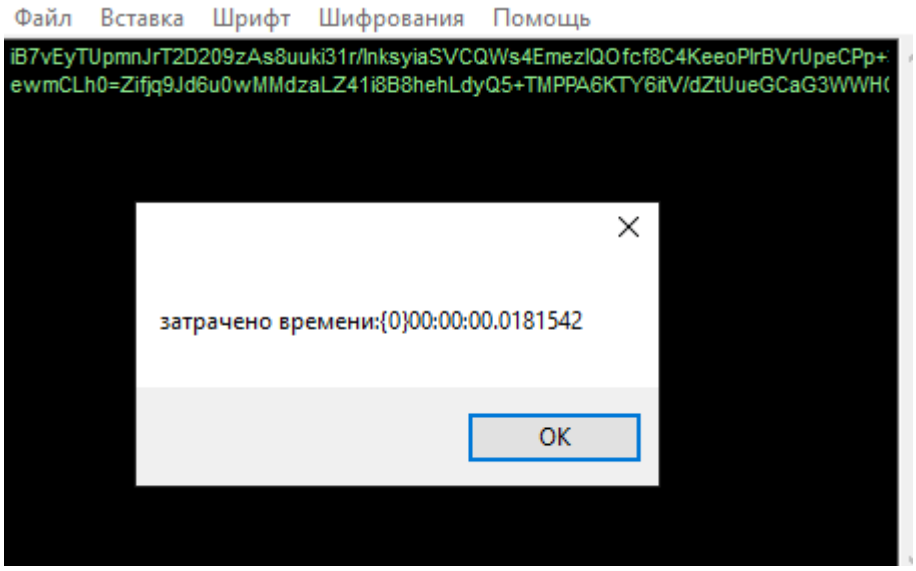

Рисунок 9.1 – Результат шифрования

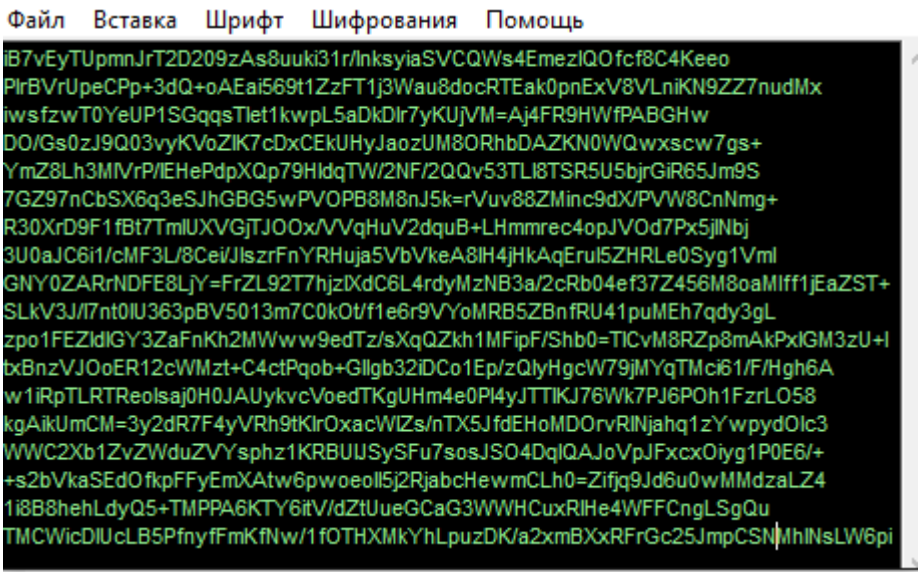

Рисунок 9.2 – Результат шифрования

После того как мы получили шифртекст, нужно проверить, правильно ли он зашифрован и получит ли получатель требуемый текст.

Для этого выбираем «Шифрование-> Расшифровать» и выбираем парный закрытый ключ. Итог выполнения дешифрации представлен на рисунке 10.

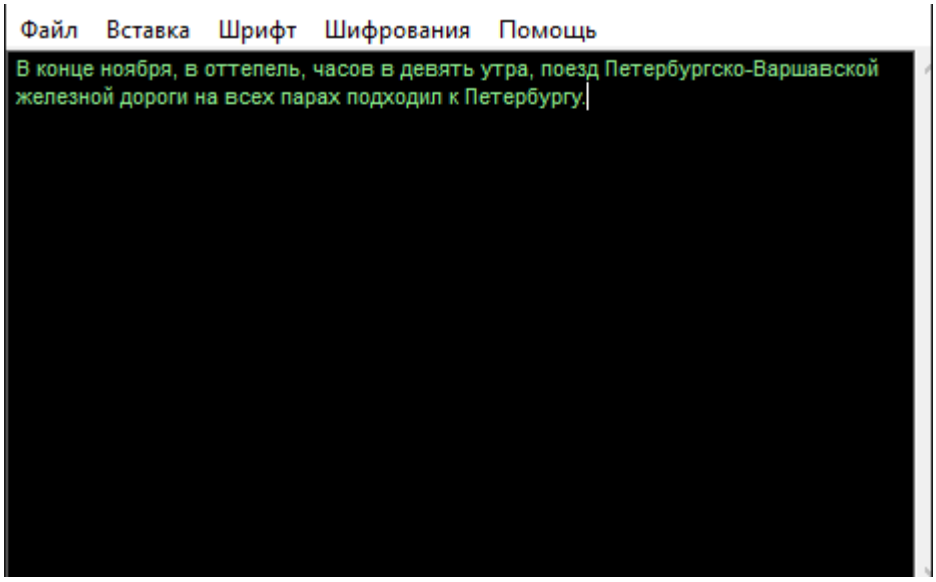

Рисунок 10 – Дешифрованный текст

Из данного примера видно, что шифрация и дешифровка выполняются корректно.

2.3.2.Анализ скорости и криптостойкости параллельного алгоритма RSA

Были проанализированы тексты различного содержания, но одинаковой длины, в ходе анализа выявлено, что скорость шифрования на узлах зависит от мощности вычислительной машины, длины ключа и используемого языка, поэтому рассматривались только 2 литературных текста: на русском и английском языке. Результаты изменения скорости от изменения количества узлов представлены в приложении В.

Анализ полученных результатов показала, что русский текст шифруется на 3-8% быстрее (0,0003-0,003). Вычисления производились на компьютере с 4-х ядерным процессором с тактовой частотой 3.4 Ггц. На компьютерах с различными вычислительными мощностями данные показатели могут немного разниться.

По полученным данным были построены графики, представленные на рисунках 11-12.

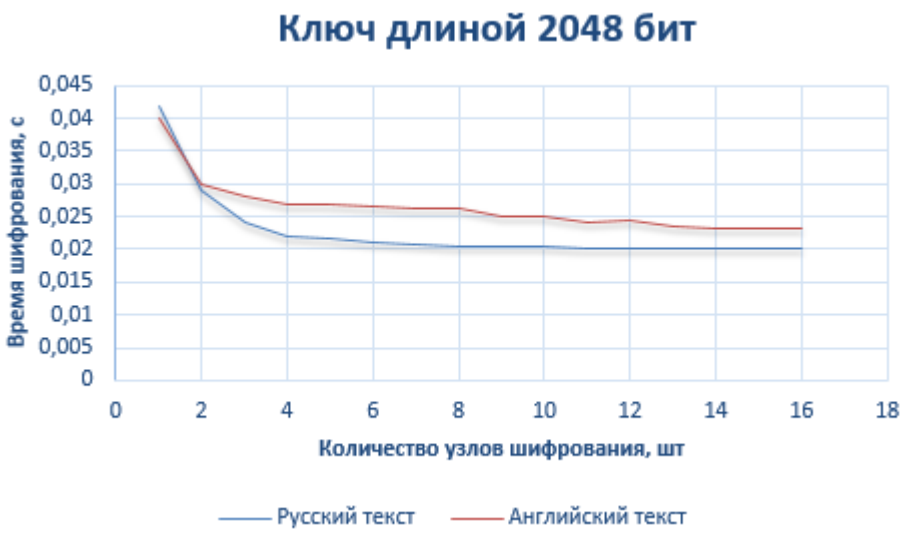

Рисунок 11 – Изменение скорости шифровки, при изменении количества узлов шифрования (2048 битный ключ)

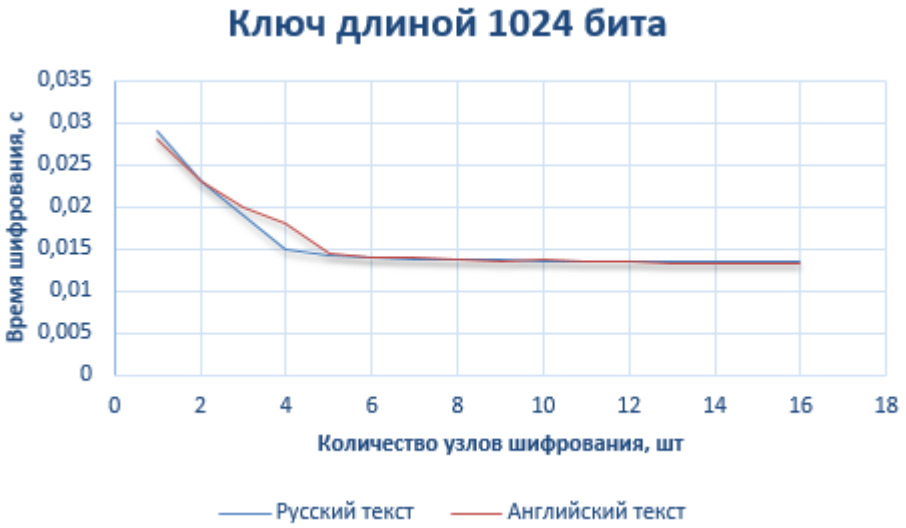

Рисунок 12 - Изменение скорости шифровки, при изменении количества узлов шифрования (1024 битный ключ)

Анализ результатов, представленных в приложении В и рисунках 11-12, показывает, что оптимальным количеством узлов для эффективного ускорения шифрования по данному методу является 3-5 узлов, которые ускорят вычисления на 42.5-52% относительно скорости шифрования на 1 машине. Дальнейшее увеличение узлов не придаст весомого ускорения и может быть использовано лишь для повышения криптостойкости. Защита будет повышаться прямо пропорционально увеличению количества узлов,

ведь будут появляться новые парные ключи, которые так же нужно будет взламывать.

Основной недостаток алгоритма RSA – медленная скорость шифрования.

Анализ результатов помог выявить оптимальный размер частей исходного текста для максимального ускорения параллельного алгоритма шифрования RSA. Оптимальным будет их разбиение на части длиной 1700-2600 символов.

Реализованная схема так же повышает криптостойкость в n раз и исключает полный перехват информации, что почти на 100% защищает пользователя, так же если будет произведена подмена пакетов данных, то получатель сможет легко найти брешь в защите, ведь он будет знать номер узла, в котором произошла подмена данных.

В связи с постоянным ростом вычислительных мощностей для шифрации использовался ключи длиной 1024-2048 бит, что равно 307-617–значному числу. На сегодняшний день ключ длиной 2048 бит является криптоустойчивым, для его взлома по общему методу решета числового поля потребуется 2.042e8 лет.

Заключение

Рассмотрен стандарт ассиметричного шифрования RSA, а также его основные достоинства и недостатки.

Разработан алгоритм параллельного шифрования, произведена его программная реализация, проведено тестирование работоспособности и сделаны замеры основного показателя реализованного алгоритма.

Предложенный алгоритм имеет применение при передаче новостных текстов, секретных документов, предназначенных для закрытого круга лиц. Из-за отсутствия эффективного метода для факторизации чисел длиной большей, чем 230 разрядов, сгенерированные единожды ключи могут использоваться на протяжении нескольких лет, а может и десятилетий. Также в любой момент можно произвести замену ключей, включенных в систему.

Не стоит забывать, что прогресс не стоит на месте и в любой момент может быть разработан способ факторизации высокоразрядных чисел, но наш алгоритм предоставляет защиту даже от этого случая, ведь он включает в себя 3 уровня защиты.

Список использованных источников

1. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. – М.:ДМК, 2000. – 448 с.
2. Онацкий А. В., Йона Л.Г. Асимметричные методы шифрования. – Модуль 2 Криптографические методы защиты информации в телекоммуникационных системах и сетях: Учеб. Пособие/ Под ред. Н.В. Захарченко – Одесса: ОНАС им. А.С Попова, 2010 – 148с.
3. Баричев, С.Г. Основы современной криптографии. – М.: 2001. – 120 с.
4. Мао, Венбо. Современная криптография: теория и практика. : Пер. с англ. – М. : Издательский дом «Вильямс», 2005. – 768 с.
5. Jalan Margonda. Perfect Shuffle Algorithm for Cripthography // ARPN Journal of Engineering and Applied Sciences.-2014-V.9, №12, p. 2384-2386
6. Pomerance, Carl. "A Tale of Two Sieves", Notices of the AMS-December 1996, T. 43 (12), p. 1473–1485

Приложение А

(Обязательное)

Программный код шифрования и расшифровки данных

Таблица 4 – Таблица основных идентификаторов

object inputObject	входной текст
string encryptedString	зашифрованная строка
string decryptedString	расшифрованная строка
int keySize	длина ключа в байтах
string InputString	Входная строка для шифрования
int dwKeySize	длина ключа в битах
byte[] bytes	исходный текст в числовых обозначениях
int maxLength	ограничитель длины шифртекста
int dataLength	количество входных символов
int iterations	количество итераций шифрования

```
public class Encryption \\ код шифрования
	{
		private ContainerControl containerControl = null;
		private Delegate finishedProcessDelegate = null;
		private Delegate updateTextDelegate = null;

		public void Encrypt( object inputObject )
		{
			object[] inputObjects = ( object[] )inputObject;
			containerControl = ( Form ) inputObjects[ 0 ];
			finishedProcessDelegate = ( Delegate ) inputObjects[ 1 ];
			updateTextDelegate = ( Delegate )inputObjects[ 2 ];
			string encryptedString = EncryptString( ( string )inputObjects[
3 ], ( int )inputObjects[ 4 ], ( string )inputObjects[ 5 ] );
			containerControl.Invoke( updateTextDelegate, new object[] {
encryptedString } );
			containerControl.Invoke( finishedProcessDelegate );

		}


		public void Decrypt( object inputObject ) \\ код дешифрования
		{
			object[] inputObjects = ( object[] )inputObject;
			containerControl = ( Form )inputObjects[ 0 ];
			finishedProcessDelegate = ( Delegate )inputObjects[ 1 ];
			updateTextDelegate = ( Delegate )inputObjects[ 2 ];
			string decryptedString = DecryptString( ( string )inputObjects[
3 ], ( int )inputObjects[ 4 ], ( string )inputObjects[ 5 ] );
			containerControl.Invoke( updateTextDelegate, new object[] {
decryptedString } );
			containerControl.Invoke( finishedProcessDelegate );
		}


		public string EncryptString( string inputString, int dwKeySize,
string xmlString ) \\ шифрование строки
		{
			RSACryptoServiceProvider rsaCryptoServiceProvider = new
RSACryptoServiceProvider( dwKeySize );
```

```
rsaCryptoServiceProvider.FromXmlString( xmlString );
                    int keySize = dwKeySize / 8;
                    byte[] bytes = Encoding.UTF32.GetBytes( inputString );

                    int maxLength = keySize - 42;
                    int dataLength = bytes.Length;
                    int iterations = dataLength / maxLength;
                    StringBuilder stringBuilder = new StringBuilder();
                    for( int i = 0; i <= iterations; i++ )
                    {
                            byte[] tempBytes = new byte[ ( dataLength - maxLength *
i > maxLength ) ? maxLength : dataLength - maxLength * i ];
                            Buffer.BlockCopy( bytes, maxLength * i, tempBytes, 0,
tempBytes.Length );
                            byte[] encryptedBytes =
rsaCryptoServiceProvider.Encrypt( tempBytes, true );

                            Array.Reverse( encryptedBytes );

                            stringBuilder.Append( Convert.ToBase64String(
encryptedBytes ) );
                    }
                    return stringBuilder.ToString();
            }

            public string DecryptString( string inputString, int dwKeySize,
string xmlString ) \\ дешифрование строки
            {
                                    RSACryptoServiceProvider
rsaCryptoServiceProvider = new RSACryptoServiceProvider( dwKeySize );
                    rsaCryptoServiceProvider.FromXmlString( xmlString );
                    int base64BlockSize = ( ( dwKeySize / 8 ) % 3 != 0 ) ? ( ( (
dwKeySize / 8 ) / 3 ) * 4 ) + 4 : ( ( dwKeySize / 8 ) / 3 ) * 4;
                    int iterations = inputString.Length / base64BlockSize;
                    ArrayList arrayList = new ArrayList();
                    for( int i = 0; i < iterations; i++ )
                    {
                            byte[] encryptedBytes = Convert.FromBase64String(
inputString.Substring( base64BlockSize * i, base64BlockSize ) );

.Reverse( encryptedBytes );
                            arrayList.AddRange( rsaCryptoServiceProvider.Decrypt(
encryptedBytes, true ) );
                    }
                    return Encoding.UTF32.GetString( arrayList.ToArray(
Type.GetType( "System.Byte" ) ) as byte[] );
            }
      }
                                          }
```

Приложение В

(Обязательное)

Скорость шифрования при различном количестве узлов

	Кол-во узлов	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Кол-во символов (рус.)	8760	4380	2920	2190	1752	1460	1251	1095	973	876	796	730	674	626	584	548
	Кол-во символов (англ.)	8674	4337	2891	2169	1735	1446	1239	1084	964	867	789	723	667	620	578	542
2048 бит	Скорость шифрования(рус.) 10^{-3} с	42	29	24	22	21,8	21,1	20,8	20,6	20,5	20,4	20,3	20,3	20,1	20,1	20,2	20,1
	Скорость шифрования(англ.) 10^{-3} с	40	30	28	27	26,8	26,4	25,1	25,1	24,2	24,3	23,6	23,3	23,2	23,3	23,2	23,1
1024 бит	Скорость шифрования(рус.) 10^{-3} с	29	23	19	15	14,3	13,9	13,8	13,8	13,7	13,6	13,6	13,6	13,5	13,6	13,5	13,5
	Скорость шифрования(англ.) 10^{-3} с	28	23	20	18	14,5	14	13,9	13,7	13,6	13,7	13,6	13,3	13,4	13,2	13,2	13,1