МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема Информационная система электронно-цифровой подписи

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/311-05 ТЗ-01

Листов 51

Руководитель разработки:

кандидат технических наук, доцент кафедры «Измерительно-вычислительные комплексы», *Шишкин Вадим Викторинович*

« » 2023 г.

Исполнитель:

студент гр. ИСТбд-21

*Александров Данила Вячеславович*

« » 2023 г.

2023

**Оглавление**

[**Аннотация** 4](#_Toc149115790)

[**Техническое задание** 5](#_Toc149115791)

Введение……………………………………………………………………………....6

[1. Функциональные требования 6](#_Toc149115792)

[1.1 Создание электронной цифровой подписи 6](#_Toc149115793)

[1.2. Проверка электронной цифровой подписи 7](#_Toc149115794)

[1.3 Управление ключами 7](#_Toc149115795)

[2. Нефункциональные требования 7](#_Toc149115796)

[2.1 Безопасность 7](#_Toc149115797)

[2.2 Интерфейс 7](#_Toc149115798)

[2.3 Производительность 8](#_Toc149115799)

[3. Требования к архитектуре и технологиям 8](#_Toc149115800)

[4. Требования к тестированию 8](#_Toc149115801)

[5. Требования к программной документации 8](#_Toc149115802)

6. Стадии и этапы разработки……………………………………………………….8

[7. Порядок контроля и приёмки 8](#_Toc149115804)

[**Пояснительная записка** 9](#_Toc149115805)

[Введение 10](#_Toc149115806)

1. Проектная часть……………………………………………………………………11

[1.1 Постановка задачи на разработку приложения 11](#_Toc149115808)

[1.2 Архитектура и алгоритмы 11](#_Toc149115809)

[1.3 Тестирование 15](#_Toc149115810)

[1.3.1 Описание отчёта о тестировании 15](#_Toc149115811)

[1.3.2 Цель тестирования 16](#_Toc149115812)

[1.3.3 Методика тестирования 16](#_Toc149115813)

1.3.4 Проведенные тесты…………………………………………………………..16

[2. Источники использованные при разработке 17](#_Toc149115814)

[**Руководство программиста** 18](#_Toc149115815)

[1 Назначения и условия приминения программы. 19](#_Toc149115816)

1.1 Назначения и функции, выполняемые приложением…………………………19

1.2 Условия необходимые для использования приложения………………………19

2. Характеристики программы……………………………………………………..20

2.1 Характеристики приложения…………………………………………………..20

2.2 Особенности реализации приложения…………………………………………25

3. Обращение к программе…………………………………………………………27

4. Сообщения………………………………………………………………………..32

**Заключение**………………………………………………………………………...33

**Исходные тексты программных модулей**………………………………………34

**Аннотация.**

*Задание на курсовую работу*: разработка программы “Электронно-цифровая подпись”.

*Суть задания*:необходимо разработать систему для возможности электронного подписания документов, которая должна представлять из себя окно с интуитивно понятным интерфейсом и основным функционалом для ЭЦП.

*Ключевые функции системы*:

1. Добавление документа и ключа в виде файлов.
2. Генерация ключей.
3. Сохранение ключей в новый файл.
4. Проверка подписи на подлинность.
5. Подписание документа с последующим его сохранением в новый файл.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема Информационная система электронно-цифровой подписи

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/311-05 ТЗ-01

Листов 51

Исполнитель:

студент гр. ИСТбд-21

*Александров Данила Вячеславович*

« » 2023 г.

2023

**Введение**

Информационная система электронно-цифровой подписи (ЭЦП) представляет собой программное обеспечение, предназначенное для создания, проверки и управления электронными цифровыми подписями. Система должна обеспечивать безопасность и подлинность электронных документов.

Механизм электронной цифровой подписи (ЭЦП) возник как побочный эффект криптографии с открытым ключом. Поэтому, характерное для систем с открытым ключом разделение ключа на 2 части, на секретную и несекретную, позволяет реализовать возможность проверки подлинности без возможности подписать другой документ.

Итак, цифровая подпись это конечная цифровая последовательность, зависящая от самого сообщения или документа и от секретного ключа, известного только подписывающему субъекту, предназначенная для установления авторства.

Так как цифровая подпись строится на базе криптосистемы с открытым ключом, то необходимо иметь пару ключей - секретный и открытый. Секретный ключ используется для формирования цифровой подписи, поэтому его нужно хранить в тайне. А открытый ключ используется для проверки соответствия подписи документу, поэтому он должен быть опубликован.

**Техническое задание:**

**1. Функциональные требования**

**1.1 Создание электронной цифровой подписи**

- Пользователь должен иметь возможность создать электронную цифровую подпись для выбранного электронного документа.

- Система должна использовать алгоритмы шифрования для генерации уникальной подписи на основе приватного ключа пользователя.

- Сгенерированная подпись должна быть сохранена вместе с исходным документом.

**1.2 Проверка электронной цифровой подписи**

- Пользователь должен иметь возможность проверить подлинность электронного документа с помощью электронной цифровой подписи.

- Система должна использовать алгоритмы шифрования для проверки подписи на основе публичного ключа пользователя.

- Результат проверки должен быть отображен пользователю, указывая, является ли подпись действительной или недействительной.

**1.3 Управление ключами**

- Система должна предоставлять возможность генерации и хранения пары ключей (приватный и публичный) для каждого пользователя.

- Пользователь должен иметь возможность сохранить и загрузить свои ключи для использования в системе.

- Система должна обеспечивать безопасное хранение приватных ключей, например, с помощью шифрования.

**2. Нефункциональные требования**

**2.1 Безопасность**

- Система должна обеспечивать высокий уровень безопасности для защиты приватных ключей и электронных документов.

- Должны использоваться современные алгоритмы шифрования для генерации и проверки электронной цифровой подписи.

**2.2 Интерфейс**

- Пользовательский интерфейс должен быть интуитивно понятным и удобным в использовании.

- Система должна предоставлять информативные сообщения об ошибках и статусе операций.

**2.3 Производительность**

- Система должна обеспечивать высокую скорость генерации и проверки электронной цифровой подписи, чтобы минимизировать время ожидания пользователя.

**3. Требования к архитектуре и технологиям**

- Система должна быть разработана на языке программирования, поддерживающем работу с криптографическими алгоритмами, а именно C++.

- Должен использоваться современный криптографический алгоритм, а именно RSA.

**4. Требования к тестированию**

- Система должна быть протестирована на соответствие функциональным и нефункциональным требованиям.

- Должны быть проведены тесты на безопасность для проверки уязвимостей системы.

**5. Требования к программной документации**

- Определяются заданием на курсовую работу

**6. Стадии и этапы разработки**

-Определяются заданием на курсовую работу.

**7. Порядок контроля и приёмки**

-Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема**  Информационная система электронно-цифровой подписи

**Пояснительная записка**

Р.02069337.22/311-05 ПЗ-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 51

**Исполнитель**:

студент гр. ИСТбд-21

*Александров Данила Вячеславович*

« » 2023 г.

2023

**Введение**

Информационная система электронно-цифровой подписи (ЭЦП) представляет собой программное обеспечение, предназначенное для создания, проверки и управления электронными цифровыми подписями. Система должна обеспечивать безопасность и подлинность электронных документов.

При разработки программы был выбран такой криптографический алгоритм, как RSA.

"Электронно-цифровая подпись с алгоритмом шифрования RSA" - это приложение, которое позволяет пользователям создавать электронные подписи для документов и сообщений с помощью криптографического алгоритма RSA. RSA - это асимметричный алгоритм шифрования, который использует пару ключей: открытый и закрытый.

Приложение предоставляет возможность создания пары ключей, хранения их в безопасности и использования для создания электронной подписи. Пользователи могут выбрать документ или сообщение, которое нужно подписать, и использовать свой закрытый ключ для создания уникальной электронной подписи. Получатели могут проверить подпись, используя открытый ключ отправителя, чтобы убедиться в том, что документ не был изменен после подписания.

Приложение обеспечивает высокий уровень безопасности благодаря использованию алгоритма RSA, который является одним из наиболее надежных криптографических алгоритмов. Кроме того, приложение имеет простой и интуитивно понятный интерфейс, что делает его доступным для широкого круга пользователей.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании.

**1.2 Архитектура и алгоритмы**

Структура проекта представлена на рис.1:

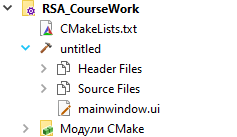
****

Рис.1 Структура проекта

Файл CMakeLists.txt представляет собой обычный текстовый файл, каждая строка которого содержит команду для утилиты CMAKE.

Файл mainwindow.ui отвечает за раздел для разработки дизайна и внешнего вид основного окна приложения(рис.2):

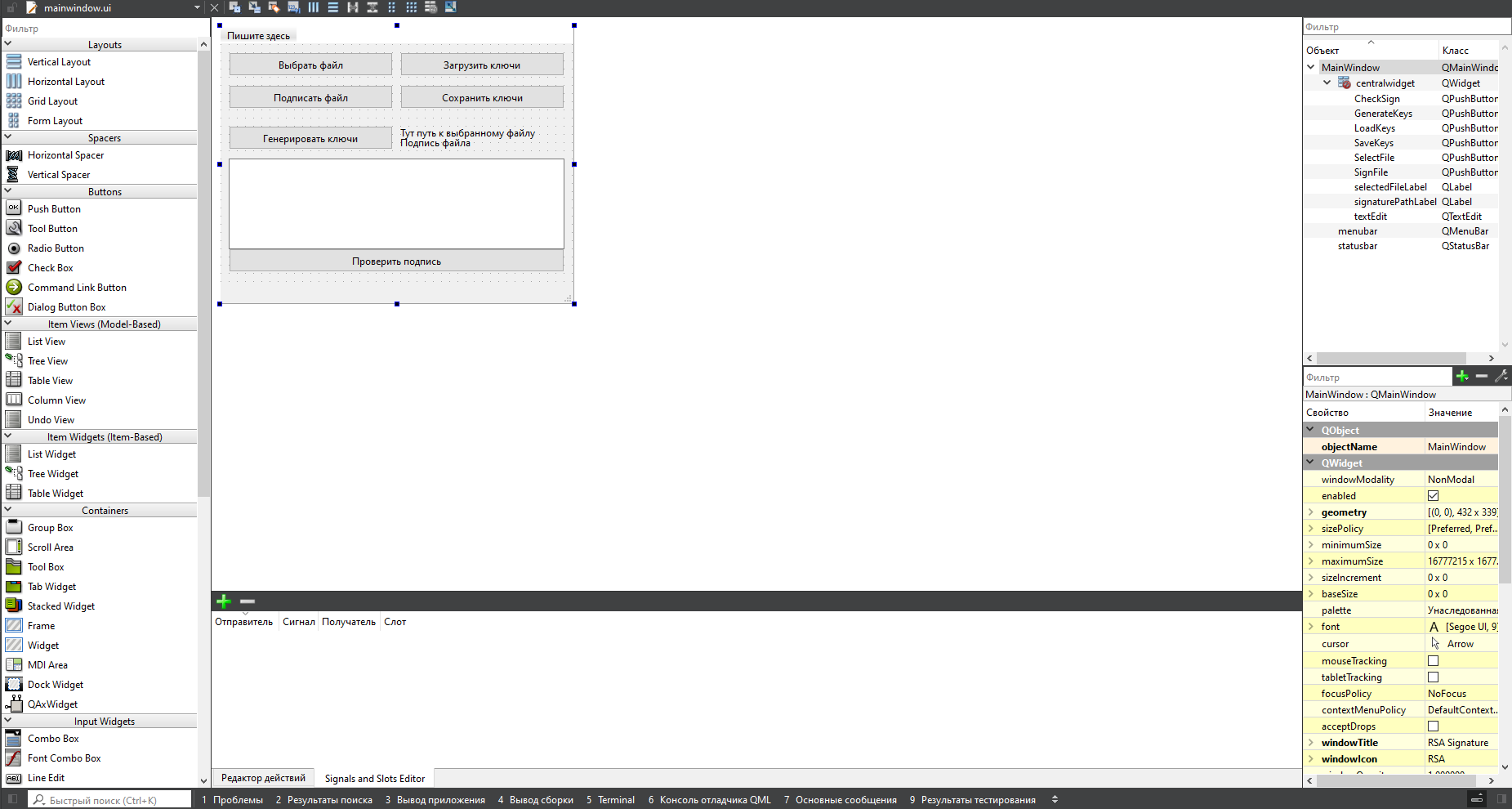
****

Рис.2Раздел для разработки дизайна

Всего в приложении 2 каталога:

В каталоге Header Files находятся следующие файлы(рис.3):



Рис.3 Каталог Header Files

mainwindow.h - это заголовочный файл для класса главного окна приложения. Этот файл содержит объявления функций, переменных и других элементов, которые используются в классе MainWindow.

Файл rsa.h содержит класс RSA, который содержит приватные переменные, которые представляют простые числа p и q, модуль n, функцию Эйлера от n и открытую и закрытую экспоненты e и d.

Класс также содержит несколько приватных методов для вычисления наибольшего общего делителя (gcd), проверки, является ли число простым (isPrime) и нахождения обратного по модулю (modInverse).

Класс имеет несколько публичных методов, таких как GenerateKeys() для генерации ключей шифрования, convertToLongLong() для преобразования массива байтов в числа типа long long, modPow() для вычисления остатка от деления числа на модуль в степени, encrypt() и decrypt() для шифрования и расшифровки сообщения, sign() и verify() для подписи и проверки подписи сообщения.

В каталоге Source Files находятся следующие файлы(рис.4):

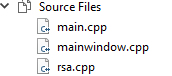


Рис.4 Каталог Source Files

Файл main.cpp - это главный файл программы, он содержит функцию main(). Эта функция является точкой входа в программу и запускает приложение.

В файле main.cpp выполняются следующие задачи:

- Создание экземпляра класса QApplication, который является основным объектом приложения.

- Создание экземпляра класса MainWindow, который представляет главное окно приложения.

- Отображение главного окна приложения с помощью метода show().

- Запуск главного цикла приложения с помощью метода exec().

То есть main.cpp является ключевым компонентом проекта, поскольку он определяет основную логику работы приложения и его запуск.

Файл mainwindow.cpp является исходным кодом для класса MainWindow, который представляет главное окно приложения. Он содержит реализацию методов и функций этого класса, таких как:

-функция on\_GenerateKeys\_clicked генерации ключей и их преобразование в base64 строки для отображения;

-функция on\_SaveKeys\_clicked сохранения ключей с последующим созданием файла с этими ключами, преобразованными в формат base64;

-функция on\_SelectFile\_clicked для выбора и загрузки файла, который собираетесь подписать;

-функция on\_LoadKeys\_clicked для выбора и загрузки файла, в который сохранили ключи и последующим преобразованием их обратно числа типа long long;

-функция on\_SignFile\_clicked для чтения содержимого файла и последующего его подписания и сохранения подписи в новый файл;

-функция on\_CheckSign\_clicked для чтения содержимого файла, проверки, является ли файл подписанным, и последующей проверки подписи на подлинность

Файл rsa.cpp содержит в себе алгоритм создания ключей с помощью криптографического алгоритма RSA.

**1.3 Тестирование**

Интеллектуальная карта приложения(рис.4).

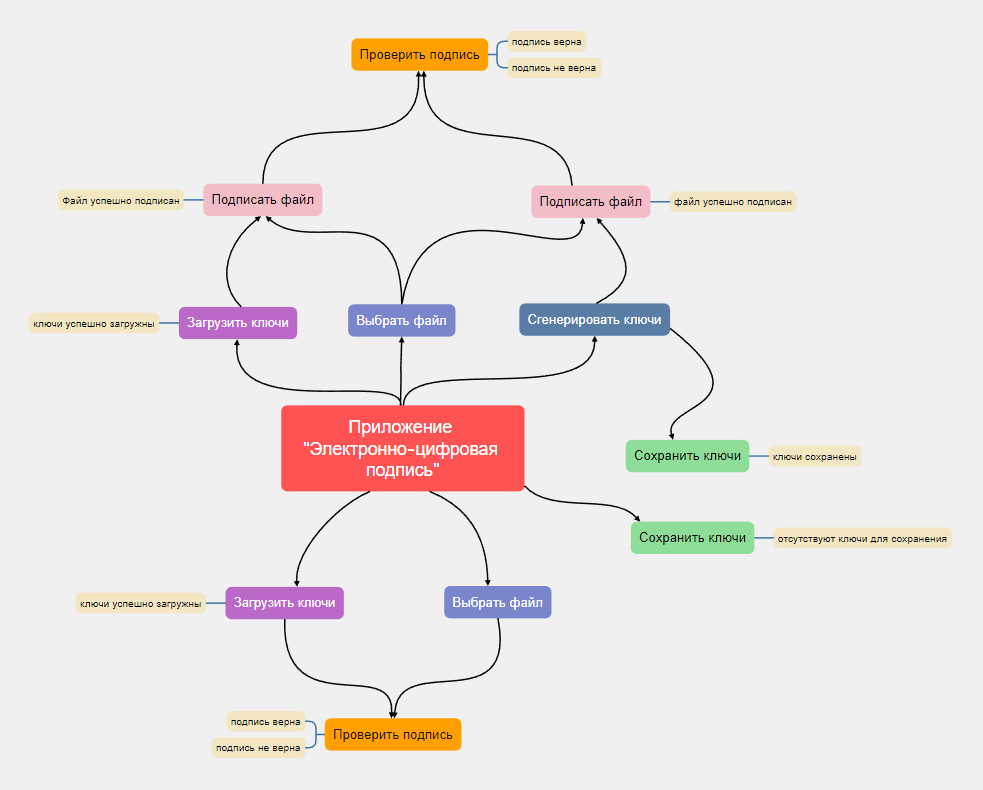


Рис.4 Интеллектуальная карта приложения

**1.3.1 Описание отчета о тестировании**

Назначение отчета о тестировании приложения заключается в документировании результатов тестирования, выявленных проблем и ошибок, а также в оценке качества и готовности приложения к использованию.

**1.3.2 Цель тестирования**

Цель тестирования приложения заключается в проверке корректности работы алгоритма шифрования и расшифровки сообщений, а также в проверке правильности создания и проверки электронной подписи.

Также целью тестирования является проверка производительности алгоритма шифрования и расшифровки сообщений на больших объемах данных и определение возможных узких мест в производительности.

**1.3.3 Методика тестирования**

1. Метод функционального тестирования. Данный метод предполагает проверку работоспособности каждой функции приложения, а именно проверка генерации ключей, подписывания и проверки цифровой подписи, шифрования и расшифрования сообщений, загрузка и сохранение файлов.

2. Методы тестирования производительности. Данный метод позволяет оценить скорость работы приложения на различных объемах данных.

3. Методы тестирования пользовательского интерфейса. Данный метод позволяет оценить удобство использования приложения пользователем. Это проверка удобства генерации ключей, подписывания и проверки цифровой подписи, шифрования и расшифрования сообщений, загрузка и сохранение файлов через пользовательский интерфейс.

**1.3.4 Проведенные тесты**

1. Сценарий тестирования функциональности:

- Проверка генерации ключей: создание пары ключей и проверка их корректности.

- Проверка подписывания и проверки цифровой подписи: подписание сообщения, проверка подписи на соответствие подписанному сообщению.

- Проверка шифрования и расшифрования сообщений: шифрование сообщения, расшифрование полученного шифротекста и проверка корректности расшифрованного сообщения.

Тесты пройдены успешно, можно утверждать, что функциональность приложения работает корректно.

2. Сценарий тестирования производительности:

- Оценка времени выполнения операций шифрования и расшифровки на различных объемах данных.

На основе полученных результатов можно сделать вывод , что приложение работает достаточно быстро на различных объемах данных, чего достаточно для приятного пользования им.

3. Сценарий тестирования пользовательского интерфейса:

- Проверка удобства использования функций приложения через пользовательский интерфейс.

На основе полученных результатов можно сделать вывод о том, что приложение достаточно удобное и пользователям будет комфортно работать ним.

**2. Источники, использованные при разработке**

1. Программирование на языке С++ в среде Qt Creator: / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О. В. Чеснокова, А. С. Чмыхало — М. : ALT Linux, 2015. — 448 с. : ил. — (Библиотека ALT Linux). [ISBN 978-5-905167-16-4]

2. Об электронной подписи и её перспективах в цифровой экономике. С.С. Дубов, Я.Я. Месенгисер [DOI: 10.32362/2500-316X-2018-6-5-5-14]

3. Дополнительная образовательная программа «ЭЛЕКТРОННО-ЦИФРОВАЯ ПОДПИСЬ» [Электронный ресурс] URL: <https://elar.rsvpu.ru/bitstream/123456789/29064/1/RSVPU_2019_060.pdf>

4. Создание простого оконного приложения в Qt Creator. [Электронный ресурс] URL: <https://statmod.ru/wiki/_media/study:spring2016:cmcpp:window_app_qtc2.pdf>

5. Справочная карточка Qt Creator. [Электронный ресурс] URL: <https://overapi.com/static/cs/qtcreator.pdf>

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема**  Информационная система электронно-цифровой подписи

**Руководство программиста**

Р.02069337.22/311-05 РП-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 51

Исполнитель:

студент гр. ИСТбд-21

*Александров Данила Вячеславович*

« » 2023 г.

2023

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение "Электронно-цифровая подпись" с алгоритмом шифрования RSA предназначено для обеспечения безопасности передачи данных в интернете. RSA является одним из наиболее распространенных алгоритмов шифрования, используемых для защиты конфиденциальных данных, таких как пароли, банковские данные, электронная почта и т.д.

Функциональные возможности приложения включают в себя генерацию пары ключей (открытый и закрытый), подписание сообщений цифровой подписью, проверку подписи, шифрование и расшифрование сообщений.

Общая характеристика функциональных возможностей приложения "Электронно-цифровая подпись" с алгоритмом шифрования RSA заключается в обеспечении безопасности передачи данных в интернете с помощью генерации пары ключей, подписания сообщений цифровой подписью, шифрования и расшифрования сообщений.

**1.2 Условия, необходимые для использования приложения**

Рекомендуемая конфигурация технического обеспечения:

* процессор – Intel Core i3-8100 3.6 ГГц;
* количество ядер – 4;
* жесткий диск – Жесткий диск WD Blue [WD10EZEX] (SATA III, 6 Гбит/с, 7200 об/мин, кэш память - 64 МБ**)** (1 TБ);
* оперативная память – 8 ГБ;
* видеокарта – NVIDIA GeForce GTX 1060 3GB;

Проект создавался с использованием такого компонента, как Desktop Qt 6.6.0 MSVC2019 64bit

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых (т.е. выполняющих какие-то действия) строк программного кода равно 481.

Сторонние библиотеки не были использованы.

Внешний вид приложения сразу после запуска выглядит следующим образом(рис.5).

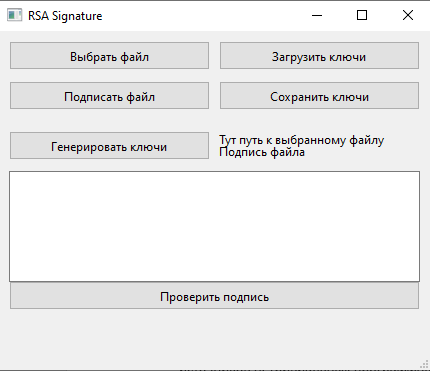


Рис.5 Дизайн приложения

После нажатия кнопки “Генерировать ключи” в окно вывода появится информация о сгенерированных ключах(рис.6).

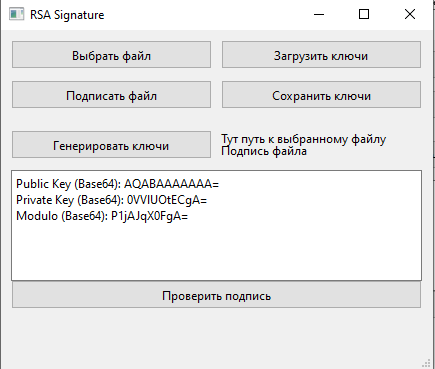


Рис.(6) Информация о ключах в окне вывода

Если вы после генерации ключей нажмёте кнопку “Сохранить ключи”, то увидите такое уведомление и ключа сохранятся в каталог, в которой находится программа(рис.7)

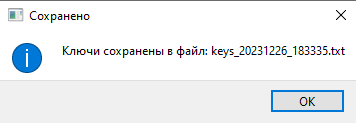


Рис.7 Уведомление о сохранении ключей

После нажатия кнопки “Выбрать файл” откроется каталог вашего компьютера, и вам нужно будет выбрать файл, который хотите подписать(рис.8).

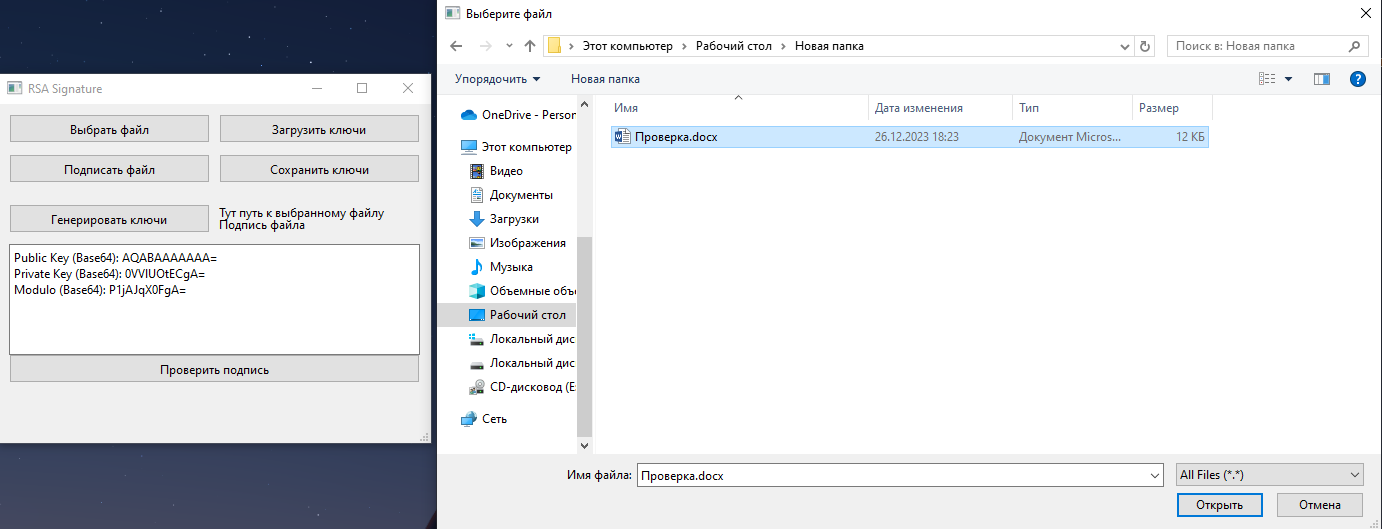


Рис.8 Открытие каталога для выбора файла, который хотите подписать

Тут будут выводиться имена файлов, которые вы загружаете в программу(рис.9).

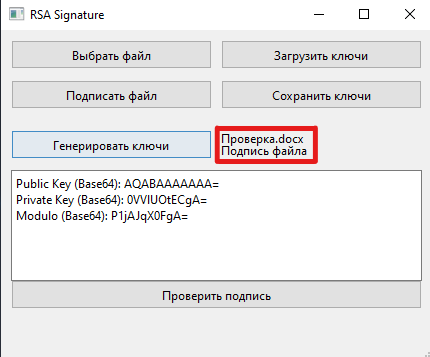
**

Рис.9 Отображение имён загружаемых файлов

Если после всех проделанных выше действий вы нажмёте копку “Подписать файл”, то увидите такое уведомление и ваш файл будет сохранён в каталог, в которой находится программа(рис.10).

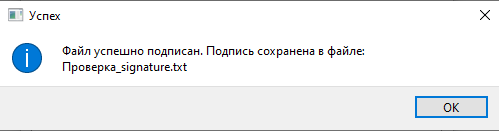


Рис.10 Уведомление после подписания файла

Если нажмёте кнопку “Загрузить ключи”, то у вас откроется каталог, как и в случае с выбором файла(рис.11), вы должны будете открыть папку с программой и там выбрать файл с ключами, после вы увидите такое уведомление(рис.12), а ключи появятся в окне вывода в приложении(рис.13).

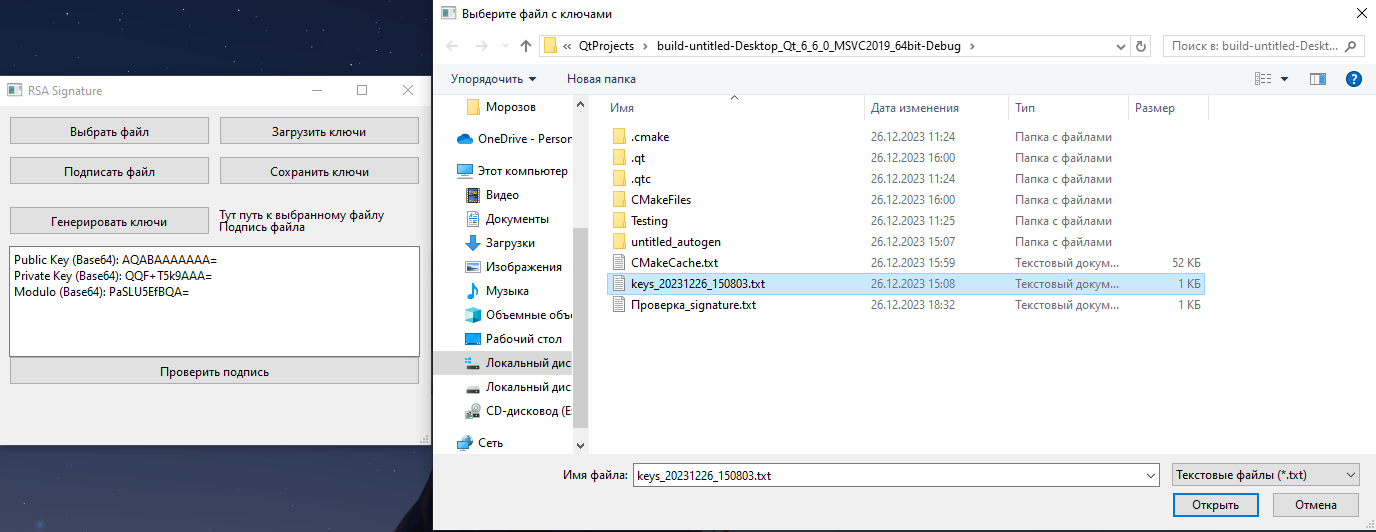


Рис.12 Выбор файла с ключами из каталога с программой

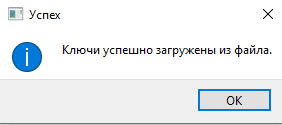
****

Рис.12 Уведомление о загрузке ключей

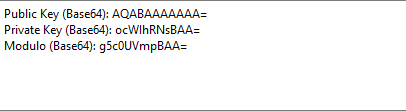
****

Рис.13 Отображение ключей

Если файл после того, как его подписали, не был изменён, то в случае нажатия на кнопку “Проверить файл” вы увидите такое уведомление(рис.14), в ином случае, если файл изменяли, вы увидите это предупреждение(рис.15).

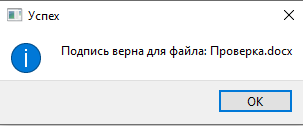


Рис.14 Удачная проверка файла на подлинность подписи

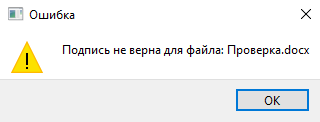


Рис.14 Неудачная проверка файла на подлинность подписи

**2.2 Особенности реализации приложения**

*1. Локальные переменные в методе isPrime:*Описание: Используются для проверки простоты числа методом проверки на делимость.  
Примечание: Включает в себя переменные num, i.

*2. Локальные переменные в методе gcd:*Описание: Используются для вычисления наибольшего общего делителя (НОД) методом Евклида.  
Примечание: Включает в себя переменные a, b, temp.

*3. Локальные переменные в методе modInverse:*Описание: Используются для вычисления обратного по модулю числа методом расширенного алгоритма Евклида.  
Примечание: Включает в себя переменные a, m, m0, t, q, x0, x1.

*4. Локальные переменные в методе GenerateKeys:*Описание: Используются для генерации простых чисел p и q и вычисления ключей RSA.  
Примечание: Включает в себя переменные rd, gen, distribution.

*5. Локальные переменные в методах encrypt, modPow, decrypt, sign,* **verify:**Описание: Используются для промежуточных вычислений в алгоритмах шифрования, расшифрования и подписи.  
Примечание: Включают в себя переменные message, base, exponent, modulus, result, hash, numericData, signedByte.

*6. Массив hash в методе sign:*Описание: Используется для хранения хеша данных (SHA-256).  
Примечание: Включает в себя элементы, представляющие хеш каждого байта данных.

*7. Массивы data, signature, sign в методах sign и verify:*Описание: Используются для представления данных (в виде массива байт), цифровой подписи и подписанного байта соответственно.  
Примечание: Включают в себя элементы, представляющие байты данных и подписи.

8. Локальные переменные в методе convertToLongLong:  
Описание: Используются для преобразования массива байт в число типа long long.  
Примечание: Включают в себя переменные result, shift, i.

**3. Обращение к программе**

Наименования и полные описания методов и алгоритмов в классе RSA:  
  
**1. RSA** (конструктор):  
*Описание*: Конструктор класса RSA, инициализирующий объект RSA.

**2. isPrime**(long long num):  
*Описание*: Метод проверки, является ли число num простым.  
*Параметры*:  
-num: Число для проверки.  
*Возвращаемое* *значение*: true, если num простое, иначе false.

**3. gcd**(long long a, long long b):  
*Описание*: Метод для вычисления наибольшего общего делителя (НОД) для чисел a и b с использованием алгоритма Евклида.  
*Параметры*:  
-a: Первое число.  
-b: Второе число.  
*Возвращаемое* *значение*: НОД(a, b).

**4. modInverse**(long long a, long long m):  
*Описание*: Метод для вычисления обратного по модулю числа a в кольце вычетов по модулю m.  
*Параметры*:  
-a: Число, для которого вычисляется обратное.  
-m: Модуль.

*Возвращаемое* *значение*: Обратное по модулю a в кольце вычетов по модулю m.

**5. GenerateKeys**():  
*Описание*: Метод генерации открытого и закрытого ключей для алгоритма RSA.

**6. convertToLongLong**(const QByteArray &data):  
*Описание*: Метод для преобразования массива байт data в число типа long long.  
*Параметры*:  
-data: Массив байт.  
*Возвращаемое* *значение*: Число типа long long, полученное из массива байт.

**7. modPow**(long long base, long long exponent, long long modulus):  
*Описание*: Метод для вычисления значения (base^exponent) mod modulus с использованием алгоритма быстрого возведения в степень по модулю.  
*Параметры*:  
-base: Основание.  
-exponent: Показатель степени.  
-modulus: Модуль.  
*Возвращаемое* *значение*: (base^exponent) mod modulus.

**8. encrypt**(long long message):  
*Описание*: Метод для шифрования сообщения с использованием открытого ключа.  
*Параметры*:  
-message: Сообщение для шифрования.  
*Возвращаемое* *значение*: Зашифрованное сообщение.

**9. decrypt**(long long encryptedMessage):  
*Описание*: Метод для расшифровки зашифрованного сообщения с использованием закрытого ключа.  
*Параметры*:  
-encryptedMessage: Зашифрованное сообщение.  
*Возвращаемое значение*: Расшифрованное сообщение.

**10. sign**(const QByteArray &data, long long privateKey, long long modulus):  
*Описание:* Метод для создания цифровой подписи для массива байт data.  
*Параметры*:  
-data: Массив байт, для которого создается подпись.  
-privateKey: Закрытый ключ.  
-modulus: Модуль.  
*Возвращаемое значение*: Цифровая подпись в виде массива байт.

**11. verify**(const QByteArray &data, const QByteArray &signature, long long publicKey, long long modulus):  
*Описание:* Метод для проверки цифровой подписи для массива байт data.  
*Параметры:*  
-data: Массив байт, для которого проверяется подпись.  
-signature: Цифровая подпись в виде массива байт.  
-publicKey: Открытый ключ.  
-modulus: Модуль.  
*Возвращаемое значение:* true, если подпись верна, иначе false.

**12. getPublicKey**():  
*Описание:* Метод для получения открытой экспоненты (открытого ключа).  
Возвращаемое значение: Открытая экспонента.

**13. getPrivateKey**():  
*Описание*: Метод для получения закрытой экспоненты (закрытого ключа).  
Возвращаемое значение: Закрытая экспонента.

**14. getModuloN**():  
*Описание*: Метод для получения модуля n.  
Возвращаемое значение: Модуль n.

Ниже представлено описание основных методов класса MainWindow, который представляет основное окно приложения:

1. **MainWindow::MainWindow(QWidget parent):**  
   Конструктор класса MainWindow, инициализирующий основное окно.  
   Вызывает метод showKeys().
2. **MainWindow::~MainWindow():**Деструктор класса MainWindow, освобождающий ресурсы.
3. **MainWindow::on\_SelectFile\_clicked():**  
   Слот, вызываемый при нажатии кнопки "SelectFile".  
   Открывает диалог выбора файла, сохраняет выбранный файл.  
   Отображает имя выбранного файла в QTextEdit.
4. **MainWindow::showKeys():**  
   Метод для отображения ключей в QTextEdit (пока не реализован).
5. **convertToBase64(long long number):**  
   Статическая функция для преобразования числа типа long long в строку Base64.
6. **convertFromBase64(const QString &base64Str):**Статическая функция для преобразования строки Base64 в число типа long long.
7. **MainWindow::on\_GenerateKeys\_clicked():**Слот, вызываемый при нажатии кнопки "GenerateKeys".  
   Генерирует пару ключей (открытый и закрытый) с использованием алгоритма RSA.  
   Отображает ключи в QTextEdit.
8. **MainWindow::on\_SaveKeys\_clicked():**Слот, вызываемый при нажатии кнопки "SaveKeys".  
   Сохраняет сгенерированные ключи в текстовый файл в формате Base64.
9. **MainWindow::on\_LoadKeys\_clicked():**Слот, вызываемый при нажатии кнопки "LoadKeys".  
   Загружает ключи из текстового файла в формате Base64.
10. **MainWindow::on\_SignFile\_clicked():**  
    Слот, вызываемый при нажатии кнопки "SignFile".  
    Подписывает выбранный файл и сохраняет подпись в новый файл.
11. **MainWindow::on\_CheckSign\_clicked():**Слот, вызываемый при нажатии кнопки "CheckSign".  
    Проверяет подпись выбранного файла и выводит сообщение о результате.

**4. Сообщения**

-Подпись верна для файла: {name\_file};

-Подпись не верна для файла: {name\_file};

-Ключи успешно загружены из файла.;

-Файл успешно подписан. Подпись сохранена в файле: {name\_file};

-Ключи сохранены в файл: {name\_file};

# 

**Заключение**  
Разработанная в рамках курсовой работы система Электронно-цифровой подписи полностью соответствует предъявляемым к ней требованиям технического задания. Заявленные функции реализованы и корректно работают.

Поддерживается создание ключей, шифрование файлов с помощью асимметричного алгоритма шифрования RSA и возможность создания электронной цифровой подписи для выбранного электронного документа.

Особые трудности возникали при работе с алгоритмом шифрования, а также при обеспечении его корректной работы. Также сложности были при переводе ключей в формат Base64 и обратно в числа типа long long, но эти трудности в основном были преодолены.

# Приложение А. Исходные тексты программных модулей

*Каталог Header Files:***Файл mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QRandomGenerator64>

#include <QMainWindow>

#include <QFileDialog>

#include <QDir>

#include <QString>

#include <QCryptographicHash>

#include "rsa.h"

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_SelectFile\_clicked();

void on\_GenerateKeys\_clicked();

void on\_SaveKeys\_clicked();

void on\_LoadKeys\_clicked();

void on\_SignFile\_clicked();

void on\_CheckSign\_clicked();

private:  
 Ui::MainWindow \*ui;

void on\_loadFileButton\_clicked();

void showKeys();

long long publicKey;

long long privateKey;

long long modulo;

QString filePath;

void generateRSAKeys();

RSA rsa;

};

#endif // MAINWINDOW\_H

**Файл rsa.h**

#ifndef RSA\_H

#include <QByteArray>

#define RSA\_H

#include <string>

class RSA {

private:

long long p, q; // простые числа

long long n; // модуль

long long phi; // функция Эйлера от n

long long e, d; // открытая и закрытая экспоненты

bool isPrime(long long num);

long long gcd(long long a, long long b);

long long modInverse(long long a, long long m);

public:

RSA();

void GenerateKeys();

long long convertToLongLong(const QByteArray &data);

long long modPow(long long base, long long exponent, long long modulus);

long long encrypt(long long message);

long long decrypt(long long encryptedMessage);

QByteArray sign(const QByteArray &data, long long privateKey, long long modulus);

bool verify(const QByteArray &data, const QByteArray &signature, long long publicKey, long long modulus);

long long getPublicKey() {

return e; // Возвращаем открытую экспоненту

}

long long getPrivateKey() {

return d; // Возвращаем закрытую экспоненту

}

long long getModuloN(){

return n;

}

};

#endif // RSA\_H

*Каталог Source Files:*

**Файл main.cpp**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**Файл** **mainwindow.cpp**

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include <QFileDialog>

#include <QDir>

#include <QMessageBox>

#include "rsa.h"

#include <QStringList>

bool publicKeyExists = false;

bool privateKeyExists = false;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

showKeys();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_SelectFile\_clicked() {

QString filePathh = QFileDialog::getOpenFileName(this, tr("Выберите файл"), "", tr("All Files (\*.\*)"));

// Проверка, что файл был выбран

if (!filePathh.isEmpty()) {

filePath = filePathh;

QFileInfo fileInfo(filePathh);

QString fileNameOnly = fileInfo.fileName(); // Эта переменная содержит только имя файла без пути

ui->selectedFileLabel->clear();

ui->selectedFileLabel->setText(fileNameOnly);

}

}

void MainWindow::showKeys()

{

}

QString convertToBase64(long long number) {

QByteArray byteArray(reinterpret\_cast<const char\*>(&number), sizeof(number));

return byteArray.toBase64();

}

long long convertFromBase64(const QString &base64Str) {

QByteArray byteArray = QByteArray::fromBase64(base64Str.toUtf8());

return \*reinterpret\_cast<const long long\*>(byteArray.constData());

}

void MainWindow::on\_GenerateKeys\_clicked()

{

rsa.GenerateKeys();

publicKey = rsa.getPublicKey();

privateKey = rsa.getPrivateKey();

modulo = rsa.getModuloN();

// Преобразование ключей в base64 строки для отображения

QString publicKeyBase64 = convertToBase64(publicKey);

QString privateKeyBase64 = convertToBase64(privateKey);

QString moduloBase64 = convertToBase64(modulo);

// Отображение ключей в QTextEdit

ui->textEdit->clear(); // Очистка текста, если нужно

ui->textEdit->append("Public Key (Base64): " + publicKeyBase64);

ui->textEdit->append("Private Key (Base64): " + privateKeyBase64);

ui->textEdit->append("Modulo (Base64): " + moduloBase64);

publicKeyExists = true;

privateKeyExists = true;

}

void MainWindow::on\_SaveKeys\_clicked()

{

if (!publicKeyExists || !privateKeyExists) {

// Если ключи не существуют, выход из функции

QMessageBox::warning(this, "Ошибка", "Отсутствуют ключи для сохранения.");

return;

}

// Создание имени файла на основе текущей даты

QString fileName = "keys\_" + QDateTime::currentDateTime().toString("yyyyMMdd\_hhmmss") + ".txt";

// Открытие файла для записи

QFile file(fileName);

if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&file);

// Запись ключей в файл в формате Base64

out << "PublicKey: " << convertToBase64(publicKey) << "\n";

out << "PrivateKey: " << convertToBase64(privateKey) << "\n";

out << "Modulo:" << convertToBase64(modulo) << "\n";

file.close();

// Если файл был успешно сохранен, показать сообщение с именем файла

QMessageBox::information(this, "Сохранено", "Ключи сохранены в файл: " + fileName);

}

}

void MainWindow::on\_LoadKeys\_clicked() {

QString filePath = QFileDialog::getOpenFileName(this, tr("Выберите файл с ключами"), "", tr("Текстовые файлы (\*.txt)"));

if (!filePath.isEmpty()) {

QFile file(filePath);

if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {

QTextStream in(&file);

QString loadedPublicKeyBase64;

QString loadedPrivateKeyBase64;

QString loadedModuloBase64;

bool publicKeyFound = false;

bool privateKeyFound = false;

bool moduloFound = false;

while (!in.atEnd()) {

QString line = in.readLine();

if (!publicKeyFound && line.startsWith("PublicKey:")) {

loadedPublicKeyBase64 = line.section(":", 1).trimmed();

publicKeyFound = true;

} else if (!privateKeyFound && line.startsWith("PrivateKey:")) {

loadedPrivateKeyBase64 = line.section(":", 1).trimmed();

privateKeyFound = true;

} else if (!moduloFound && line.startsWith("Modulo:")){

loadedModuloBase64 = line.section(":", 1).trimmed();

moduloFound = true;

}

if (publicKeyFound && privateKeyFound && moduloFound) {

break;

}

}

// Преобразование строк из Base64 обратно в числа типа long long

publicKey = convertFromBase64(loadedPublicKeyBase64);

privateKey = convertFromBase64(loadedPrivateKeyBase64);

modulo = convertFromBase64(loadedModuloBase64);

// Отображение ключей в QTextEdit в формате Base64

ui->textEdit->clear(); // Очистка текста, если нужно

ui->textEdit->append("Public Key (Base64): " + loadedPublicKeyBase64);

ui->textEdit->append("Private Key (Base64): " + loadedPrivateKeyBase64);

ui->textEdit->append("Modulo (Base64): " + loadedModuloBase64);

// Вывод сообщения об успешном чтении ключей

QMessageBox::information(this, "Успех", "Ключи успешно загружены из файла.");

file.close();

} else {

QMessageBox::critical(this, "Ошибка", "Не удалось открыть файл с ключами.");

}

}

}

void MainWindow::on\_SignFile\_clicked() {

if (!filePath.isEmpty()) {

// Чтение содержимого файла

QFile file(filePath);

if (file.open(QIODevice::ReadOnly)) {

QByteArray data = file.readAll();

file.close();

QByteArray signature = rsa.sign(data, privateKey, modulo);

// Сохранение подписи в новый файл

QFileInfo fileInfo(filePath);

QString signatureFileName = fileInfo.baseName() + "\_signature.txt";

QFile signatureFile(signatureFileName);

if (signatureFile.open(QIODevice::WriteOnly | QIODevice::Text)) {

signatureFile.write(signature); // Записываем подпись в файл

signatureFile.close();

// Вывод сообщения об успешной подписи

ui->signaturePathLabel->clear();

ui->signaturePathLabel->setText(signatureFileName);

QMessageBox::information(this, "Успех", "Файл успешно подписан. Подпись сохранена в файле: " + signatureFileName);

} else {

// Обработка ошибки сохранения подписи

QMessageBox::critical(this, "Ошибка", "Не удалось сохранить подпись в файл.");

}

} else {

// Обработка ошибки чтения файла

QMessageBox::critical(this, "Ошибка", "Не удалось прочитать файл для подписи.");

}

}

}

void MainWindow::on\_CheckSign\_clicked() {

if (!filePath.isEmpty()) {

// Чтение содержимого файла

QFile file(filePath);

if (file.open(QIODevice::ReadOnly)) {

QByteArray data = file.readAll();

file.close();

// Получение имени файла с подписью

QFileInfo fileInfo(filePath);

QString signatureFileName = fileInfo.baseName() + "\_signature.txt";

QFile signatureFile(signatureFileName);

if (signatureFile.open(QIODevice::ReadOnly)) {

QByteArray signature = signatureFile.readAll();

signatureFile.close();

// Проверка подписи

bool isValid = rsa.verify(data, signature, privateKey, modulo);

if (isValid) {

QMessageBox::information(this, "Успех", "Подпись верна для файла: " + fileInfo.fileName());

} else {

QMessageBox::warning(this, "Ошибка", "Подпись не верна для файла: " + fileInfo.fileName());

}

} else {

QMessageBox::critical(this, "Ошибка", "Не удалось открыть файл с подписью.");

}

} else {

// Обработка ошибки чтения файла

QMessageBox::critical(this, "Ошибка", "Не удалось прочитать файл для проверки подписи.");

}

}

}

**Файл rsa.cpp**

#include "rsa.h"

#include <random>

#include <cmath>

#include <QByteArray>

#include <QCryptographicHash>

bool RSA::isPrime(long long num) {

if (num <= 1) return false;

if (num <= 3) return true;

if (num % 2 == 0 || num % 3 == 0) return false;

for (long long i = 5; i \* i <= num; i = i + 6) {

if (num % i == 0 || num % (i + 2) == 0) return false;

}

return true;

}

long long RSA::gcd(long long a, long long b) {

while (b != 0) {

long long temp = b;

b = a % b;

a = temp;

}

return a;

}

long long RSA::modInverse(long long a, long long m) {

long long m0 = m, t, q;

long long x0 = 0, x1 = 1;

if (m == 1) return 0;

// Применение расширенного алгоритма Евклида

while (a > 1) {

// q - частное

q = a / m;

t = m;

// m - остаток, пересчитываем его

m = a % m;

a = t;

t = x0;

// x0, x1 - обновление коэффициентов

x0 = x1 - q \* x0;

x1 = t;

}

// Если x1 отрицательное, приводим его к положительному значению

if (x1 < 0) x1 += m0;

return x1;

}

void RSA::GenerateKeys()

{

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<long long> distribution(10000000, 100000000);

// Генерация простых чисел p и q

p = distribution(gen);

q = distribution(gen);

while (!isPrime(p)) p = distribution(gen);

while (!isPrime(q) || q == p) q = distribution(gen);

// Вычисление n и функции Эйлера от n

n = p \* q;

phi = (p - 1) \* (q - 1);

// Выбор открытой экспоненты e

e = 65537; // Начальное значение для e

while (e < phi) {

if (gcd(e, phi) == 1) break;

else e++;

}

// Вычисление закрытой экспоненты d

d = modInverse(e, phi);

}

RSA::RSA()

{

}

long long RSA::encrypt(long long message) {

return modPow(message, e, n);

}

// Функция для возведения числа в степень по модулю

long long RSA::modPow(long long base, long long exponent, long long modulus) {

if (modulus == 1) return 0;

long long result = 1;

base = base % modulus;

while (exponent > 0) {

if (exponent % 2 == 1) {

result = (result \* base) % modulus;

}

exponent = exponent >> 1;

base = (base \* base) % modulus;

}

return result;

}

long long RSA::decrypt(long long encryptedMessage) {

return modPow(encryptedMessage, d, n);

}

QByteArray RSA::sign(const QByteArray &data, long long privateKey, long long modulus) {

// Хеширование данных (SHA-256)

QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);

// Применение RSA к каждому байту хеша

QByteArray signature;

for (int i = 0; i < hash.size(); ++i) {

long long numericData = static\_cast<long long>(static\_cast<unsigned char>(hash[i]));

// Подписание данных

long long signedByte = modPow(numericData, privateKey, modulus);

// Добавление подписанного байта в массив

signature.append(static\_cast<char>(signedByte));

}

return signature;

}

long long RSA::convertToLongLong(const QByteArray &data) {

// Проверка на то, что размер данных не превышает размер long long

long long result = 0;

int shift = 0;

// Проходим по массиву байтов, преобразуя каждый байт в число и добавляя его к результату

for (int i = 0; i < data.size(); ++i) {

result |= static\_cast<long long>(static\_cast<unsigned char>(data[i])) << shift;

shift += 8; // Смещение для следующего байта

}

return result;

}

bool RSA::verify(const QByteArray &data, const QByteArray &signature, long long publicKey, long long modulus) {

// Хеширование данных (SHA-256)

QByteArray hash = QCryptographicHash::hash(data, QCryptographicHash::Sha256);

// Применение RSA к каждому байту хеша

QByteArray sign;

for (int i = 0; i < hash.size(); ++i) {

long long numericData = static\_cast<long long>(static\_cast<unsigned char>(hash[i]));

// Подписание данных

long long signedByte = modPow(numericData, publicKey, modulus);

// Добавление подписанного байта в массив

sign.append(static\_cast<char>(signedByte));

}

return signature==sign;

}