Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

РЕАЛИЗАЦИЯ ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ

(АЛГОРИТМ RSA)

Пояснительная записка

к курсовому проекту по дисциплине

« Основы алгоритмизации и программирования»

КП 52493.011101.081 ПЗ

Руководитель (С.А Апанасевич)

Учащийся гр. 52493 (Д. С. Ермакович)

Минск 2017

**Содержание**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

КП 52493.011101.081

Разраб.

Ермакович

Провер.

Апанасевич

Реценз.

Н. Контр.

Утверд.

Реализация электронной цифровой подписи (алгоритм RSA)

Лит.

Листов

41

МРК

Введение 4

1 Постановка 5

1.1 Описание предметной области 5

1.2 Обзор существующих решений 6

1.3 Входные и выходные данные 8

2 Проектирование задачи 9

2.1 Диаграмма вариантов использования 9

2.2 Разработка алгоритма работы программы 10

2.3 Разработка алгоритмов решения задачи 11

3 Программная реализация 12

3.1 Выбор и обоснование инструментов разработки 12

3.2 Структура программы 13

3.3 Описание разработанных процедур и функций 14

3.4 Назначение и условие применения программы 15

4 Системные требования 16

4.1 Руководство пользователя 17

4.2 Тестирование 20

Заключение 23

Список использованной литературы 24

Приложение А 25

Приложение B………………………………………………………………………………………...............26

# Введение

Электронно-цифровая подпись (ЭЦП) — электронный аналог собственноручной подписи — используемый в системах электронного документооборота для придания электронному документу юридической силы, равной бумажному документу, подписанного собственноручной подписью правомочного лица и/или скрепленного печатью. Документ (файл), подписанный ЭЦП, гарантированно защищен от изменений — проверка подписи мгновенно выявит расхождение. ЭЦП обеспечивает проверку целостности документов, конфиденциальность, установление лица, отправившего документ. Это позволяет усовершенствовать процедуру подготовки, доставки, учета и хранения документов, гарантировать их достоверность. Главное преимущество использование ЭЦП — значительное сокращение временных и финансовых затрат на оформление и обмен документацией. Таким образом, по функциональности ЭЦП даже превосходит обычную подпись. Предположим, что две стороны (назовем их условно «А» и «Б») решили организовать между собой обмен документами на машинных носителях. Как должен выстраиваться документооборот между этими сторонами? В первую очередь, стороны должны договориться об использовании средств ЭЦП. Лучше, если это будет программный или программно-аппаратный комплекс, сертифицированный в нашей стране. После того, как средство ЭЦП выбрано, стороны должны выполнить генерирование ключей — по открытому и секретному (личному) ключу ЭЦП для каждой стороны.

Из исходного текста документа и его ЭЦП формируется электронный документ. Здесь нужно понимать, что электронный документ — это не просто файл на магнитном носителе с текстом документа, а файл, состоящий из двух частей: общей (в которой содержится текст) и особенной, содержащей все необходимые ЭЦП (рис. 3). Текст документа без ЭЦП — это не более чем обычный текст, который не имеет юридической силы. Его можно распечатать, передать по электронной почте, отредактировать, но нельзя установить его подлинность. ЭЦП без текста документа вообще представляет собой непереводимую игру букв. Восстановить документ по ЭЦП невозможно точно так же, как невозможно восстановить дворец по найденному кирпичу. ЭЦП сама по себе не имеет ни ценности, ни смысла.

Примерами алгоритмов ЭЦП могут служить DSA, RSA, Схема Эль-Гамаля, FDH, Схема Шнорра.

Задачи можно разделить на:

— Разработка непосредственно алгоритма создания и проверки ЭЦП.

— Создание интерфейса взаимодействия с пользователем.

Целью моего курсового проекта была реализация алгоритма электронной цифровой подписи алгоритмом RSA на языке Delphi.

# 1 Постановка

## 1.1 Описание предметной области

Тема проекта – “Реализация электронной цифровой подписи (алгоритм RSA)”. Необходимо разработать программу, которая будет добавлять подпись к txt документу и создавать новый файл с результатом, а также проверять передаваемый файл с цифровой подписью, используя вводимый пользователем открытый ключ.Суть электронной подписи заключается в том, чтобы получатель с помощью открытого ключа и цифровой подписи мог проверить, изменен ли документ кем - либо еще в процессе доставки, кроме создателя электронной подписи. Для подделки электронной подписи, нужно подобрать такой же секретный ключ, с помощью которого отправитель создавал электронную подпись. А далее с помощью этого секретного ключа повторно сгенерировать электронную подпись.

Алгоритм создания открытого и секретного ключей:

1. Выбираются два различных случайных простых p и q заданного размера
2. Вычисляется их произведение n = p \* q, которое называется модулем.
3. Вычисляется значение функции Эйлера от числа n:
4. Выбирается целое число e (1 < e <   взаимно простое со значением .
5. Вычисляется число d, мультипликативно обратное к числу e по модулю , т.е. число, удовлетворяющее сравнению . Число d – секретная экспонента.
6. Пары {e;n} и {d;n} – открытый и закрытый ключи RSA

Алгоритм создания ЭЦП:

1. Взять число m (обычное число, либо хеш целевого документа).
2. Сформировать открытый и закрытый ключи: e, d, n.
3. Создать цифровую подпись s с помощью секретного ключа {d;n}:
4. Передать пару {m;s}, состоящую из сообщения и подписи, а также пару открытого ключа {e;n} для проверки подписи.

Алгоритм проверки ЭЦП:

1. Принять пару {m;s} и получить соответствующий открытый ключ {e;n}.
2. Вычислить прообраз сообщения по формуле .
3. Сравнить m и m’. Если они равны, то m и подпись не изменена.

Поскольку подписываемые документы — переменного (и как правило достаточно большого) объёма, в схемах ЭП зачастую подпись ставится не на сам документ, а на его хэш. Для вычисления хэша используются криптографические хэш-функции, что гарантирует выявление изменений документа при проверке подписи.

## 1.2 Обзор существующих решений

В настоящее время существует несколько схем построения цифровой подписи:

* На основе алгоритмов симметричного шифрования. Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.
* На основе алгоритмов асимметричного шифрования. На данный момент такие схемы ЭП наиболее распространены и находят широкое применение.

Кроме этого, существуют другие разновидности цифровых подписей (групповая подпись, неоспоримая подпись, доверенная подпись), которые являются модификациями описанных выше схем. Их появление обусловлено разнообразием задач, решаемых с помощью ЭП.

Симметричные схемы ЭП. Они менее распространены, чем асимметричные, так как после появления концепции цифровой подписи не удалось реализовать эффективные алгоритмы подписи, основанные на известных в то время симметричных шифрах.

Симметричные схемы основаны на хорошо изученных блочных шифрах. В связи с этим симметричные схемы имеют следующие преимущества:

* Стойкость симметричных схем ЭП вытекает из стойкости используемых блочных шифров, надежность которых также хорошо изучена.
* Если стойкость шифра окажется недостаточной, его легко можно будет заменить на более стойкий с минимальными изменениями в реализации.
* Однако у симметричных ЭП есть и ряд недостатков:
* Нужно подписывать отдельно каждый бит передаваемой информации, что приводит к значительному увеличению подписи. Подпись может превосходить сообщение по размеру на два порядка.
* Сгенерированные для подписи ключи могут быть использованы только один раз, так как после подписывания раскрывается половина секретного ключа.

Асимметричные схемы ЭП. Они относятся к криптосистемам с открытым ключом. В отличие от асимметричных алгоритмов шифрования, в которых шифрование производится с помощью открытого ключа, а расшифровка — с помощью закрытого, в асимметричных схемах цифровой подписи подписание производится с применением закрытого ключа, а проверка подписи — с применением открытого.

Общепризнанная схема цифровой подписи охватывает три процесса:

* Генерация ключевой пары. При помощи алгоритма генерации ключа равновероятным образом из набора возможных закрытых ключей выбирается закрытый ключ, вычисляется соответствующий ему открытый ключ.
* Формирование подписи. Для заданного электронного документа с помощью закрытого ключа вычисляется подпись.
* Проверка (верификация) подписи. Для данных документа и подписи с помощью открытого ключа определяется действительность подписи.
* Для того, чтобы использование цифровой подписи имело смысл, необходимо выполнение двух условий:
* Верификация подписи должна производиться открытым ключом, соответствующим именно тому закрытому ключу, который использовался при подписании.
* Без обладания закрытым ключом должно быть вычислительно сложно создать легитимную цифровую подпись.

Схемы электронной подписи могут быть одноразовыми и многоразовыми. В одноразовых схемах после проверки подлинности подписи необходимо провести замену ключей, в многоразовых схемах это делать не требуется.

Перечень алгоритмов ЭП. Асимметричные схемы:

* FDH (Full Domain Hash), вероятностная схема RSA-PSS (Probabilistic Signature Scheme), схемы стандарта PKCS#1 и другие схемы, основанные на алгоритме RSA;
* Схема Эль-Гамаля;
* американские стандарты электронной цифровой подписи: DSA, ECDSA (DSA на основе аппарата эллиптических кривых);
* российские стандарты электронной цифровой подписи: ГОСТ Р 34.10-94 (в настоящее время не действует), ГОСТ Р 34.10-2001 (не рекомендован к использованию после 31 декабря 2017 года), ГОСТ Р 34.10-2012
* схема Шнорра;
* pointcheval-Stern signature algorithm;
* вероятностная схема подписи Рабина;
* схема BLS (Boneh-Lynn-Shacham);
* схема GMR (Goldwasser-Micali-Rivest);

Также алгоритмы ЭП делятся на детерминированные и вероятностные.

Детерминированные ЭП при одинаковых входных данных вычисляют одинаковую подпись.

## 1.3 Входные и выходные данные

Вариант А. Создание электронной подписи.

В качестве входных данных поступает файл.

Файл должен иметь расширение txt и содержать некоторый текст.

Выходные данные:

Копия входного файла с записанной электронной подписью в конце.

Файл closedKey.txt, содержащий секретный ключ ЭЦП.

Данный файл необходимо спрятать, ибо доступ к нему позволит злоумышленникам изменить документ и переделать электронную цифровую подпись.

Файл openedKey.txt, содержащий открытый ключ ЭЦП.

Пояснение. Электронная подпись имеет форматирование {s1;s2}, либо {s;0}, либо {;s1;s2} ({;s;0}).

Первый вариант используется, если хеш документа по модулю > 200000, это значит, что даже при оптимизированном возведении степень по модулю возможно переполнение типа int64 в Delphi.

Тогда хеш разделяется на 2 части и для каждой формируется часть подписи.

Второй вариант используется, если хеш документа по модулю <= 200000. Тогда подпись создается для всего хеша документа и помещается в первую ячейку. Вторая часть остается равной 0.

Третий вариант отличается помещением ‘;’ перед частью ЭЦП. Это обозначает, что хеш документа имел отрицательный вид (все действия производятся над модулем хеша).

Вариант Б. Проверка электронной подписи.

В качестве входных данных поступает файл и открытый ключ.

Пояснение. Открытый ключ должен иметь форматирование ‘{e;n}’,

где e – открытая экспонента, а n – модуль.

Для проверки используется та же функция, что и для создания подписи, только с другими данными. Я ее реализовал через быстрое возведение в степень по модулю.

Выходные данные:

Информация о подлинности документа.

3 вещи должны быть неизменны для положительного результата:

* открытый ключ;
* электронная цифровая подпись;
* сам документ;

После всех проверок на экран выводится информация о подлинности документа.

# 2 Проектирование задачи

## 2.1 Диаграмма вариантов использования

Оператор

Открыть файл

Создание ЭЦП

Вывод результатов работы

Рисунок 2.1 – Диаграмма вариантов использования программы.

Проверка документа

Сначала пользователь в любом случае открывает файл, с которым он хочет далее взаимодействовать.

Необходимость открыть файл проистекает из того факта, что нужен целевой документ, для которого будет создана электронная цифровая подпись.

Затем пользователь совершает одно из далее перечисленных действий:

– Применяет создание ЭЦП для файла, с созданием файлов с ключами и парой документ – подпись.

– Вводит Открытый ключ и проверяет подлинность документа.

– Совершает выход из программы.

Стандартный алгоритм работы – сначала создание подписи, затем проверка у получателя.

В конце цепочки действий выводится сообщение с результатом работы программы.

## 2.2 Разработка алгоритма работы программы

Первым этапом является открытие пользователем в меню MainMenu необходимого ему функционала: «Создать подпись»(1) либо «Проверить подпись»(2).

После этого на экране отобразятся кнопки в зависимости от решения:

при выборе 1 варианта, отобразится кнопка «Открыть файл», реализованная через MainMenu, а также кнопка «Создать эл. подпись»;

при выборе 2 варианта, отобразится кнопка «Открыть файл», поле для ввода Открытого ключа, а также кнопка «Проверить эл. подпись»;

Далее пользователь в любом случае открывает нужный файл (оригинал либо копия с ЭЦП), используя диалоговое окно OpenDialog.

После этого в первом случае пользователь нажимает кнопку «Создать эл. подпись».

Во втором случае пользователю нужно ввести открытый ключ в поле для ввода, сформированный и записанный в файл заранее, при создании ЭЦП. Далее пользователь должен нажать кнопку «Проверить эл. подпись».

В любом случае на экран выводится результат работы программы. В первом случае это сообщение об успехе и вывод пути до нового файла, а во втором это информация о подлинности документа.

Алгоритм работы программы состоит из событий, которые активируются по действию пользователя.

В проект входит 1 форма – Form1 и 1 модуль – Unit1.

Форма Form1 состоит из следующих элементов:

* Кнопка Button1 – «Создать эл. подпись»;
* Кнопка Button2 – «Проверить эл. подпись»;
* Кнопка Button3 – «Открыть файл»;
* Поле для ввода Edit2 – ввод Открытого ключа;
* Метки Label1 и Label3 – отображение пути выбранного файла;
* MainMenu – для определения начального курса выполнения программы;

Кнопка Button3 напрямую связана с вызовом OpenDialog.

Содержание MainMenu:

* N2 – «Создать подпись»;
* N3 – «Проверить подпись»;
* N4 – «Выход»;

выбор N2 отображает на экране компоненты: Button1, и Button3;

выбор N3 отображает на экране компоненты: Button2, и Button3;

выбор N4 завершает выполнение программы – вызывает процедуру Close();

## 2.3 Разработка алгоритмов решения задачи

Из открытого файла извлекаем весь текст как строку. Данную строку хешируем одним из алгоритмов хеширования.

Алгоритм создания открытого и секретного ключей

* 1. Выбираются два различных случайных простых числа  p и q заданного размера
  2. Вычисляется их произведение n = p \* q, которое называется модулем.
  3. Вычисляется значение функции Эйлера от числа n:
  4. Выбирается целое число e (1 < e <   взаимно простое со значением функции – открытая экспонента.
  5. Вычисляется число d, мультипликативно обратное к числу e по модулю , т.е. число, удовлетворяющее сравнению . Число d – секретная экспонента.
  6. Пара {e;n} – открытый ключ RSA, нужна для проверки подписи.
  7. Пара {d;n} – закрытый ключ RSA, нужна для создания ЭЦП, держится в секрете.

Для генерации простых чисел p и q я использую решето Эратосфена, позволяющее быстро сгенерировать массив простых чисел заданного размера, и затем из массива случайно выбираются 2 числа. N = p \* q. Выбирается число e, взаимно простое с функцией Эйлера от n.

По расширенному алгоритму Евклида находим множитель перед e. Из данного множителя путем преобразований находим закрытую экспоненту, мультипликативно обратную выбранной открытой экспоненте e.

( и

d = (A mod ), где A – множитель перед A в расширенном алгоритме Евклида.

Алгоритм создания ЭЦП:

* 1. Взять число m (обычное число, либо хеш целевого документа).
  2. Сформировать открытый и закрытый ключи: e, d, n.
  3. Создать цифровую подпись s с помощью секретного ключа {d;n}:
  4. Передать пару {m;s}, состоящую из сообщения и подписи, а также пару открытого ключа {e;n} для проверки подписи.

Алгоритм проверки ЭЦП:

1. Принять пару {m;s} и получить соответствующий открытый ключ {e;n}.
2. Вычислить прообраз сообщения по формуле .
3. Сравнить m и m’. Если они равны, то m и подпись не изменена.

Проверка подписи осуществляется всего одной формулой.

# 3 Программная реализация

## 3.1 Выбор и обоснование инструментов разработки

КП написан на языке программирования Delphi. Среда разработки – BorlandDelphi 7.

BorlandDelphi 7, выпущенная в августе 2002 года, стала стандартом для многих разработчиков Delphi.

Используя среду разработки BorlandDelphi 7 можно быстро и эффективно создавать, как простые, так и сложные программы. BorlandDelphi 7 - среда предназначена для быстрой (RAD) разработки прикладного ПО для операционных систем Windows. Выделение и освобождение памяти контролируется в основном пользовательским кодом, что, с одной стороны, ужесточает требования к качеству кода, а с другой — делает возможным создание сложных приложений с высокими требованиями к отзывчивости (работа в реальном времени).

Прежде всего Delphi предназначен для профессиональных разработчиков, желающих очень быстро разрабатывать приложения в архитектуре клиент-сервер. Delphi производит небольшие по размерам высокоэффективные исполняемые модули (.exe и .dll), поэтому в Delphi должны быть, прежде всего, заинтересованы те, кто разрабатывает продукты на продажу. С другой стороны, небольшие по размерам и быстро исполняемые модули означают, что требования к клиентским рабочим местам существенно снижаются – это имеет немаловажное значение и для конечных пользователей.

Преимущества Delphi по сравнению с аналогичными программными продуктами:

* Быстрота разработки приложения (RAD);
* Высокая производительность разработанного приложения;
* Низкие требования разработанного приложения к ресурсам компьютера;
* Наращиваемость за счет встраивания новых компонент и инструментов в среду Delphi;

– удачная проработка иерархии объектов.

Система программирования Delphi рассчитана на программирование различных приложений и предоставляет большое количество компонентов для этого. К тому же работодателей интересует, прежде всего, скорость и качество создания программ, а эти характеристики может обеспечить только среда визуального проектирования, способная взять на себя значительные объемы рутинной работы по подготовке приложений, а также согласовать деятельность группы постановщиков, кодировщиков, тестеров и технических писателей. Возможности Delphi полностью отвечают подобным требованиям и подходят для создания систем любой сложности.

## 3.2 Структура программы

Программа состоит из двух основных частей. Первая отвечает за ЭЦП, а вторая за проверку подписи в документе по открытому ключу и вывод результата.

Первая часть состоит из двух компонентов: кнопки открытия файла с документом, которая задействует openDialog, и кнопки «Создать эл. подпись», которая запускает всю цепочку создания ЭЦП.

Используется всего 1 форма Form1.

Button1Click – нажатие на кнопку «Создать эл. подпись». Содержит в себе процедуры и функции:

1. SDBM – хеш-функция от текста документа
2. createSign – формирование данных для создания ЭЦП.
3. формирования ЭЦП - s
4. создание файлов и вывод результата (!)

Вышеназванная createSign содержит в себе вызов еще двух процедур:

– reshErat – создание решета Эратосфена для формирования массива простых чисел до заданного числа. Размер массива – 30. Из данного массива мы берем случайные 2 числа, которые должны быть 1 размера. Произведение этих случайных чисел даст нам модуль – число n. А функция Эйлера от n легко считается как (p-1)(q-1), где p и q – наши выбранные случайные числа.

– generateKeys – выбор простых чисел p и q, создание n, функция Эйлера от n (eiler), выбор открытого ключа e, нод e и eiler, создание закрытого ключа dexp. Также содержит в себе вызов gcd\_ext – нод 2х чисел и смежные множители для вычисления dexp.

Вторая часть состоит из трех компонентов: кнопки открытия файла с документом, которая задействует openDialog, поля для ввода открытого ключа и кнопки «Проверить эл. подпись», которая запускает алгоритм проверки подписи в выбранном файле по введенному открытому ключу.

Алгоритм проверки подписи:

Button2Click. нажатие на кнопку «Проверить эл. подпись». Содержит в себе процедуры и функции:

1. parsePair – обрабатывает строку ЭЦП в конце файла.
2. SDBM – хеш-функция от текста документа.
3. Еще раз parsePair – обрабатывает строку открытого ключа из поля ввода.
4. проверка ЭЦП по функции и сравнение с хэш-функцией документа

Также в меню существует элемент «Выход», который просто завершает работу программы. Логика написана в N4Click и состоит лишь из 1 процедуры Close.

## 3.3 Описание разработанных процедур и функций

Процедуры и функции:

procedure reshErat(maxEl : integer; var B : SimpleArr); - Создание решета Эратосфена (нахождение всех простых чисел) до числа maxEl. Затем с конца этими значениями заполняется и возвращается B.

function ToBin(x: integer): string; - Быстрый перевод из integer в двочную строку string (только для положительных).

function modexp(x,y,n:int64):Int64; - функция возведения в степень по модулю (рекурсия) (x^y mod n) x – основание, y – степень, n – модуль.

function SDBM(mess: string): integer; - хеширование строки методом SDBM (х32) с хорошим распределением по всем битам.

procedure gcd\_ext(a, b: integer; var x, y, gcd: integer); - НОД чисел по расширенному алгоритму Евклида (рекурсия). x – множитель для a, y – множитель для b, gcd – НОД чисел.

function pow(base, up: int64): int64; - функция возведения в степень для положительных чисел и степеней. base – основание, up – степень

procedure generateKeys(minimalN: integer; var n : Int64; var eexp : integer; var dexp : integer; B : SimpleArr); n – модуль, произведение случайных чисел p и q, eiler – функция Эйлера от n, minimalN – минимальный N, который может быть (он должен быть не меньше хеш-функции входящего сообщения), eexp, dexp – открытый и закрытый ключи.

procedure parsePair(str : string; var one : Int64; var two : int64; var neg : boolean); - парсит пары вида [one;two], извлекая из них one и two. Если neg = True, то [;one;two] – указывает на то, что изначальный hashCode был отрицателен, записывается в neg.

procedure createRSA(); - метод цепочки создания электронной цифровой подписи.

procedure checkDoc(); - метод цепочки проверки электронной цифровой подписи.

procedure chooseFile(); - Выбор файла из директорий ОС Windows

procedure TForm1.Button1Click(Sender: TObject); - нажатие на кнопку, создание электронно-цифровой подписи RSA для выбранного файла.

procedure TForm1.Button2Click(Sender: TObject); - нажатие на кнопку, проверка электронно-цифровой подписи RSA для выбранного файла.

procedure TForm1.Button3Click(Sender: TObject); - выбор файла из OpenDialog.

Кнопки меню:

procedure TForm1.N2Click(Sender: TObject); - создать подпись

procedure TForm1.N3Click(Sender: TObject); - проверить эл. подпись

procedure TForm1.N4Click(Sender: TObject); - Выход

## 3.4 Назначение и условие применения программы

Программа предназначена для создания электронной цифровой подписи для документов, а также проверки подписи на уже сгенерированном файле с парой исходного документа и электронной цифровой подписи ( m;s ).

Пользователю следует использовать текстовые файлы, т.к. информация записывается внутрь документа, и созданный документ иного формата может быть поврежден записью ЭЦП и стать более недоступным для открытия. Также исходный файл не должен быть пустым.

После генерации электронной цифровой подписи секретный ключ, который хранится в созданном файле следует держать секретно во избежание изменения передаваемого документа и подделки электронной цифровой подписи.

Открытый ключ, сохраняемый в файл, стоит передать получающей стороне. Он используется для проверки электронной цифровой подписи.

Если изменить в созданном файле с парой исходного документа и электронной цифровой подписи ( m;s ) исходный текст либо электронную цифровую подпись, то проверка документа по открытому ключу даст несовпадение.

Также произойдет, если открытый ключ будет не верен, но изменять только открытый ключ нет необходимости даже злоумышленникам.

3 вещи должны быть неизменны:

* открытый ключ
* электронная цифровая подпись
* сам документ

Единственный возможный вариант подделки электронной цифровой подписи это перехватить прием получателем открытого ключа, изменение файла, генерирование новой собственной электронной подписи по новому секретному и открытому ключу и отправка нового документа с новой электронной цифровой подписью вместо старого, а также доставить новый открытый ключ вместо старого.

Поэтому открытый ключ следует передавать отдельно от документа с подписью, ибо если злоумышленник получит и открытый ключ, и документ с подписью, а открытый ключ при этом не дойдет до получателя, то подделка подписи возможно. Все упирается в получение проверяющей стороной нового открытого ключа.

Таким образом, открытый ключ хоть и может храниться открыто вместе с документом, но необходимо, чтобы принимающая сторона получила именно истинный открытый ключ, тогда вне зависимости от перехвата документа и подделки, открытый ключ не совпадет и подделка будет вскрыта.

# 4 Системные требования

Таблица 4.1 – Системные требования.

|  |  |
| --- | --- |
| Компонент | Требование |
| ПК и процессор | Процессор с тактовой частотой 500 МГц или выше; |
| Память | Не менее 256 МБ оперативной памяти; |
| Жесткий диск | 10МБ свободного дискового пространства. |
| Монитор | Монитор с разрешением 1024 х 768 или больше. |
| Операционная система | Windows XP и выше. |
| Дополнительно | Требования и функциональные возможности продуктов могут различаться в зависимости от конфигурации системы и используемой операционной системы. |

Системные требования — это описание примерных характеристик, которым должен соответствовать компьютер для того, чтобы на нём могло использоваться какое-либо определённое программное обеспечение. Эти характеристики могут описывать требования как к аппаратному обеспечению (тип и частота процессора, объём оперативной памяти, объём жёсткого диска), так и к программному окружению (операционная система, наличие установленных системных компонентов и сервисов и т. п.). Обычно такие требования составляются производителем или автором ПО.

Для некоторого ПО различают минимальные и рекомендуемые системные требования:

Минимальные системные требования — это набор условий, необходимых для возможности запуска и работы программного продукта. Однако, наличие минимальных системных требований не отменяет возможность запуска ПО на компьютерах, которые по характеристикам слабее минимальных.

Рекомендуемые системные требования — набор характеристик, подразумевающих оптимальную работу большей части возможностей продукта. Однако, даже если компьютер и подходит под рекомендуемые системные требования, это не значит высокой производительности ПО, например, в некоторых играх невозможно играть на максимальных настройках графики.

Конфигурация компьютера, в частности характеристики процессора напрямую влияют на скорость выполнения алгоритмов программы.

Программа разработана в среде разработки Borland Delphi, поэтому она должна работать на операционных системах семейства Windows, начиная с Windows XP.

## 4.1 Руководство пользователя

Работа программы делится на 2 функциональные возможности: создание электронной цифровой подписи и проверка ее в документе.

Для начала пользователь должен выбрать в меню модуль, соответствующий цели пользователя.

Сначала опишу действия при создании электронной цифровой подписи.

При открытии программы пользователь видит главный экран приложения (Рисунок 4.1), сначала пустой. Пользователь выбирает в меню сверху один из 3х вариантов: Создать электронную подпись, Проверить электронную подпись, Выход.

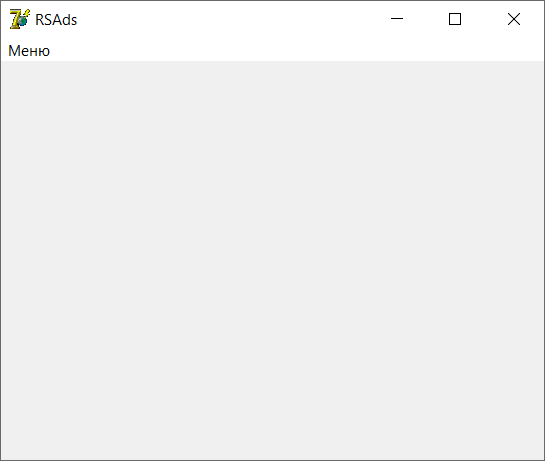


Рисунок 4.1 – начальный экран программы.

После выбора в меню соответсующего элемента, на экране появляются 2 кнопки (Рисунок 4.2). Сначала пользователь должен выбрать исходный файл с текстом, нажав на кнопку Открыть файл.

Сначала опишу как пользователь создает электронную цифровую подпись.

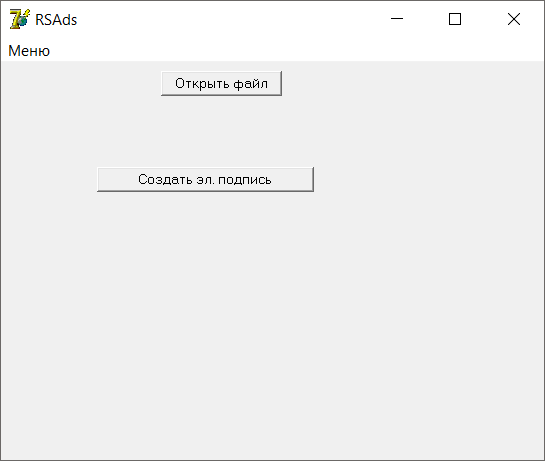


Рисунок 4.2 – экран создания электронной цифровой подписи.

Результат открытия файла – появления текста с путем до файла в операционной системе (Рисунок 4.3).

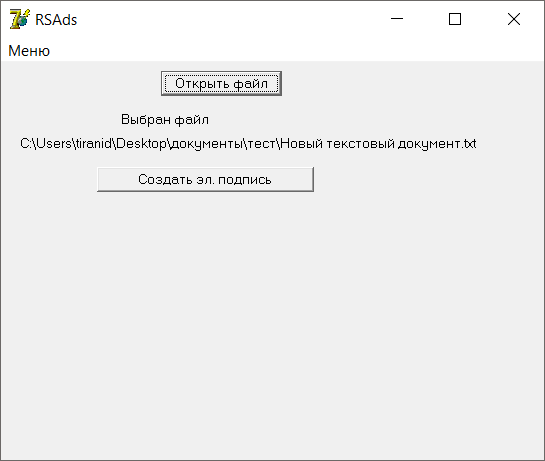


Рисунок 4.3 – Открытие файла.

Далее пользователю нужно лишь нажать на кнопку «Создать электронную подпись».

Результат создания подписи – новый файл с текстом исходного документа и электронной подписью в конце. (Рисунок 4.4).

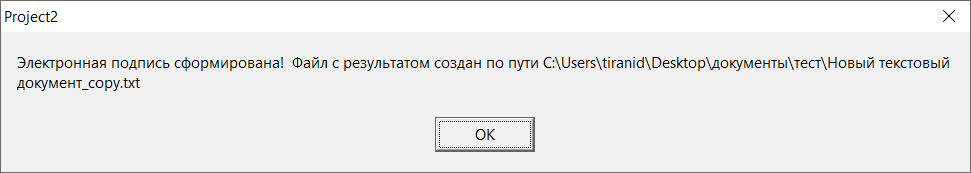


Рисунок 4.4 – Результат создания электронной цифровой подписи.

Помимо файла создаются 2 файла – openedKey.txt и closedKey.txt, содержащие открытый и закрытый ключи. Для проверки документа нам необходим открытый ключ (Рисунок 4.5).

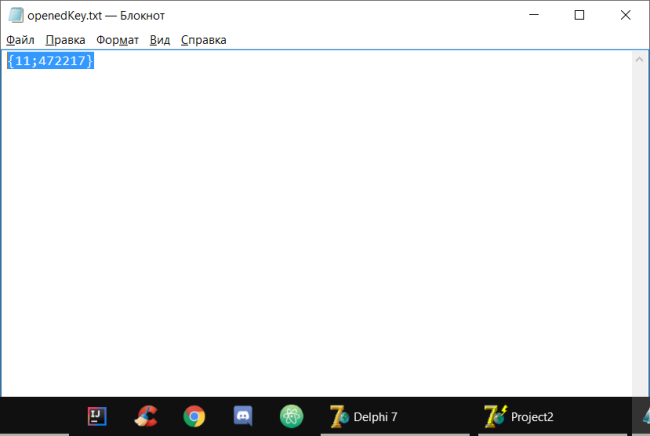


Рисунок 4.5 – содержимое файла openedKey.txt – открытый ключ.

Следующий сценарий – проверка электронной цифровой подписи документа.

Пользователь должен выбрать файл с текстом и электронной подписью, а также ввести открытый ключ, сохраненный при генерации в файл openedKey.txt (Рисунок 4.6)

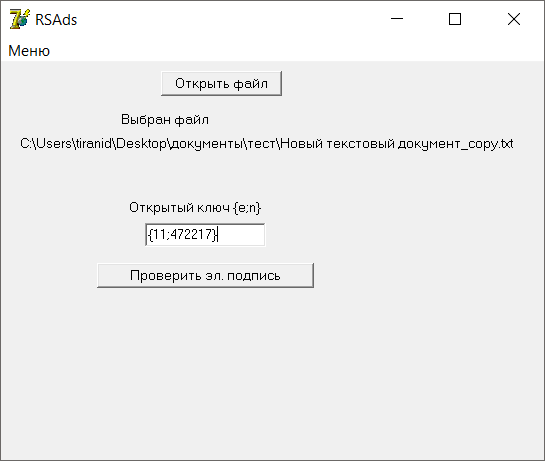


Рисунок 4.6 – Проверка электронной цифровой подписи.

После ввода всех данных и нажатия кнопки «Проверить электронную подпись», отображается результат – сообщение о подлинности документа. (Рисунок 4.7).

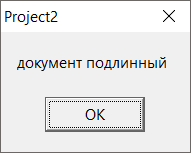


Рисунок 4.7 – Информация о подлинности документа.

Для выхода из программы можно использовать 3 элемент меню – «Выход».

## 4.2 Тестирование

Существовали проблемы с переполнением типов: даже при оптимизированном возведении в степень по модулю, при хешах более 200000 при попытки возвести в куб могло возникнуть переполнение типа int64, что ломало всю логику функций создания и проверки электронной цифровой подписи RSA.

Так как я использую 32 – битную хеш-функцию SDBM (выбор сделан из-за скорости работы и хорошего распределения по всем битам), то неизбежны значения хеша более 200000. Нужно было как либо избегать переполнения типов, работая с большими числами.

Для решения этой задачи я решил разбить входящее число хеш-функции на 2 части. Я разделял число на 2 части (первую и вторую половину слова), и работал отдельно с каждой из них, формирую промежуточные электронные цифровые подписи.

Если применить определенную формулу к полученным формулам, они легко преобразуются в нормальную, полную электронную цифровую подпись, что и происходит у меня при проверке.

Суть: из полученных кусков электронной цифровой подписи генерируем с помощью открытого ключа прообразы, которые потенциально равны изначальным 2 частям хеша.

Далее довольно легко по формуле сделать полный прообраз и сравнить с хешем документа.

Разбиение при некоторых условиях хеш-функции документа на 2 примерно равные части.

При этом потенциально большая по длине часть будет правой. При значениях хеш-функции меньше 200000 по модулю часть будет лишь одна.

// разбиваем большие хеши на 2

if (HashNumber > 200000) then

begin

// разбить на 2 части пополам

strHashCode:= IntToStr(HashNumber);

nextIndex:= (Length(strHashCode) div 2) + 1;

// 1 до nextIndex-1

partOneStr := copy(strHashCode, 1, nextIndex-1);

partOne:= StrToInt(partOneStr);

// от nextIndex до конца

partTwoStr := copy(strHashCode, nextIndex, Length(strHashCode)-nextIndex+1);

partTwo:= StrToInt(partTwoStr);

end;

Формирование электронной цифровой подписи из 2 кусков хеш-функции документа. Подпись также состоит из 2 частей, и если хеш-функция по модулю меньше 200000, то правая часть электронной цифровой подписи будет равняться 0.

// формируем цифровую подпись

s1:= modexp(partOne, dexp, n);

s2:= modexp(partTwo, dexp, n);

str := '{';

if (isHashNegative) then

str := '{;';

// формируем пару ключей {dexp;n}

str := str+IntToStr(s1)+';'+IntToStr(s2)+'}';

Проверка подлинности документа: решение проблемы с отрицательной хеш-функцией а также 2 частями s:

// осуществляем проверку

if (e = 0) then

Exit;

proizv1 := modexp(hashPart1, e, n);

proizv2 := modexp(hashPart2, e, n);

hashLen := Length(IntToStr(hashCode));

if (hashCode < 0) then

dec(hashLen);

h1 := proizv1 \* pow(10, hashLen - Length(IntToStr(proizv1)));

h2 := proizv2;

hsum := h1 + h2;

if (neg) then

hsum := -hsum;

if (hsum = hashCode) then

ShowMessage('документ подлинный')

else

ShowMessage('документ не подлинный');

Для создания простых случайных чисел заданной длины p и q я использовал решето Эратосфена, как метод быстрого создания большого количества простых чисел.

Из этого массива простых чисел случайно выбираются 2 до те пор, пока они не будут одной длины в двоичном представлении.

Так как в ходе алгоритма приходится возводить в огромные степени и делить по модулью, в качестве оптимизации я использую быстрое возведение в степень по модулю (функция modexp).

Используя данную функцию, исходное число возводится максимум в куб, в то время как если бы мы по очереди возводили в степень, а затем делили, то пришлось бы использовать BigDecimal и это в целом неэффективно.

Далее необходимо было создать число d на основе функции Эйлера от n и открытой экспоненты e.

D – закрытая экспонента, используется для генерации электронной цифровой подписи документа.

Число e в свою очередь будет использовано для проверки электронной цифровой подписи документа.

Для поиска мультипликативно обратного по модулю n числа секретной экспоненты d я использую расширенный алгоритм Евклида для открытой экспоненты e и функции Эйлера от n. Множитель перед e проходит через некоторые преобразования и в результате образуется секретная экспонента d.

dexp := (aa mod eiler + eiler) mod eiler;

где dexp – секретная экспонента

aa – множитель перед e в расширенном алгоритме Евклида

eiler – функция Эйлера от n.

Для извлечения значений из сгенерированного документа с парой (m, s), а также передаваемой строки открытого ключа я использую собственный обработчик строки parseString, написанный для извлечения значений из пар {;x;y} и {x;y}

Электронная цифровая подпись может иметь в себе знак “;” сразу после “{“.

Это обозначает, что исходный хеш документа отрицателен, и играет важную роль в проверке подписи.

Без этого хеш -200 и прообраз 200 выдавали бы подтверждение подписи, т.е. сравнивались бы по модулю, ибо внутренние преобразования RSA должны проводиться над положительными числами.

# Заключение

В ходе выполнения курсовой работы представленная программа была выполнена и работала, поставленные задачи были выполнены.

Программа может использоваться на практике, но ее безопасность будет намного ниже, чем промышленные реализации алгоритма RSA, потому что в данной работе стояло ограничение в 64 бита в типе int64, в то время как стандартный RSA имеет 1024 битные (и выше) значения.

Еще одно отличие - программа была написана на языке Delphi.

Язык Delphi работал без наладок, был прост в обращении, возникающие ошибке при программировании легко исправлялись.

Delphi зарекомендовал себя с лучшей стороны, и он рекомендуется как для начинающих программистов, так и для профессионалов.

Благодаря Delphi создания интерфейса приложения не составило труда и заняло очень мало времени.

Программа отлично подойдет для обучения и знакомства с понятием электронной цифровой подписи.

В программе реализованы все функции создания и проверки электронной цифровой подписи.

Это рабочий прототип.

Также в ходе разработки программы изучено несколько хеш-функций, выбор был остановлен на SDBM;

Также были реализованы следующие функции:

* поиск мультипликативно обратного числа по модулю, используя расширенный алгоритм Евклида;
* алгоритм построения решета Эратосфена для генерирования большого количества случайных простых чисел;
* алгоритм обработки строк.

Электронные цифровые подписи используются повсеместно для самых различных целей. Алгоритм RSA используется также и для шифрования данных, использую ту же самую формулу, как и при создании и проверке электронной подписи, только ключи поменялись местами.

RSA является очень быстрым и криптостойким алгоритмом.

# Список использованной литературы

1. Мао, Венбо, Современная криптография: Теория и практика, Вильямс, 2005 - 768 стр.
2. Молдовян, Н.А. Теоретический минимум и алгоритмы цифровой подписи СПб, 2010 - 304 с.
3. Шнайер, Брюс. Практическая криптография, Вильямс, 2005 - 424 с.
4. Бобровский, С. И. Delphi 7 : учеб.курс / С. И. Бобровский. – СПб. : Питер, 2004. – 736 с.
5. Стивенс, Р. Delphi. Готовые алгоритмы, ДКМ Пресс, 2001 – 379 с.

# Приложение А

Оператор

Открыть файл

Создание ЭЦП

Вывод результатов работы

Рисунок 2.1 – Диаграмма вариантов использования программы.

Проверка документа

# Приложение B

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Menus;

type

TForm1 = class(TForm)

Button1: TButton;

Edit2: TEdit;

Label2: TLabel;

Button2: TButton;

OpenDialog1: TOpenDialog;

Button3: TButton;

Label1: TLabel;

Label3: TLabel;

MainMenu1: TMainMenu;

N1: TMenuItem;

N2: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure N4Click(Sender: TObject);

procedure N2Click(Sender: TObject);

procedure N3Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

uses

Math;

type

SimpleArr = array [1..30] of integer;

var

pathToOriginalFile : string;

{$R \*.dfm}

{

Создание решета Эратосфена (нахождение всех простых чисел) до числа maxEl.

Затем с конца этими значениями заполняется и возвращается B

}

procedure reshErat(maxEl : integer; var B : SimpleArr);

var

A: array of boolean;

n, x, y: integer;

begin

SetLength(A, maxEl);

A[1] := false;

n := maxEl;

for x := 2 to n do

A[x] := true;

for x := 2 to n div 2 do

for y := 2 to n div x do

A[x \* y] := false;

y := 1;

// идем от максимального значения назад, пока не заполним B

x:= n;

while (y <= Length(B)) and (x > 0) do

begin

if A[x] then

begin

B[y] := x;

inc(y);

end;

x := x-1;

end;

end;

{

быстрый перевод из integer в двочную строку string (только для положительных)

}

function ToBin(x: integer): string;

var

res: string;

d: 0..1;

begin

res := '';

while (x <> 0) do

begin

d := x mod 2;

res := IntToStr(d) + res;

x := x div 2;

end;

Result := res;

end;

{

функция возведения в степень по модулю (рекурсия)

(x^y mod n)

x - основание

y - степень

n - модуль

}

function modexp(x,y,n:int64):Int64;

var z, k : int64;

begin

if (y = 0) then

Result:= 1

else

begin

k:= y;

z := modexp(x, k div 2, n);

if ((y mod 2) = 0) then

Result:= (z\*z) mod n

else

Result:= (x\*z\*z) mod n;

end;

end;

// хеширование строки методом SDBM (х32) с хорошим распределением по всем битам

function SDBM(mess: string): integer;

var

i: cardinal;

hash: integer;

begin

hash := 0;

for i := 1 to Length(mess) do

hash := ord(mess[i]) + (hash shl 6) + (hash shl 16) - hash;

Result := hash;

end;

{

НОД чисел по расширенному алгоритму Евклида (рекурсия)

x - множитель для a

y - множитель для b

gcd - НОД чисел

}

procedure gcd\_ext(a, b: integer; var x, y, gcd: integer);

var

x1, y1: integer;

begin

if b = 0 then

begin

gcd := a;

x := 1;

y := 0;

Exit

end;

gcd\_ext(b, a mod b, x1, y1, gcd);

x := y1;

y := x1 - (a div b) \* y1

end;

{

функция возведения в степень для положительных чисел и степеней

base - основание

up - степень

}

function pow(base, up: int64): int64;

var

i: cardinal;

begin

if (base = 1) or (up = 0) then

Result := 1

else

begin

Result := base;

for i := 2 to up do

begin

Result := base \* Result;

if (result < 0) then

begin

break;

end;

end;

end;

end;

{

n - модуль, произведение случайных чисел p и q

eiler - функция Эйлера от n

minimalN - минимальный N, который может быть

(он должен быть не меньше хеш-функции входящего сообщения)

eexp, dexp - открытый и закрытый ключи

}

procedure generateKeys(minimalN: integer; var n : Int64; var eexp : integer; var dexp : integer; B : SimpleArr);

var

aa, bb, nod: integer;

k1, k2, p, q : Cardinal;

eiler : int64;

begin

Randomize();

p := 0;

q := 0;

k1 := 0;

k2 := 1;

// формирование случайных чисел в диапазоне от minimalN до minimalN \* 50

// числа должны быть одного порядка

while ((p = q) or (p = 0) or (q = 0) or (n < minimalN) or (n > minimalN \* 50) or (k1 <> k2)) do

begin

// числа берем из массива с неким диапазоном решета Эратосфена

p := RandomFrom(B);

q := RandomFrom(B);

k1 := Length(toBin(p));

k2 := Length(toBin(q));

if (k1 = k2) then

n := p \* q;

end;

// функция Эйлера от n

eiler := (p - 1) \* (q - 1);

eexp := 1;

nod := 0;

// формирование открытого ключа e,

// пока НОД по расширенному алгоритму Евклида

// этого ключа и функции Эйлера не равен 1 (тогда они взаимно простые)

while (nod <> 1) do

begin

inc(eexp);

gcd\_ext(eexp, eiler, aa, bb, nod);

end;

// секретный ключ d - обратный к открытому ключу e по модулю n

// вычисляется по формуле (A mod E(n) + E(n)) mod E(n),

// где A - множитель в алгоритме перед открытым ключом e,

// а E(n) - функция Эйлера от n

dexp := (aa mod eiler + eiler) mod eiler;

end;

procedure createSign(minimalN: integer; var n : Int64; var eexp : integer; var dexp : integer);

var simpleNumbersArray : SimpleArr;

border : Integer;

begin

// вначале формируем решето Эратосфена, задавая ему максимальный элемент,

// опираясь на minimalN

// (10minN)^(1/2) округлить ~ 3.16minN^(1/2)

border:= Round(Sqrt(minimalN\*10));

// 31 простое число

if (border < 127) then

border := 127;

reshErat(border, simpleNumbersArray);

// формируем n, а также открытый и закрытый ключи

generateKeys(minimalN, n, eexp, dexp, simpleNumbersArray);

end;

{

парсит пары вида [one;two], извлекая из них one и two

если neg = True, то [;one;two] - указывает на то,

что изначальный hashCode был отрицателен, записывается в neg

}

procedure parsePair(str : string; var one : Int64; var two : int64; var neg : boolean);

var i : Cardinal;

Ch: Char;

s1, s2 : string;

index : Integer;

begin

i:= Length(str);

Ch := '0';

s1 := '';

s2 := '';

// парсим с конца

while (Ch <> '}') do

begin

Ch := str[i];

i := i - 1;

end;

if (i = 0) then

begin

one := 0;

two := 0;

neg := False;

end;

Ch := str[i];

i := i - 1;

while (Ch <> ';') do

begin

s2 := Ch + s2;

Ch := str[i];

i := i - 1;

end;

Ch := str[i];

i := i - 1;

while (Ch <> '{') do

begin

// негативный флаг

if (Ch = ';') then

begin

neg := True;

Break;

end;

s1 := ch + s1;

Ch := str[i];

i := i - 1;

end;

// переводы значений из строк в int64

Val(s1, one, index);

if (index <> 0) then

begin

ShowMessage('One не число');

end;

Val(s2, two, index);

if (index <> 0) then

begin

ShowMessage('Two не число');

end;

end;

procedure generateRSA();

var

f : TextFile;

newFilePath, mesage, newMessage, str, partOneStr, partTwoStr, strHashCode : string;

nextIndex, indexOfDot : Cardinal;

s1, s2, dexp, eexp, hashCode, partOne, partTwo : integer;

HashNumber, n : Int64;

TStrList : TStringList;

isHashNegative : Boolean;

begin

if (pathToOriginalFile = '') then

begin

ShowMessage('откройте файл');

exit;

end;

partTwo:= 0;

isHashNegative := False;

TStrList := TStringList.Create;

TStrList.LoadFromFile(pathToOriginalFile);

mesage := TStrList.Text;

TStrList.Free;

hashCode := SDBM(mesage);

if (hashCode < 0) then

begin

// отмечаем, что хеш был отрицательным

isHashNegative:= True;

end;

// наш алгоритм работает с положительным hash

HashNumber:= Abs(hashCode);

partOne:= HashNumber;

partOneStr := IntToStr(HashNumber);

// разбиваем большие хеши на 2

if (HashNumber > 200000) then

begin

// разбить на 2 части пополам

strHashCode:= IntToStr(HashNumber);

nextIndex:= (Length(strHashCode) div 2) + 1;

// 1 до nextIndex-1

partOneStr := copy(strHashCode, 1, nextIndex-1);

partOne:= StrToInt(partOneStr);

// от nextIndex до конца

partTwoStr := copy(strHashCode, nextIndex, Length(strHashCode)-nextIndex+1);

partTwo:= StrToInt(partTwoStr);

end;

// формируем ключи по наибольшему из 2х значений

createSign(Max(partOne, partTwo), n, eexp, dexp);

// openedKey.txt

str := '{' +IntToStr(eexp)+';'+IntToStr(n) + '}';

AssignFile(f, 'openedKey.txt');

Rewrite(f);

Append(f);

Writeln(f, str);

CloseFile(f);

// closedKey.txt

str := '{'+IntToStr(dexp)+';'+IntToStr(n)+'}';

AssignFile(f, 'closedKey.txt');

Rewrite(f);

Append(f);

Writeln(f, str);

CloseFile(f);

// формируем цифровую подпись

s1:= modexp(partOne, dexp, n);

s2:= modexp(partTwo, dexp, n);

str := '{';

if (isHashNegative) then

str := '{;';

// формируем пару ключей {dexp;n}

str := str+IntToStr(s1)+';'+IntToStr(s2)+'}';

// позиция точки (перед расширением файла)

indexOfDot := Pos('.', pathToOriginalFile);

//from, to-from+1 (inclusively)

newFilePath :=

Copy(pathToOriginalFile, 1, indexOfDot-1-1+1)

+ '\_copy'

+ Copy(pathToOriginalFile, indexOfDot, Length(pathToOriginalFile)-indexOfDot+1);

newMessage := mesage + str;

AssignFile(f, newFilePath);

Rewrite(f);

Append(f);

Write(f, newMessage);

CloseFile(f);

ShowMessage('Электронная подпись сформирована! '

+ ' Файл с результатом создан по пути ' + newFilePath);

end;

procedure checkDoc();

var e, n, hashPart1, hashPart2 : int64;

indexer, hashLen : cardinal;

hashCode, h1, h2, hsum, proizv1, proizv2 : integer;

ne, neg : Boolean;

TStrList : TStringList;

mesage, mes, openKeyString : string;

begin

if (pathToOriginalFile = '') then

begin

ShowMessage('откройте файл');

exit;

end;

if (Length(Form1.Edit2.Text) < 1) then

begin

showMessage('Введите открытый ключ');

exit;

end;

openKeyString := Form1.Edit2.Text;

if (openKeyString[Length(openKeyString)] <> '}') or (openKeyString[1] <> '{') then

begin

showMessage('ключ не соответстует размеру');

exit;

end;

hashPart1 := 0;

hashPart2 := 0;

e := 0;

n := 0;

neg := False;

TStrList := TStringList.Create;

TStrList.LoadFromFile(pathToOriginalFile);

mesage := TStrList.Text;

TStrList.Free;

if (mesage = '') then

begin

ShowMessage('файл пуст');

exit;

end;

parsePair(mesage, hashPart1, hashPart2, neg);

indexer := Length(mesage) - (Length(IntToStr(hashPart1)) + Length(IntToStr(hashPart2)) + 4);

if (neg) then

indexer := indexer - 1;

mes := Copy(mesage, 1, indexer-1-1+1);

hashCode := SDBM(mes);

parsePair(openKeyString, e, n, ne);

// осуществляем проверку

if (e = 0) then

Exit;

proizv1 := modexp(hashPart1, e, n);

proizv2 := modexp(hashPart2, e, n);

hashLen := Length(IntToStr(hashCode));

if (hashCode < 0) then

dec(hashLen);

h1 := proizv1 \* pow(10, hashLen - Length(IntToStr(proizv1)));

h2 := proizv2;

hsum := h1 + h2;

if (neg) then

hsum := -hsum;

if (hsum = hashCode) then

ShowMessage('документ подлинный')

else

ShowMessage('документ не подлинный');

end;

procedure chooseFile();

begin

if Form1.OpenDialog1.Execute then

begin

pathToOriginalFile := Form1.OpenDialog1.FileName;

Form1.Label1.Caption:= 'Выбран файл';

Form1.Label3.Caption:= pathToOriginalFile;

end;

end;

{

создание электронно-цифровой подписи RSA для выбранного файла

}

procedure TForm1.Button1Click(Sender: TObject);

begin

generateRSA();

end;

{

проверка электронно-цифровой подписи RSA для выбранного файла

}

procedure TForm1.Button2Click(Sender: TObject);

begin

checkDoc();

end;

{

Выбор файла из OpenDialog

}

procedure TForm1.Button3Click(Sender: TObject);

begin

chooseFile();

end;

// создать подпись

procedure TForm1.N2Click(Sender: TObject);

begin

Button1.Visible:=True;

Button3.Visible:=True;

Button2.Visible:=False;

Label2.Visible:=False;

Edit2.Visible:=False;

end;

// проверить эл. подпись

procedure TForm1.N3Click(Sender: TObject);

begin

Button2.Visible:=True;

Button3.Visible:=True;

Label2.Visible:=True;

Edit2.Visible:=True;

Button1.Visible:=False;

end;

// Выход

procedure TForm1.N4Click(Sender: TObject);

begin

Close;

end;

end.