

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ
ШИФРОВАНИЯ ДАННЫХ**

Студент

Веня 31.05.22 Симакова В.А.
(подпись, дата)

Руководитель

Дорошенко 31.05.22 Дорошенко И.Н.
(подпись, дата)

Нормоконтролёр

Бычков 31.05.22 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой

Митрохин
(подпись)

М.А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от
15.06.2022 № 10)

Секретарь ГЭК

Гурин
(подпись)

Е.И. Гурин

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 16 » апреля 20 22 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Симакова Виктория Алексеевна гр. 18BB2 факультета вычислительной техники.
2. Тема работы «Программное обеспечение для шифрования данных»
Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022г.
3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Дорошенко И.Н.
4. Задание на работу:
5. Перечень подлежащих разработке вопросов:

Разработать программное обеспечение, обеспечивающее шифрование данных методом комбинированного шифрования, и их дешифрацию. Спроектировать интуитивно понятный графический интерфейс.

Требуется провести анализ предметной области, рассмотреть программы-аналоги, разработать архитектуру приложения, выбрать используемые средства для разработки. Необходимо изучить способы проектирования объектно-ориентированных приложений на платформе .NET Framework с помощью технологии WPF.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение темы предметной области	10	26.04.2022-28.04.2022	<i>Дорошенко</i>
2	Анализ требований к ПО	10	29.04.2022-01.05.2022	<i>Дорошенко</i>
3	Выбор используемых технологий	10	02.05.2022-04.05.2022	<i>Дорошенко</i>
4	Проектирование архитектуры	15	04.05.2022-06.05.2022	<i>Дорошенко</i>
5	Выбор алгоритмов решения	10	07.05.2022-09.05.2022	<i>Дорошенко</i>
6	Разработка ПО	30	10.05.2022-20.05.2022	<i>Дорошенко</i>
7	Тестирование и отладка	10	21.05.2022-28.05.2022	<i>Дорошенко</i>
8	Оформление выпускной работы	5	29.05.2022-31.05.2022	<i>Дорошенко</i>

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы *Дорошенко* 26.04.22 Дорошенко И.Н.
(подпись, дата)

Задание к исполнению принял студент *Симакова* 26.04.22 Симакова В.А.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л.Р. Фионова* 14.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 96 листов, 47 рисунков, 4 таблицы, 6 источников, 3 приложения.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ШИФРАЦИЯ И ДЕШИФРАЦИЯ ДАННЫХ, КОМБИНИРОВАННОЕ ШИФРОВАНИЕ, ХЭШИРОВАНИЕ, ОТКРЫТЫЙ И ЗАКРЫТЫЙ КЛЮЧ, XML.

Цель работы – разработать программное обеспечение, позволяющее пользователю зашифровывать данные с помощью метода комбинированного шифрования и расшифровывать их.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: авторизацию пользователя в системе, генерацию открытого и закрытого ключа, шифрацию и дешифрацию файлов, создание XML-файла.

В результате проведенной работы реализован проект, удовлетворяющий поставленной задаче, а именно создано программное обеспечение осуществляющее шифрацию и дешифрацию данных.

					ПГУ 1.090301.37.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение для шифрования данных Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Симакова В.А.					2	96	
Проверил		Дорошенко И.Н.							
Н. Контр.		Бычков А.С.				ФВТ, каф. ВТ			
Утверд.		Митрохин М.А.				Гр. 18ВВ2			

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Программное обеспечение для шифрования данных.....	8
1.1 Анализ предметной области	8
1.1.1 Существующие алгоритмы шифрования	8
1.1.1.1 Симметричный алгоритм AES-128.....	8
1.1.1.2 Ассиметричный алгоритм RSA-2048	10
1.1.1.3 Комбинированное шифрование	12
1.1.1.4 Алгоритм хэширования SHA-256	13
1.1.2 Обзор программных средств шифрования.....	14
1.1.2.1 Finencrypt	14
1.1.2.2 Max File Encryption	16
1.1.3 Постановка задачи	17
1.1.4 Выбор используемых программных средств	17
1.1.4.1 Язык разработки	18
1.1.4.2 Платформа разработки графического интерфейса.....	20
1.1.4.3 База данных	21
1.1.5 Выбор архитектуры решения	22
1.2 Реализация	24
1.2.1 Описание архитектуры решения.....	24
1.2.2 Описание классов программы	27
1.2.2.1 Класс AuthWindow.....	28
1.2.2.2 Класс MainWindow	28
1.2.2.3 Класс User.....	29
1.2.2.4 Класс ApplicationContext.....	30
1.2.2.5 Класс UserPage	30
1.2.2.6 Класс Window1.....	31

1.2.2.7 Класс HybridEncryption	32
1.2.3 Разработка графического интерфейса	33
1.3 Оценка результата.....	35
1.3.1 Руководство пользователя	35
1.3.1.1 Вход и регистрация в приложении	35
1.3.1.2 Шифрация и дешифрация файлов	37
1.3.1 Отладка и тестирование	42
1.3.1.1 Тестирование ввода данных	42
1.3.1.2 Тестирование целостности файлов при шифровании.....	50
1.3.1.3 Тестирование скорости шифрования документов	51
2 Безопасность жизнедеятельности.....	54
2.1 Характеристика потенциальных опасностей и вредностей.....	54
2.2 Санитарно-гигиеническая характеристика производства	55
2.3 Электробезопасность	57
2.4 Пожарная безопасность	58
2.5 Расчетно-аналитическая часть.....	58
Заключение	62
Список используемых источников.....	63
Приложение А – UML – диаграммы	64
Приложение Б – Листинг программы	68
Приложение В – Презентация.....	89

Перечень сокращений и обозначений

AES (Advanced Encryption Standart) – симметричный алгоритм блочного шифрования

RSA (аббревиатура от фамилий *Rivest, Shamir, Adleman*) – ассиметричный алгоритм шифрования

SHA (Secure Hash Algorithm) – алгоритм хэширования

UI (User Interface) – пользовательский интерфейс

WPF (Windows Presentation Foundation) – система для построения приложений

XML (eXtensible Markup Language) – расширяемый язык разметки

ООП – объектно-ориентированное программирование

ПК – персональный компьютер

ПО – программное обеспечение

ПФ – проверка функциональности

СУБД – система управления базами данных

Введение

За последние несколько десятилетий произошла революция в области вычислительной техники, и все указывает на то, что технический прогресс и использование информационных технологий будут продолжаться быстрыми темпами. В настоящее время инновации в области информационных технологий оказывают широкомасштабное воздействие на многие сферы жизни. Благодаря результатам прогресса мы можем облегчить различные сферы жизнедеятельности: от заказа доставок на дом до быстрого отправления денежных переводов за границу.

В свою очередь предоставленные информационной эпохой возможности также позволяют вторгнуться и угрожать безопасности частной жизни граждан. В современном мире криптография является важнейшим инструментом защиты информации в компьютерных системах и обеспечения конфиденциальности данных.

Криптография в повседневной жизни используется повсеместно. Криптографические методы позволяют безопасно передавать информацию в сети Интернет, надежно сохранять пароли для онлайн-покупок, и обеспечивают устойчивую аутентификацию пользователей и устройств.

В период *COVID-19* многие компании столкнулись с необходимостью быстрого внедрения практик удаленной работы для сотрудников, что породило проблемы, связанные с безопасностью данных. Многие неопытные пользователи стали сохранять, передавать и работать с защищенными файлами компании на своих персональных компьютерах. Появилась возможность того, что файлы, которые обычно защищены с помощью нескольких уровней безопасности и контролируются профессиональными ИТ-специалистами, могут быть подвергнуты кибератакам, что потенциально может привести к раскрытию конфиденциальности и нарушению целостности данных.

Актуальным решением данной проблемы является разработка и проектирование приложения, позволяющего обеспечить безопасность данных с помощью надежного алгоритма, а также имеющего интуитивно понятный интерфейс для быстрого освоения программы пользователями, независимо от их опыта использования вычислительной техники.

В данной работе будет рассмотрена реализация программного обеспечения для шифрования данных. Приложение направлено на шифрацию документов методом комбинированного шифрования с помощью криптостойких алгоритмов *AES-128* и *RSA-2048*, и последующей их дешифрации.

					ПГУ 1.090301.37.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Программное обеспечение для шифрования данных

1.1 Анализ предметной области

1.1.1 Существующие алгоритмы шифрования

С появлением компьютера необходимость в автоматизированных средствах для защиты файлов и другой информации, хранящейся на компьютере, стала обязательной. Криптография является ключевым способом предотвратить раскрытие данных. Она защищает информацию с помощью математики и логики надежных методов шифрования.

Данные должны быть закодированы с помощью алгоритмов шифрования, которые представляют собой математические вычисления. Существует различное множество алгоритмов, каждый из которых отличается путем применения и индексом безопасности.

Для использования алгоритмов также необходим ключ шифрования. После применения ключа и соответствующего алгоритма, исходный открытый текст преобразуется в зашифрованный фрагмент данных. В дальнейшем можно использовать ключ дешифрования для преобразования зашифрованного текста обратно в его исходный читаемый формат.

Далее подробно рассмотрены алгоритмы, используемые в данной работе, а именно симметричный алгоритм *AES-128*, асимметричный алгоритм *RSA-2048* и алгоритм хэширования *SHA-256*, и также их совместное применение.

1.1.1.1 Симметричный алгоритм AES-128

Симметричным называют шифр, в котором для шифрования и дешифрования используется один и тот же ключ (или разные ключи, которые связаны между собой в математическом смысле, то есть один ключ легко вычисляется из другого – фактически, такая ситуация все равно сводится к одному ключу). Поскольку единственный ключ полностью обеспечивает

секретность шифра, он должен храниться в полной тайне. Если злоумышленник завладеет ключом, он сможет расшифровывать и зашифровывать сообщения, сведя все усилия обороняющейся стороны на нет.

На рисунке 1 показано, как работает симметричное шифрование:



Рисунок 1 – Схема симметричного шифрования

AES (Advanced Encryption Standard) является наиболее широко используемым блочным симметричным шифром на сегодняшний день.

Алгоритм *AES* использует сеть подстановки-перестановки с несколькими раундами для получения зашифрованного текста [1]. Количество раундов зависит от используемого размера ключа (128-битный размер ключа диктует десять раундов). Каждый из этих раундов требует раундового ключа, но поскольку в алгоритм вводится только один ключ, этот ключ необходимо видоизменять, чтобы получить ключи для каждого раунда.

Каждый раунд алгоритма состоит из четырех шагов. Первоначально при этом методе шифрования ваш исходный текст (то есть информация, которую вы хотите зашифровать) разделяется на блоки. Размер блока, выбранного алгоритма, *AES* составляет 128 бит, поэтому он разделяет данные на столбец четыре на четыре из шестнадцати байтов.

Далее определяется ключ, который будет в дальнейшем использоваться для создания ряда ключей для следующих раундов процесса шифрования. Ключ добавляется к блоку исходного текста с помощью шифра *XOR*.

Затем следует шаг подстановки байтов, где данные изменяются в соответствии с таблицей. Потом в матрице происходит сдвиг строк и смешение столбцов, что помогает внести путаницу в информацию. В заключении к результату добавляется раундовый ключ. Повторное выполнение шагов гарантирует безопасность конечного зашифрованного текста.

1.1.1.2 Ассиметричный алгоритм RSA-2048

Асимметричные алгоритмы шифрования используют два разных ключа. Ключ, используемый для зашифровывания информации, называется открытым, а ключ для расшифровывания – закрытым. Эти два ключа связаны между собой особым математическим образом.

Открытый ключ используется для зашифровывания информации и проверки электронно-цифровой подписи. Этот ключ передается по открытому каналу связи, что решает проблему симметричных алгоритмов первой передачи ключа. Закрытый ключ используется для расшифровывания информации и для генерации электронно-цифровой подписи. Этот ключ хранится у одной стороны и является секретным. Закрытый ключ является наиболее уязвимым компонентом всей криптосистемы. Поэтому особое внимание нужно уделять способу хранения закрытого ключа. Например, пользователь может хранить закрытый ключ на своем персональном компьютере, защитив его с помощью пароля.

На рисунке 2 показано, как работает асимметричное шифрование:



Рисунок 2 – Схема асимметричного шифрования

Процесс шифрования можно описать следующим образом. Исходный текст должен быть переведен в числовую форму [2]. В результате этого текст представляется в виде одного большого числа. Затем полученное число разбивается на блоки так, чтобы каждая из них была числом в промежутке $[0, N - 1]$. Процесс шифрования одинаков для каждого блока. Поэтому мы можем считать, что блок исходного текста представлен числом x , $0 \leq x \leq N-1$.

Каждый абонент вырабатывает свою пару ключей. Для этого он генерирует два больших простых числа p и q , вычисляет произведение:

$$N = p \cdot q.$$

Затем он вырабатывает случайное число e , взаимно простое со значением функции Эйлера от числа N :

$$\phi(N) = (p-1) \cdot (q-1).$$

И находит число d из условия:

$$e \cdot d = 1 \pmod{\phi(N)}.$$

Так как $(e, \phi(N)) = 1$, то такое число d существует, и оно единственно. Пару (N, e) он объявляет открытым ключом и помещает в открытый доступ. Пара (N, d) является секретным ключом.

Безопасность алгоритма RSA основана на трудоемкости разложения на множители больших чисел, что обеспечивает криптостойкость алгоритма.

1.1.1.3 Комбинированное шифрование

Анализ рассмотренных выше особенностей симметричных и асимметричных алгоритмов показывает, что при совместном использовании они эффективно дополняют друг друга, компенсируя недостатки.

Действительно, главное достоинство асимметричной криптосистемы с открытым ключом заключается в обеспечении высокой безопасности. Однако их быстроедействие обычно в сотни раз меньше, чем в симметричных криптосистемах с секретным ключом.

В свою очередь, недостатком симметричных криптосистем является обновляемый секретный ключ, который система должна регулярно передавать для информационного обмена. Во время передач возникает угроза раскрытия секретного ключа, что подвергает к раскрытию конфиденциальности данных.

Совместное использование алгоритмов позволяет эффективно реализовывать такую функцию защиты, как закрытие информации с целью обеспечения ее конфиденциальности при передаче.

Стоит отметить, что применяемые в данной работе алгоритмы относятся к вычислительно стойким шифрам. Это означает, что возможность потенциального криптоаналитического вскрытия шифра при всех заданных параметрах и ключах алгоритма шифрования на современной стадии развития криптографического анализа у злоумышленника не существует, т.к. нет необходимых вычислительно-алгоритмических и временных ресурсов, чтобы реализовать вскрытие. Считается что, если алгоритм, который осуществляет вскрытие шифра должен выполнить больше или порядка 280 операций, шифр является вычислительно стойким.

Таким образом, комбинированная (или гибридная) криптосистема – это система шифрования, которая совмещает безопасность алгоритмов с открытым ключом с производительностью симметричных криптосистем.

Метод комбинированного использования заключается в применении симметричных алгоритмов для шифрования исходного открытого текста, а ассиметричные с открытым ключом применяют только для кодирования секретного ключа симметричной криптосистемы. В результате ассиметричная криптосистема с открытым ключом не заменяет, а лишь дополняет симметричную с секретным ключом, позволяя повысить защищенность передаваемой информации.

1.1.1.4 Алгоритм хэширования *SHA-256*

Хеширование — это алгоритм, который преобразует исходные данные любого объёма в небольшой набор выходных данных фиксированной длины. Важными моментами для нас в этом преобразовании являются:

- длина выходного сообщения, т.е. хэша, всегда одинакова, независимо от длины входных данных;
- любые изменения в исходном сообщении приводят к генерации уникального хеша;
- практическая невозможность восстановления исходного сообщения по известному хешу.

SHA-256 — это реализация одного из множества возможных алгоритмов хеширования. Благодаря своей относительной простоте, достаточной надёжности и криптостойкости, а также приемлемой скорости работы, *SHA-256* получил широкое распространение в современном мире. Он используется в телекоммуникациях, интернете, для шифрования беспроводных пакетов и т. д.

Исходя из этих свойств хеширования, становятся понятными области его применения — это контроль за целостностью различных сообщений, хранение «отпечатков» паролей и т. д. Связка шифрующих алгоритмов с *SHA-256* в практических приложениях позволяет создавать надёжно защищённые коммуникации.

1.1.2 Обзор программных средств шифрования

Программы, предназначенные для предотвращения несанкционированного доступа к конфиденциальной информации, условно можно разделить на три типа:

- программы, шифрующие информацию;
- программы, скрывающие информацию
- программы, не шифрующие информацию, но блокирующие несанкционированный доступ или ограничивающие доступ к данным.

На практике многие программы могут одновременно относиться к разным типам. К примеру, некоторые программы, не шифрующие информацию, но блокирующие несанкционированный доступ к ней, могут также скрывать эту информацию на жестком диске, чтобы соответствующие файлы или папки не отображались в проводнике.

В настоящее время существует достаточное количество аналогов программ для шифрования данных, таких как, например, *FineCrypt* и *Max File Encryption*.

1.1.2.1 *Finecrypt*

Программа *Finecrypt* предназначена для шифрования данных для обеспечения их безопасного хранения на компьютере или передачи через сеть Интернет. Утилита совместима с операционными системами семейства *Windows*. После установки на ПК программа интегрируется в оболочку *Windows*, в контекстном меню появляется соответствующий пункт, показанный на рисунке 3.

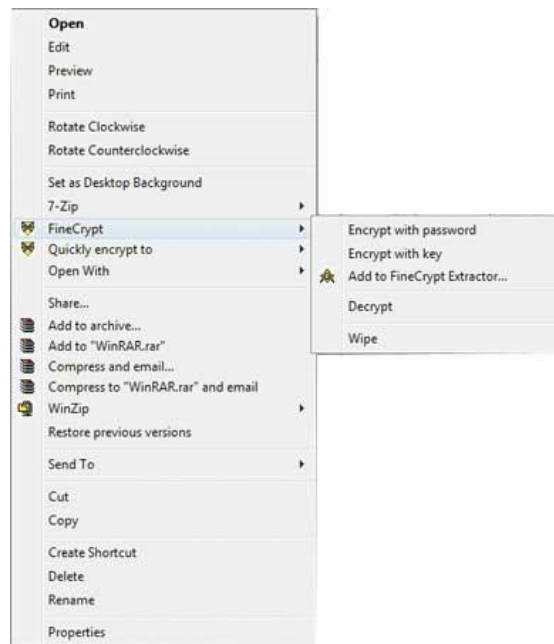


Рисунок 3 – Программа *FineCrypt* в оболочке *Windows*

Программа *FineCrypt* поддерживает шифрование отдельных документов и целых папок. При этом поддерживается как симметричное, так и асимметричное шифрование. На рисунке 4 отображены алгоритмы шифрования данных, предусмотренные программой.

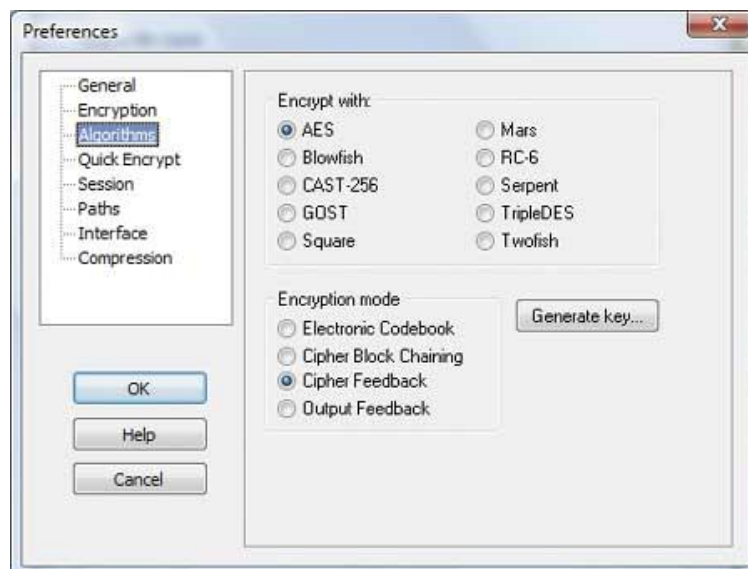


Рисунок 4 – Окно программы *FineCrypt*

Стоит отметить, что программа сложна для усвоения, т.к. не имеет четкого руководства и поддерживает только английский язык.

1.1.2.2 Max File Encryption

Max File Encryption 1.8 является небольшой и платной утилитой, предназначенной для шифрования файлов. Согласно технической документации, программа поддерживает один алгоритм шифрования – *Blowfish*, и имеет английский интерфейс, представленный на рисунке 5.

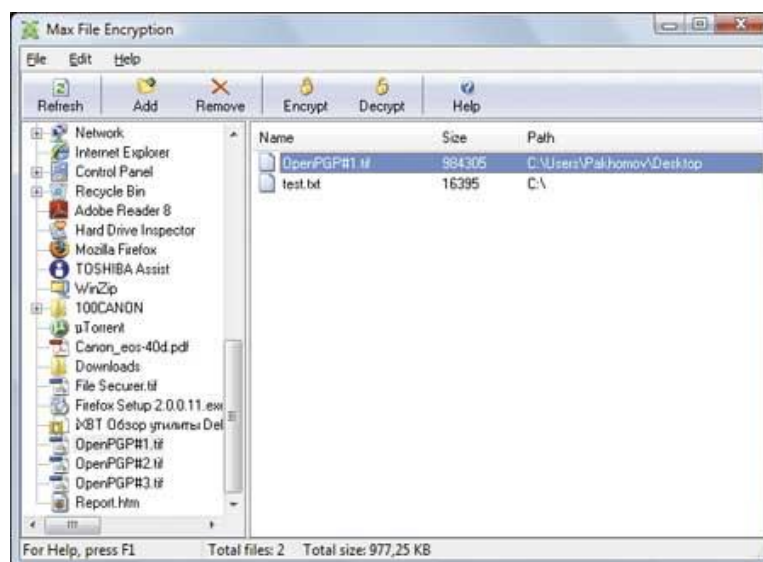


Рисунок 5 – Окно программы *Max File Encryption 1.8*

Одной из интересных функций утилиты является поддержка стеганографии, т.е. возможность скрывать файл внутри других файлов, которые являются контейнерами. Недостатком программы является невозможность шифровать папки и директории с помощью ассиметричного алгоритма.

Итак, исходя из рассмотренных выше программ, можно отметить, что преимуществом является наличие дополнительных функций, например, возможность шифрования целых папок. Стоит отметить, что утилиты применяют к файлам отдельные алгоритмы, и не предусматривают использование комбинированного шифрования. Это может привести к снижению криптостойкости и надежности кодирования документов программой, что является одним из главных недостатков. Также одним из недостатков, рассмотренных приложений, является наличие только

иностранной документации, что может привести к затрудненному освоению пользователями без навыка владения техническим английским языком.

На основе проведенного исследования можно сделать вывод, что существует множество аналогов программ, направленных на защиту данных. Однако вышеперечисленные недостатки не позволяют пользователям без труда освоить программу и зашифровывать свои данные, применяя метод комбинированного шифрования для более безопасного хранения документов.

1.1.3 Постановка задачи

В данной работе реализуется программное обеспечение, которое должно соответствовать перечню выдвинутых возможностей приложения и иметь в своём функционале следующее:

- возможность защищенной регистрации и дальнейшей работы с выбранным профилем пользователя;
- шифрование документов различных форматов посредством совместного применения алгоритмов *AES-128* и *RSA-2048*;
- генерацию пары ключей для реализации *RSA* алгоритма;
- создание *XML*-файла для хранения информации;
- создание хэш-функции для проверки целостности *XML*-файла с помощью алгоритма *SHA-256*;
- возможность дешифрации документов различных форматов.

Также стоит учитывать, что приложение должно быть совместимо с операционными системами семейства *Windows*. Для реализации используется среда разработки *Visual Studio*, где представлены наборы функций для разработки *Windows*-приложений.

1.1.4 Выбор используемых программных средств

Для разработки программного обеспечения должны быть использованы различные программные средства, позволяющие обеспечить стабильность работы программы, функциональность и ее внешний вид.

1.1.4.1 Язык разработки

После проведения анализа технического задания были рассмотрены некоторые языки программирования, которые позволили бы разработать графическое приложение с функционально обширным кодом. Во время сравнения, учитывая преимущества и особенности, был выбран – C#.

Данный язык программирования является одним из самых широко используемых языков в мире. Он прост в освоении, универсален и известен своей эффективностью, а также подходит для разработки широкого спектра программного обеспечения, включая веб-приложения, приложения Windows и видеоигры для консолей и ПК.

Язык C# использует объектно-ориентированную модель программирования, которая позволяет создавать конструкции (или же объекты), способные принимать на вход данные и отдавать их другим объектам для построения между ними взаимодействия. Данный подход обладает своими особенностями, например, использование принципа инкапсуляции, который позволяет объектам быть самодостаточными и обладать собственной функциональностью. Использование ООП при разработке, позволяет повторно использовать код через наследование, что даёт возможность не повторять написание скрипта несколько раз. Перечисленные выше преимущества помогают организовать процесс программирования наиболее продуктивным и эффективным для получения качественных результатов.

Для создания устойчивого и надежного приложения в C# используются различные функции. Например, C# имеет очень эффективную систему для стирания и удаления всего мусора, присутствующего в системе. Данный язык не создает беспорядка, и система не зависает во время выполнения. Также конструкции для обработки исключений предоставляют расширяемый и структурируемый подход к обнаружению и дальнейшему исправлению ошибок.

Стоит отметить, что язык *C#* признают наиболее понятным и быстрым для освоения благодаря простоте синтаксиса. Готовые конструкции заменяют большие блоки кода, что делает их приоритетнее в использовании, а схожесть синтаксических правил позволяет проще переходить к разработке на данном языке.

Программы, написанные на *C#*, работают на базе платформы *.NET Framework*. Платформа *.NET Framework* предназначена для создания, развертывания и запуска веб-сервисов и приложений *Windows*. Она обеспечивает контролируемую среду программирования, в которой:

- в процессе развертывания программного обеспечения и управления его версиями вероятность конфликтов сведена к минимуму;
- существует гарантия безопасного выполнения кода;
- исключаются проблемы с производительностью сред выполнения скриптов или интерпретируемого кода;
- гарантируются единые принципы разработки для создания различных типов приложений;
- обеспечивается согласованная среда для объектно-ориентированного программирования с целью локального хранения и выполнения объектного кода.

Платформа *.NET Framework* включает в себя набор стандартных библиотек классов [3]. Библиотека классов представляет собой набор методов и функций, которые можно использовать для достижения основной цели разработки. Большинство методов разделены на пространства имен, таких как *System.** или *Microsoft.**.

Важным достоинством при разработке программного обеспечения, нацеленного на шифрование данных на языке *C#*, является то, что платформа *.NET Framework* предоставляет набор криптографических объектов, поддерживающих известные и широко используемые алгоритмы. Благодаря

пространству имен *System.Security.Cryptography* разработчикам открыт программный доступ к разнообразным криптографическим сервисам. В результате их использования приложения могут кодировать и декодировать данные, обеспечивая их целостность, а также выполнять множество других операций, таких как хэширование, проверку подлинности сообщений, обработку цифровых подписей и сертификатов [4].

К описанным выше преимуществам можно добавить также легко доступный и бесплатный ряд инструментов для разработки, таких как *Microsoft Visual Studio*, *Windows Server* и другие. Среда разработки *Visual Studio* включает такую систему как *NuGet*. Она позволяет найти и скачать нужную библиотеку, затем система обновит проект, подключая скачанные файлы. Это позволяет упростить и автоматизировать работу со сторонними ресурсами.

1.1.4.2 Платформа разработки графического интерфейса

Выбранный язык для разработки предоставляет возможность использования разнообразных технологий для построения графических приложений. Например, технология *WPF*, *WinUI* и *Windows Forms*, а для разработки кроссплатформенных десктопных и мобильных приложений используется *Xamarin/MAUI*.

Наиболее известной платформой разработки графического интерфейса для клиентских приложений является *Windows Forms*. Данная технология является основной для изучения основ построения и работы *UI (User Interface)*. Такие достоинства, как большое количества элементов управления и простота работы с ними, позволяет быстро разработать простой и интуитивно понятный пользовательский интерфейс.

Однако рассматриваемая платформа не позволяет реализовать визуальное представление программы, в виде анимации, мультимедиа и т.д. В свою очередь существует такая технология как *WPF (Windows Presentation*

Form), которая позволяет использовать обширные по разнообразию наполнения библиотеки, элементы управления, стили и шаблоны, мультимедиа, двухмерную и трехмерную графики [5].

WPF позволяет разрабатывать приложения, используя как декларативный язык разметки *XAML*, так и код программной части. *XAML* используется для определения пользовательских элементов управления, окон и страниц, а для реализации поведения используется непосредственно язык программирования [6]. Такое разделение имеет ряд преимуществ:

- снижается временной ресурс, затраченный на разработку и обслуживание приложения, так как разметка не тесно связана с кодом;
- повышается эффективность разработки за счет возможности параллельной работы разработчиков, отвечающих за реализацию поведения приложения, и дизайнеров, занимающихся интерфейсом.

На основе произведенного анализа технологий можно сделать вывод, что *Windows Presentation Form* является подходящим решением для разработки и реализации проектируемого программного обеспечения.

1.1.4.3 База данных

В ходе анализа и изучения продуктов по работе с базами данных, исходя из особенностей, был выбран *SQLite*. Это локальная база данных, встраиваемая в приложения. Данная СУБД является мощным быстрым инструментом за счет обслуживающих библиотек и возможности обращаться к файлам напрямую. Она также использует *SQL*, что позволяет строить и выполнять запросы к *MySQL* или *PostgreSQL*.

Стоит отметить, что для работы с данной СУБД достаточно установить нужную библиотеку в *Visual Studio* с помощью сервиса *NuGet*. Также в данной работе используется сторонняя программа *DB Browser (SQLite)*. Окно программы изображено на рисунке 6.

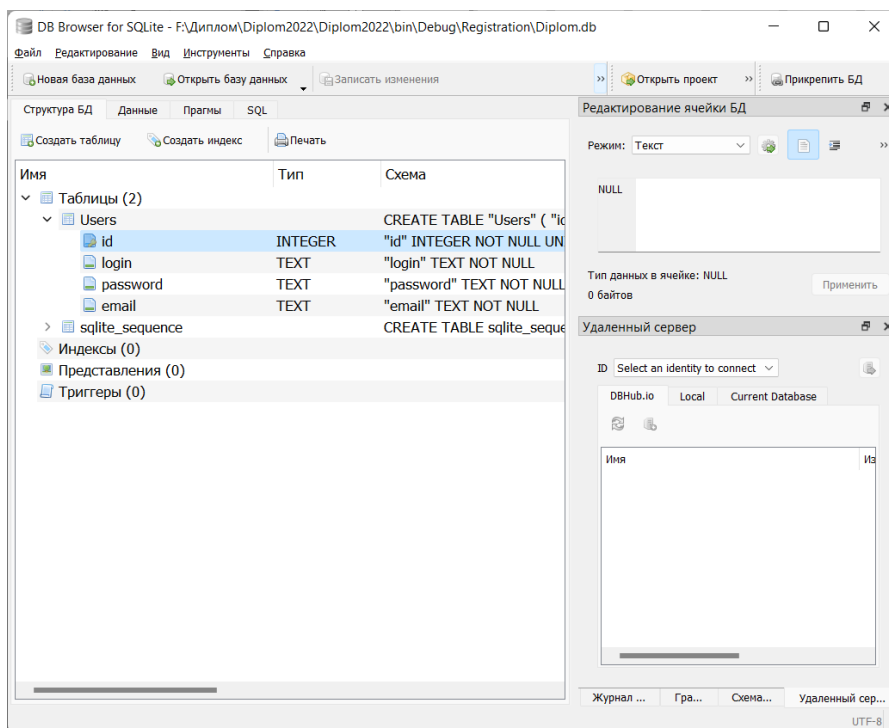


Рисунок 6 – Программа для работы с СУБД *DB Browser*

Данное программное средство обладает интуитивно понятным интерфейсом и функционалом, заменяющим сложные обращения к структуре базы данных на нажатие кнопок в окне программы. Имеется возможность размещать и подключаться к СУБД, находящейся не на локальном компьютере, посредством личного сервиса *DBHub.io*, хранящим тысячи баз данных в облачном хранилище.

1.1.5 Выбор архитектуры решения

Для разработки программного обеспечения, позволяющего зашифровывать и расшифровывать данные, была выбрана многоуровневая архитектура.

Данная архитектура является одной из моделей разработки программного обеспечения, при которой функциональные области приложения разделяются на уровни, которые имеют определенные функции. Характерной особенностью является принцип разделения ответственностей

между компонентами проекта. Каждый компонент будет обрабатывать логику только уровня, к которому он относится.

Одним из вариантов является трехуровневая архитектура (рисунок 7). Это широко применяемая архитектура разработки ПО, в которой приложения разделены на три физических и логических уровня: уровень представления, уровень приложения и уровень данных.

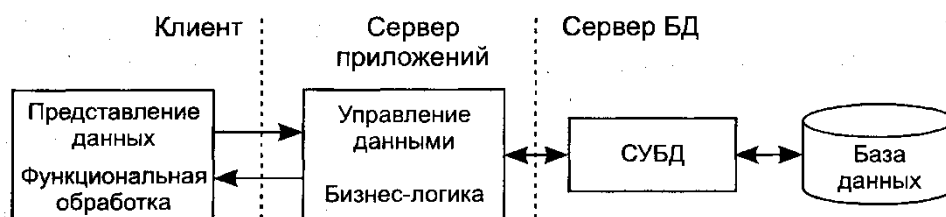


Рисунок 7 – Схема взаимодействия в трехуровневой архитектуре

Уровень представления отвечает за обработку всей логики отображения интерфейса и взаимодействия с пользователем. Его основное предназначение состоит в отображении информации. Реализацией уровня является *UI*, т. е. представление контента конечному пользователю через графический интерфейс.

Основная логика проекта реализуется на уровне приложения. Данные, обработанные на этом уровне, передаются на уровень представления, где пользовательский интерфейс принимает запрос и отображает информацию на экране.

Уровень данных представляет собой набор компонентов для доступа к хранимым данным и управления информацией, обработанной приложением. В качестве источника данных могут выступать, например, различного рода базы данных, такие как *PostgreSQL* и *MySQL*, *SOAP* и *REST* сервисы, файлы в файловой системе.

В трехуровневой архитектуре зависимости между компонентами приложения идут от уровня представления к уровню данных, т.е. обмен

информацией осуществляется только через уровень приложения. Остальные уровни не могут взаимодействовать друг с другом напрямую.

Итак, главным преимуществом трехуровневой архитектуры является разделение функциональных возможностей. Содержимое уровней можно изменять без риска глобальных конфликтов между слоями. Каждый уровень можно настраивать и оптимизировать независимо друг от друга. Также достоинством является масштабируемость и повышенная надежность, так как сбой одного из уровней не повлияет на доступность и производительность других.

1.2 Реализация

1.2.1 Описание архитектуры решения

При разработке программного обеспечения была использована трехуровневая архитектура, состоящая из уровня представления – графического интерфейса, уровня приложения, где реализуется основная логика программы, и уровня данных, отвечающего за обеспечения доступа к обрабатываемым приложением данным.

Уровень представления содержит 5 модулей, отвечающих за графический интерфейс. Модуль *AuthWindow* предназначен для входа зарегистрированного пользователя в приложение. *MainWindow* обеспечивает регистрацию и дальнейший безопасный вход в программу. *UserPage* – это главное окно приложения, благодаря которому пользователь может добавить файлы в список, создать пару ключей, зашифровать и расшифровать документ. С помощью модулей *DecryptPage* и *EncryptPassword* реализованы вспомогательные диалоговые окна, предназначенные для загрузки файлов, необходимых для дешифрации, и ввода пароля для шифрации файла соответственно. Взаимодействие модулей представлено на рисунке 8.

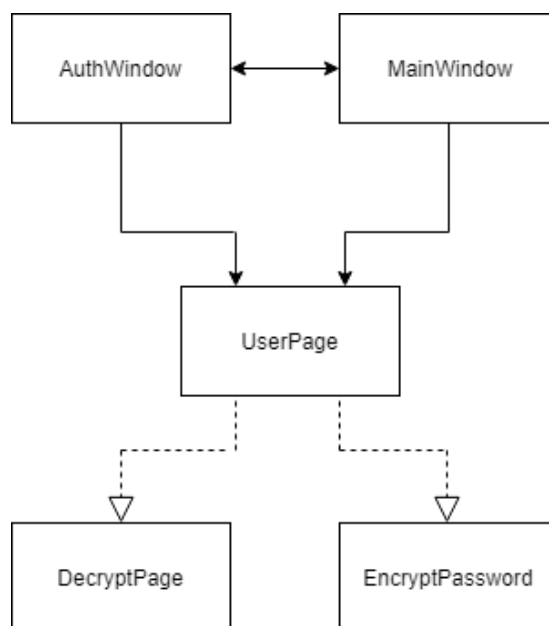


Рисунок 8 – Модули уровня представления

Уровень приложения состоит из класса *HybridEncryption*, реализующего основную логику шифрации и дешифрации. А также класс *User*, благодаря которому создается модель таблицы базы данных. Как и предполагает многоуровневая архитектура, данные, обработанные в этих классах, передаются на уровень представления и уровень данных, т.е. данный уровень является их связующим звеном.

При построении уровня данных был разработан класс *ApplicationContext*, позволяющий установить связь с базой данных. В качестве источника выступает база данных *SQLite*.

На рисунке 9 приведена диаграмма вариантов использования разрабатываемых модулей.



Рисунок 9 – Диаграмма вариантов использования

Исходя из представленной диаграммы, можно определить субъект, взаимодействующий с системой, в данном случае это пользователь. Также на диаграмме отображены множества действий, выполняемых программой, для получения определенного результата.

Вариант использования «Расшифровать документ» имеет вид отношения *include* между действиями «Ввести пароль» и «Добавить XML-файл». Такая взаимосвязь иллюстрирует, что именно использует базовый вариант для выполнения операции.

1.2.2 Описание классов программы

Каждый из описанных уровней реализуется на одном или нескольких классах, выполняющих конкретные функции. Далее будет описана работа классов с приложением *UML*-диаграмм.

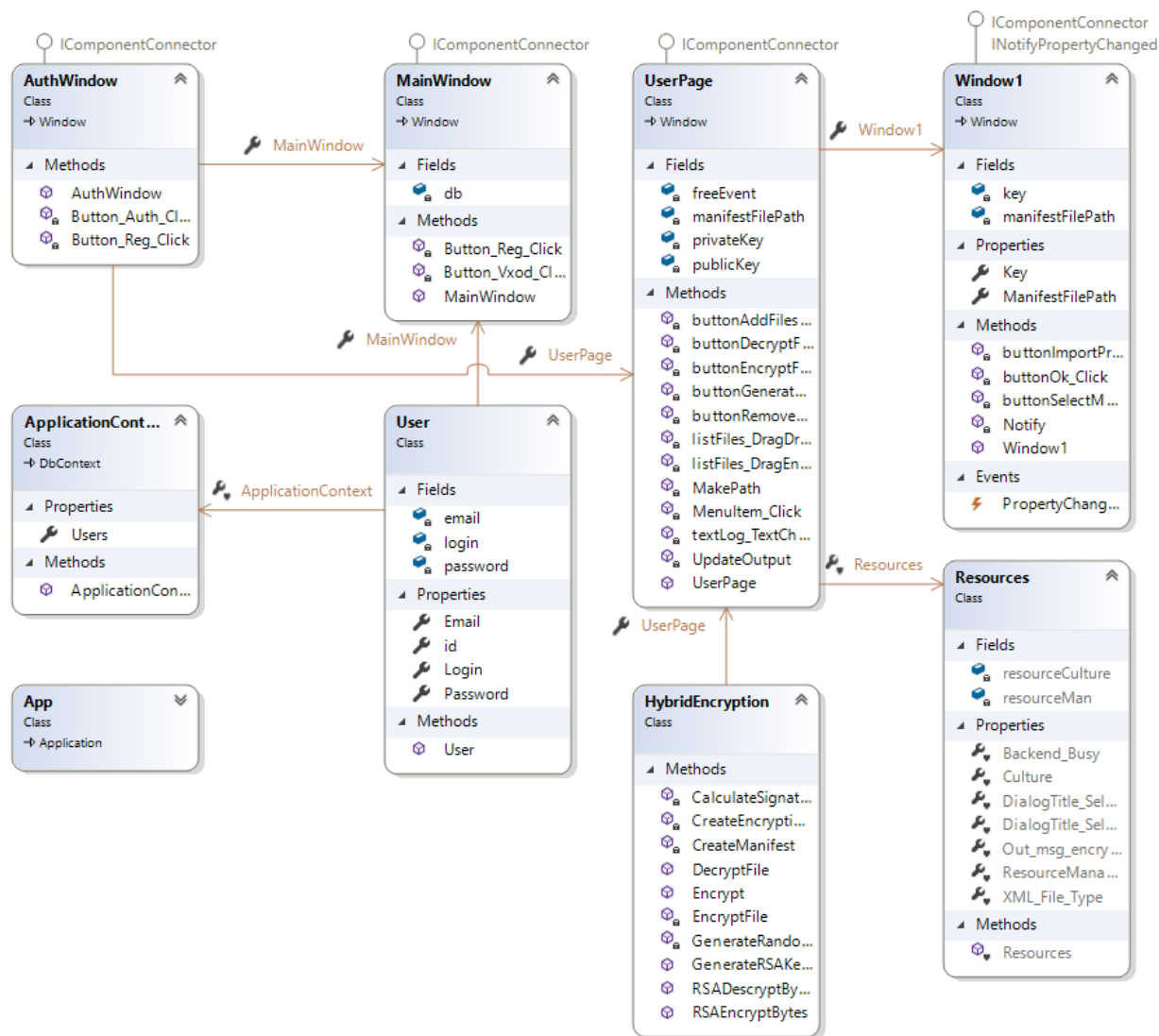


Рисунок 10 – Диаграмма классов приложения

На данной диаграмме классов представлены разработанные классы приложения: *AuthWindow*, *MainWindow*, *User*, *ApplicationContext*, *UserPage*, *Window1*, *HybridEncryption*, *EncryptPassword*, *Resources*.

1.2.2.1 Класс *AuthWindow*

Данный класс обеспечивает безопасный вход в приложение, а также определяет взаимодействие между окнами *MainWindow* и *UserPage* (рисунок 11).

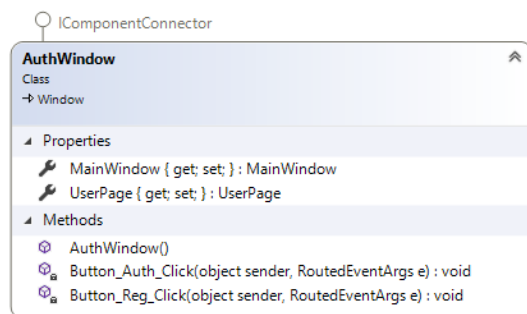


Рисунок 11 – Структура класса *AuthWindow*

Внутри класса описаны методы *Button_Auth_Click* и *Button_Reg_Click()*. Первый метод - результат нажатия кнопки, которая дает возможность пользователю произвести вход в приложение, после введения данных в соответствующие поля. В случае неправильно введенных данных или данных несуществующего пользователя вход будет неудачным и на экране появится сообщение с предупреждением. Второй метод обеспечивает переход к окну регистрации пользователя в случае, если ранее регистрация не производилась.

1.2.2.2 Класс *MainWindow*

Класс обеспечивает регистрацию нового пользователя в приложении и занесение информации о нем в базу данных (рисунок 12).

С помощью метода *Button_Reg_Click()* данные, занесенные пользователем в поля формы, обрабатываются, и после проверки заносятся в СУБД.

В методе класса имеется обработка исключений. При некорректно введенных данных, наличия пустых полей или при обнаружении схожего

электронного адреса в базе данных, будут выведены окна ошибок. Это позволяет безопасно произвести регистрацию в приложении.

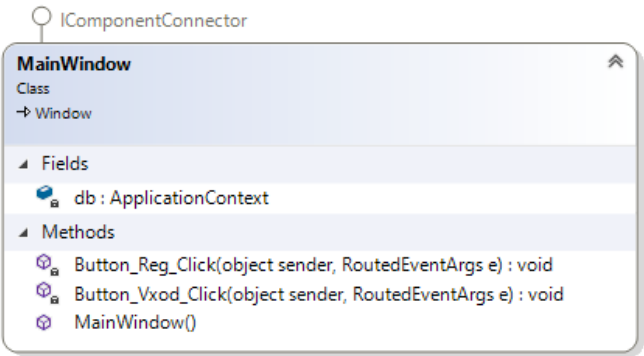


Рисунок 12 – Структура класса *MainWindow*

Метод *Button_Vxod_Click()* обеспечивает переход к графическому окну *AuthWindow*.

1.2.2.3 Класс *User*

Данный класс представляет собой класс модели, который описывает таблицу в базе данных (рисунок 13).

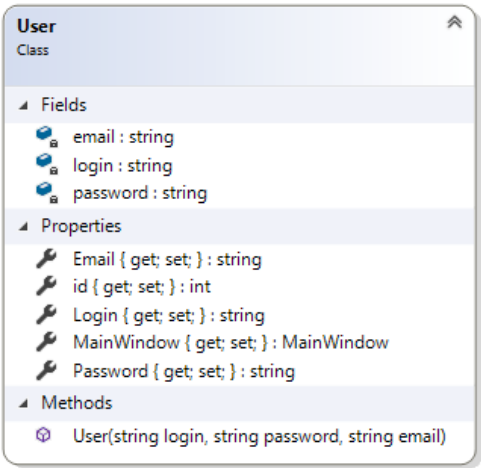


Рисунок 13 – Структура класса *User*

Требуется для хранения структур и реализации метода *User()*, который позволяет установить поля, такие как *login*, *email*, *pass*, для работы с передаваемыми данными СУБД.

1.2.2.4 Класс *ApplicationContext*

Благодаря вспомогательному классу *ApplicationContext*, представленному на рисунке 14, устанавливается соединение с текущей базой данных, что позволяет остальным классам работать с ней.

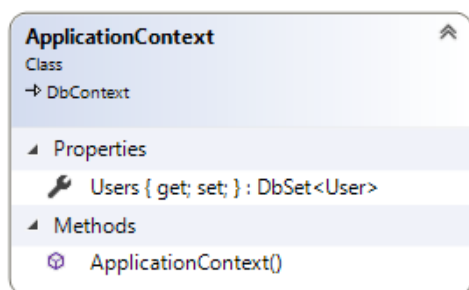


Рисунок 14 – Структура класса *ApplicationContext*

1.2.2.5 Класс *UserPage*

Класс, представленный на рисунке 15, является одним из важнейших модулей графического интерфейса.

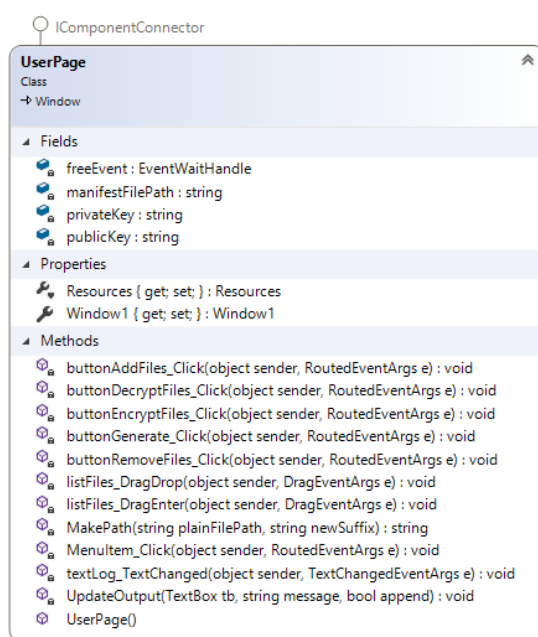


Рисунок 15 – Структура класса *UserPage*

В нем реализована обработка нажатия кнопок главного меню и взаимодействие с классом *HybridEncryption*, который отвечает за алгоритм шифрования документа.

Внутри данного класса существует метод *MakePath()*, который отвечает за создание нового файла с именем расширения для зашифрованного файла - *.encrypted*, и расшифрованного - *.decrypted*. Также реализован метод *UpdateOutput()*, который позволяет добавлять текст в поле *textLog* при удачно выполненной шифрации и дешифрации, а также отображать подсказки для пользователя.

1.2.2.6 Класс *Window1*

В представленном классе (рисунок 16) реализуется отображение диалогового окна при нажатии кнопки, предназначенной для расшифровки файлов. При помощи обработки кнопок осуществляется загрузка XML-файла и закрытого ключа.

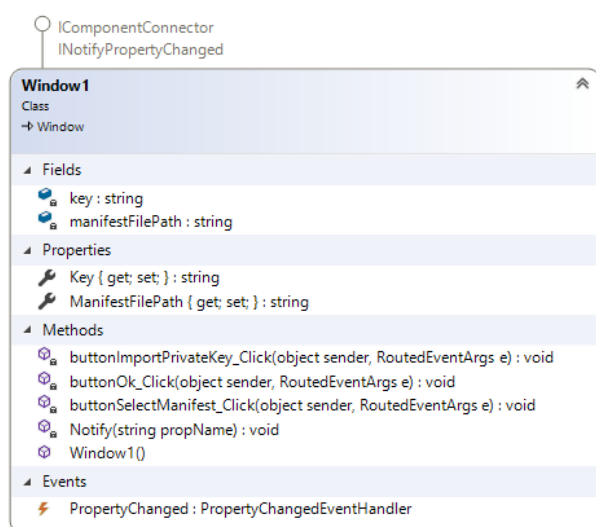


Рисунок 16 – Структура класса *Window1*

Для реализации механизма привязки полученных данных используется интерфейс *INotifyPropertyChanged*. Интерфейс предполагает генерацию события *PropertyChanged*, которое обновляется каждый раз, когда значение

свойства объекта изменяется. Событие обрабатывается в специальном методе *Notify()*. Данное поведение позволяет отслеживать изменение объектов и передавать данные между классами для дальнейшего использования.

1.2.2.7 Класс *HybridEncryption*

Данный класс предназначен для реализации комбинированного шифрования (рисунок 17).

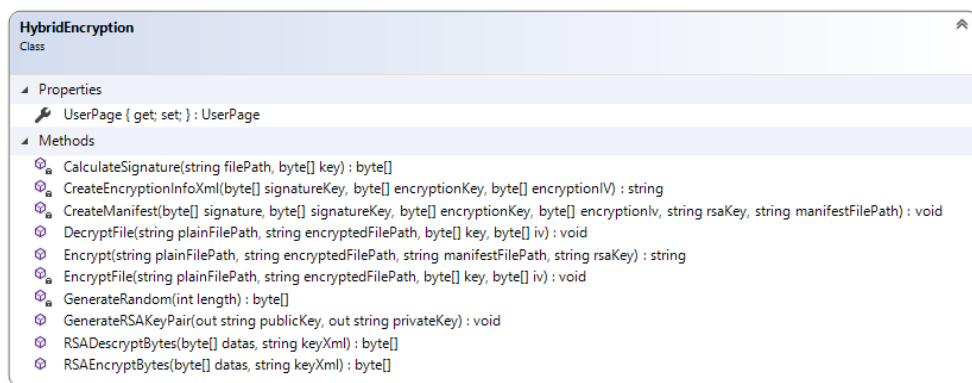


Рисунок 17 – Структура класса *HybridEncryption*

В метод *Encrypt()* передаются параметры из класса *UserPage*. Здесь вызывается метод *EncrypFile()*, позволяющий зашифровать исходный текст с помощью алгоритма *AES-128* и поместить его в новый файл с расширением *.encrypted*. С помощью функции *CreateSignature()* формируется хэш-функция для *XML*-файла. Метод *CreateManifest()* формирует файл *.manifest.XML*, где выводится ключ и вектор для *AES* алгоритма, пара ключей для *RSA* алгоритма, и созданная ранее хэш-функция. К строкам, включающим ключ и вектор *AES* алгоритма, применяется метод *RSAEncryptBytes()*, который позволяет зашифровать их с применением алгоритма *RSA-2048*. Строка, включающая хэш-функцию, также шифруется с помощью этого метода. Далее *XML*-файл отображает информацию об уже зашифрованных файлах, перезапись файла происходит с помощью метода *CreateEncryptionInfoXml()*. Данная информация выводится в блок *textLog* в классе *UserPage* при завершении кодирования.

Метод *GenerateRSAKeyPair()* формирует открытый и закрытый ключ. Функция *GenerateRandom()* позволяет сформировать массив определенного размера произвольных символов для формирования ключа и вектора *AES* алгоритма.

После применения методов *DecryptFile()* и *RSADecryptBytes()*, происходит дешифрация файла и ключей.

1.2.3 Разработка графического интерфейса

Для реализации графического интерфейса использовалась технология *WPF*. В ходе разработки были реализованы 5 окон для представления *UI*: *AuthWindow*, *MainWindow*, *UserPage*, *DecryptPage* и *EncryptPassword*.

При их проектировании главной целью было создать интуитивно понятный и удобный для пользователя интерфейс для обеспечения простоты использования приложения. Разработка интерфейса окон производилась с помощью языка разметки *XAML*, используемого для инициализации объектов в технологиях на платформе *.NET*. Использование декларативного языка позволяет отделить графический интерфейс от логики приложения. Благодаря этому разработка разных частей приложения производилась отдельно друг от друга.

Фреймворк позволяет осуществить компоновку элементов с помощью таких контейнеров, как *Grid* или *StackPanel*. Благодаря компоновке можно настроить элементы интерфейса и позиционировать их определенным образом. При разработке окон интерфейса программного обеспечения использовался контейнер *Grid* для эффективного распределения доступного пространства между элементами и нахождения их предпочтительных размеров.

При разработке интерфейса окна, представленного на рисунке 18, использованы элементы управления содержанием такие как кнопки и метки, помещенные в специальный контейнер *Grid*. А также элемент управления

Изм.	Лист	№ докум.	Подпись	Дата

списками *ListBox* для наглядного отображения добавленных файлов. Текстовый элемент управления *TextBox* позволяет отобразить уведомления для пользователя. Также на окне присутствует вкладка «Профиль», размещенная с помощью элемента *MenuItem*.

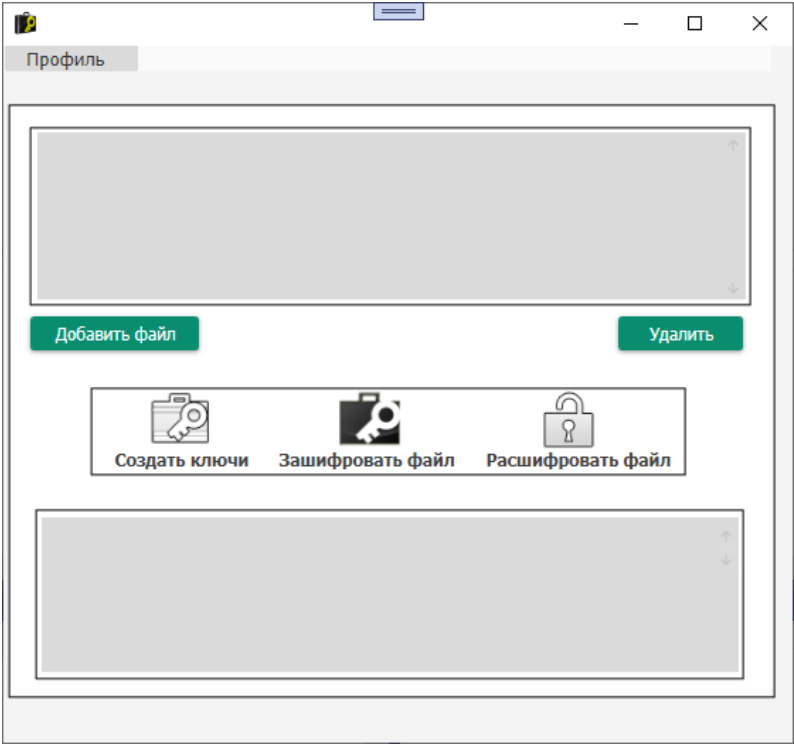


Рисунок 18 – Интерфейс главного окна программы

Каждый элемент обладает определенными дополнительными свойствами для поддержки стилистики, включающей в себя шрифты, цвета фона и т.д. Благодаря применению свойств установился единый стиль приложения.

В результате в каждом разработанном окне были использованы однотипные элементы, помещенные в контейнер, с определенными похожими дополнительными свойствами.

1.3 Оценка результата

1.3.1 Руководство пользователя

Ниже описана инструкция пользователя для корректного использования программного обеспечения.

1.3.1.1 Вход и регистрация в приложении

После запуска исполняемого файла открывается окно входа в приложение, в котором можно заполнить данные для ранее зарегистрированного пользователя или открыть окно регистрации.

После того, как данные были введены, требуется нажать кнопку «Войти». Окно отображено на рисунке 19.

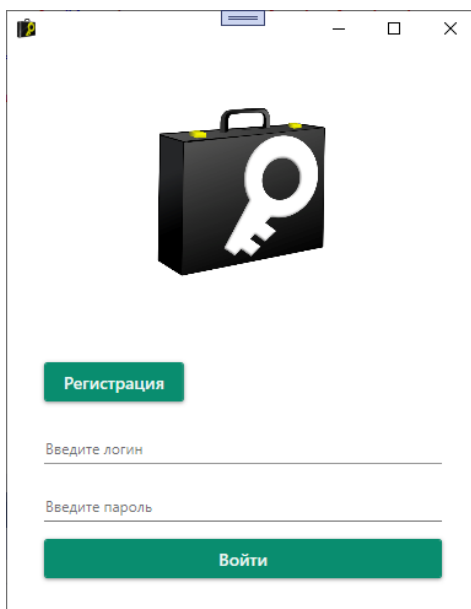


Рисунок 19 - Окно входа

Если пользователь не был зарегистрирован, то следует нажать кнопку «Регистрация». После этого появится окно, показанное на рисунке 20.

Изм.	Лист	№ докум.	Подпись	Дата

Рисунок 20 – Окно регистрации пользователя

В данном окне нужно заполнить пользовательские данные, они будут занесены в базу данных для дальнейшего использования. После введения данных в соответствующие поля требуется нажать кнопку «Зарегистрироваться». Если пользователь хочет попасть на стартовое окно, нужно нажать кнопку «Войти».

Если данные введены корректно, то на экране появится уведомление об успешно пройденной регистрации. Данное окно представлено на рисунке 21.

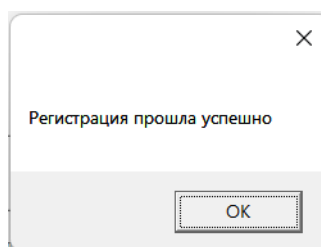


Рисунок 21 – Окно с уведомлением об успешной регистрации

В случае, если информация, введенная пользователем, не будет корректна, например, поля не были заполнены или в поле электронной почты была введена почта, уже имеющаяся в базе данных, то на экране появится окно с уведомлением об ошибке. Пример такой ошибки показан на рисунке 22.

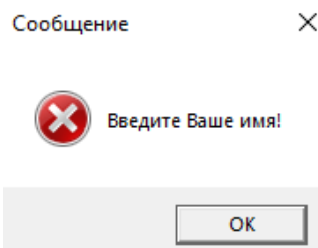


Рисунок 22 – Сообщение с уведомлением об ошибке

1.3.1.2 Шифрация и дешифрация файлов

После успешной регистрации пользователя, откроется главное окно программы, показанное на рисунке 23.

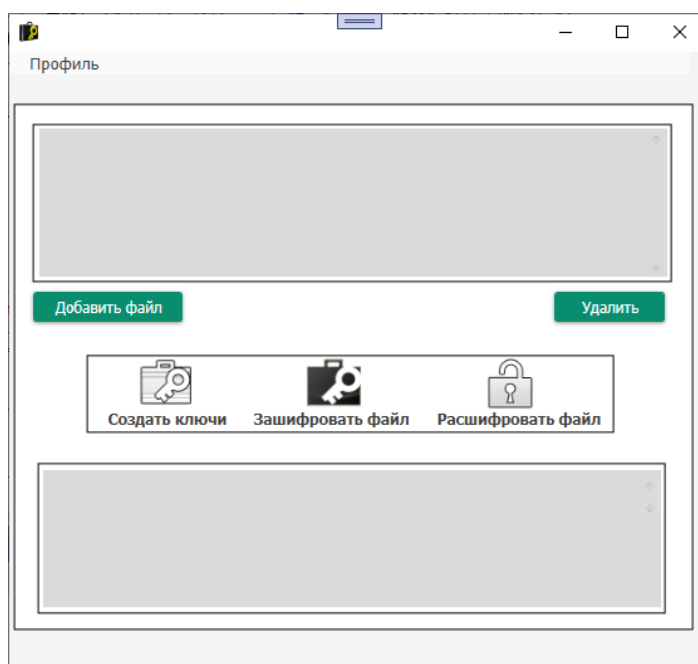


Рисунок 23 – Главное окно программы

С помощью кнопки «Добавить файл» добавляется путь до файла в список. При надобности файл можно удалить из списка, нажав кнопку «Удалить». Пример добавленного файла показан на рисунке 24.

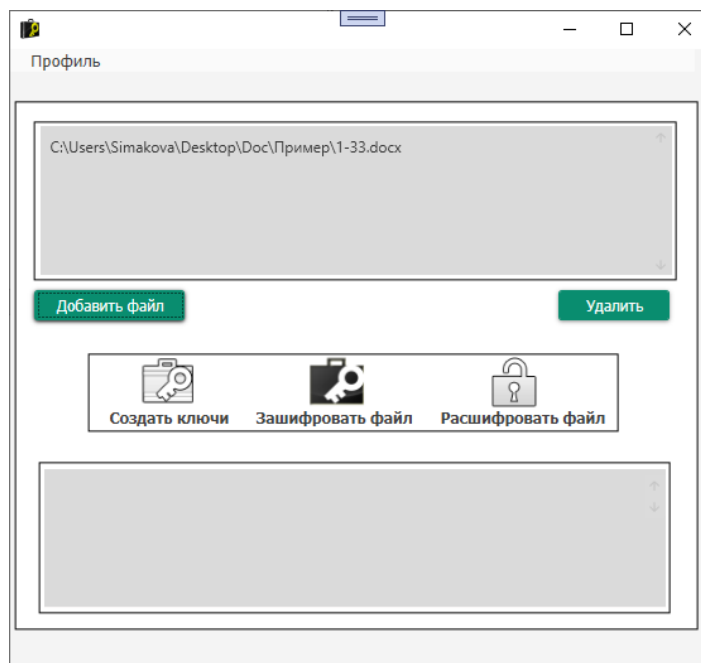


Рисунок 24 – Добавлен файл в список

Перед тем, как зашифровать файл, нужно сгенерировать пару ключей с помощью кнопки «Создать ключи». При нажатии появится окно, где автоматически создается открытый и закрытый ключ в виде XML-файла.

После генерации ключей, из списка выбирается файл, который нужно зашифровать. Если файл не выбран, то в нижнем поле выведется сообщение с предупреждением (рисунок 25).

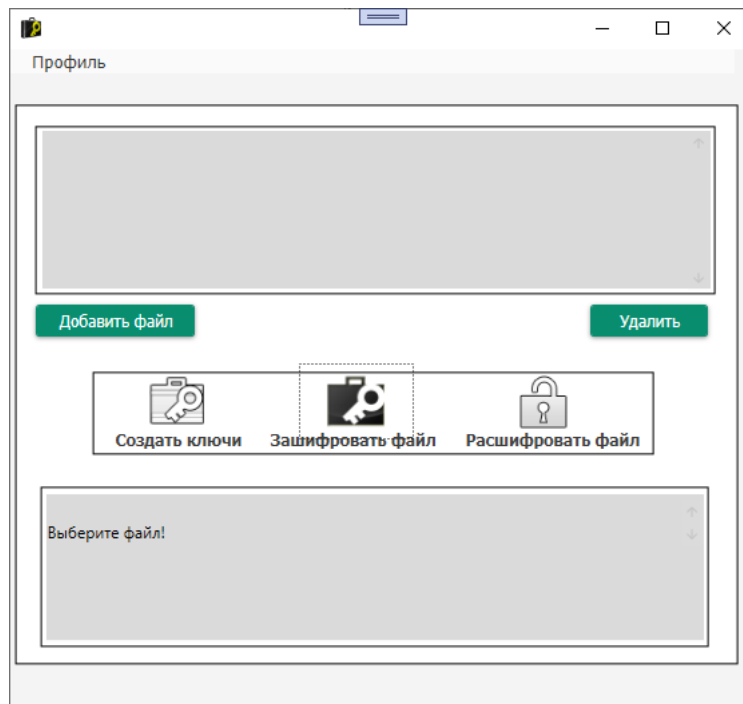


Рисунок 25 – Сообщение с предупреждением

Для шифрации нужно нажать кнопку «Зашифровать файл», после нажатия появится диалоговое окно для ввода пароля (рисунок 26).

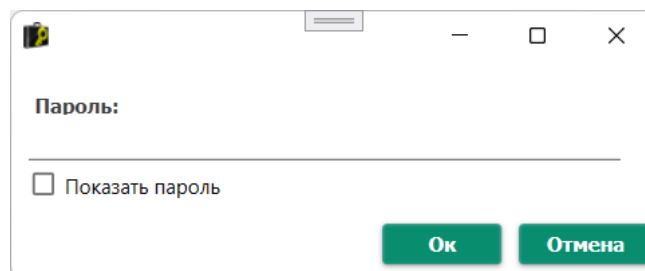


Рисунок 26 – Диалоговое окно для ввода пароля

В нижнее поле будет выведено сообщение о начале шифрации и сведения о примененных ключах, как на рисунке 27.

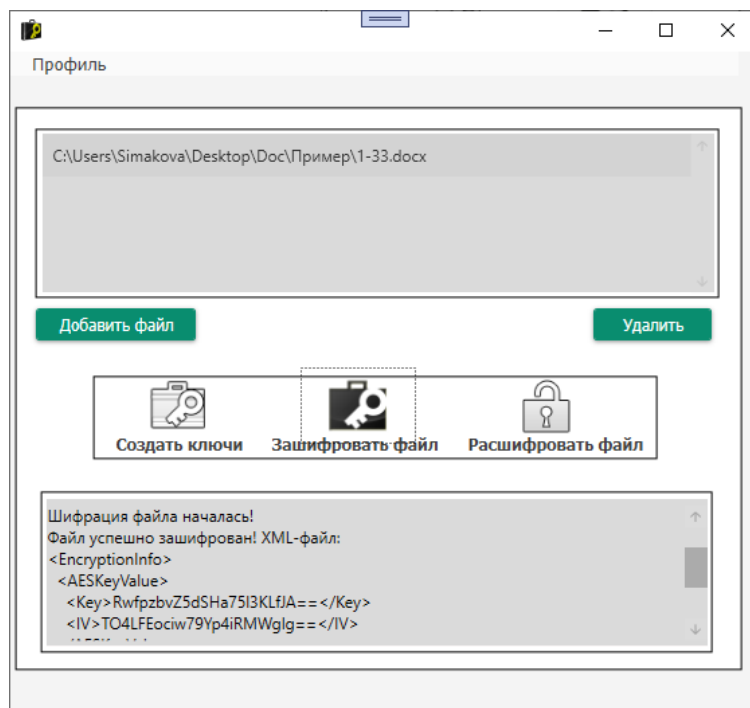


Рисунок 27 – Сообщение об успешной шифрации документа

Для дешифрации документа нужно добавить в список файл с расширением *.encrypted* с помощью кнопки «Добавить файл». После нажатия на нужный документ, требуется нажать кнопку «Расшифровать файл». При нажатии появится диалоговое окно для ввода пароля, который был использован для шифрования данного файла. Диалоговое окно представлено на рисунке 28.

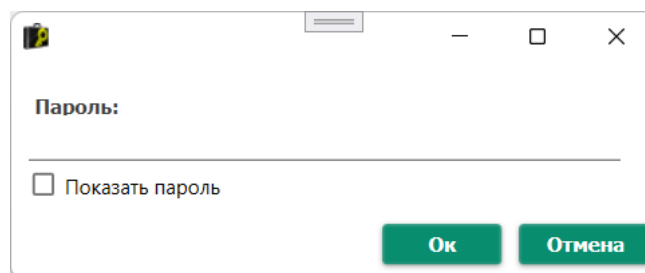


Рисунок 28 – Вспомогательное окно для ввода пароля

После откроется диалоговое окно, как на рисунке 29.

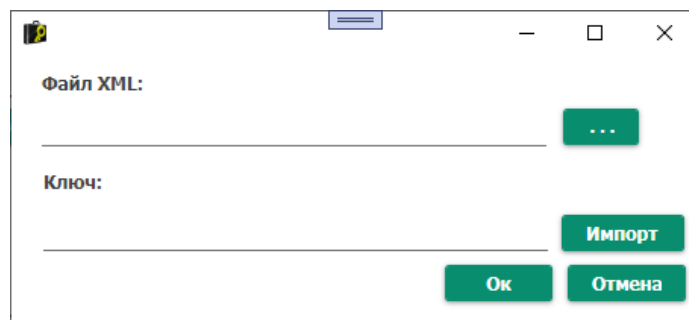


Рисунок 29 – Диалоговое окно при расшифровке файла

В данном окне необходимо добавить соответствующий документу XML-файл и добавляется закрытый ключ. После добавления нужно нажать кнопку «Ок» для начала дешифрации. Кнопка «Отмена» служит для возвращения в главное меню приложения.

Главное окно после дешифрации документа показано на рисунке 30.

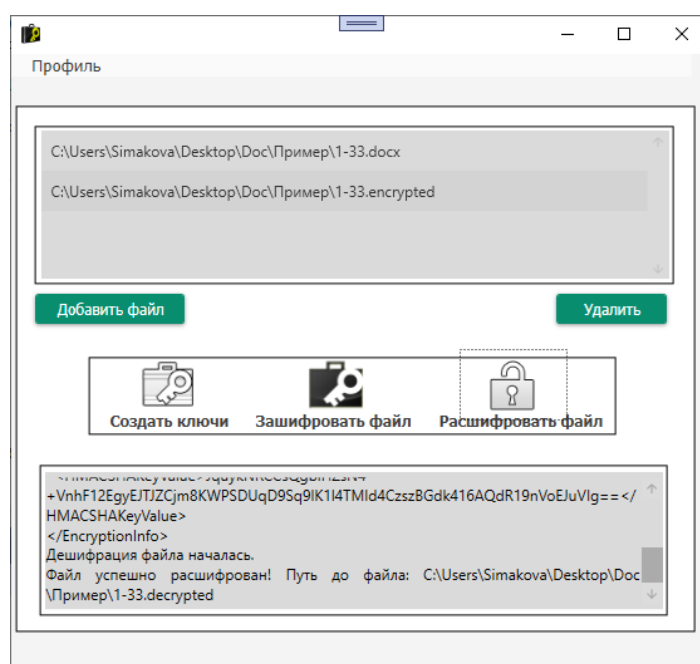


Рисунок 30 – Главное окно после дешифрации файла

С помощью кнопки «Профиль» можно выйти из текущего профиля, и зайти другому пользователю в приложение.

1.3.1 Отладка и тестирование

В процессе разработки программы было произведено тестирование программного обеспечения на предмет обнаружения некорректного ввода пользователем значений в окнах, отвечающих за вход в приложение и регистрацию. А также выполнено тестирование, направленное на установление целостности данных документа после шифрования и сравнение скорости работы алгоритма шифрования разработанной программы и программ аналогов.

1.3.1.1 Тестирование ввода данных

Ниже приведены таблицы 1 и 2, которые включают в себя набор тестов для окон *AuthWindow* и *MainWindow*, где приведена информация о полученных результатах, идее теста, входной информации и процедуре выполнения. После таблиц на рисунках 31-42 приведены результаты выполнения для каждой проверки.

Таблица 1 – Тестирование ввода данных в окно *AuthWindow*

Проверяется: Текстовые поля окна <i>AuthWindow</i>					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.1.1	Проверить ввод значения только числового формата в текстовое поле «Введите логин»	123	Ввести данные в текстовое поле, нажать кнопку «Войти»	Текстовое поле «Введите логин» выделится красным цветом	Совпадает с ожидаемым
ПФ.1.2	Проверить ввод четырехзначного значения	asdd	Ввести данные в текстовое	Текстовое поле «Введите	Совпадает с ожидаемым

Проверяется: Текстовые поля окна AuthWindow

Документ: Глава 3, пункт 1

№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
	текстового формата в текстовое поле «Введите логин»		поле, нажать кнопку «Войти»	логин» выделится красным цветом	
ПФ.1.3	Проверить ввод корректного значения только в текстовое поле «Введите логин»	simakova16	Ввести данные в текстовое поле, нажать кнопку «Войти»	Текстовое поле «Введите пароль» выделится красным цветом	Совпадает с ожидаемым
ПФ.1.4	Проверить ввод корректного значения только в текстовое поле «Введите пароль»	11111111	Ввести данные в текстовое поле, нажать кнопку «Войти»	Текстовое поле «Введите логин» выделится красным цветом	Совпадает с ожидаемым
ПФ.1.5	Проверить ввод корректных значений в оба поля	simakova16 11111111	Ввести данные в текстовые поля, нажать кнопку «Войти»	Появится окно с уведомлением «Вход выполнен успешно!»	Совпадает с ожидаемым

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

43

Проверяется: Текстовые поля окна AuthWindow					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.1.6	Проверить ввод четырехзначного значения в текстовое поле «Введите пароль»	1111	Ввести данные в текстовые поля, нажать кнопку «Войти»	Текстовое поле «Введите пароль» выделится красным цветом	Совпадает с ожидаемым

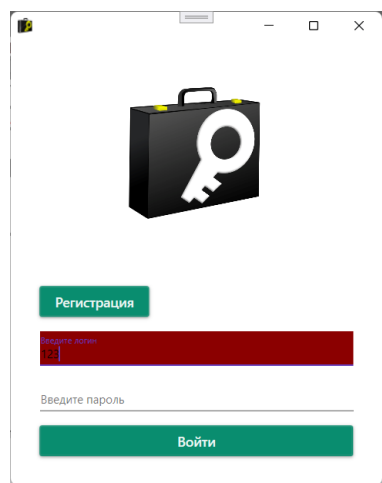


Рисунок 31 – ПФ.1.1

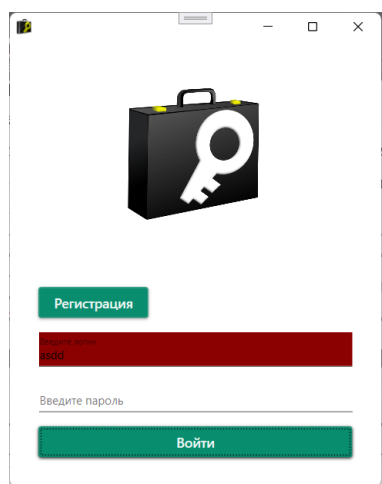


Рисунок 32 – ПФ.1.2

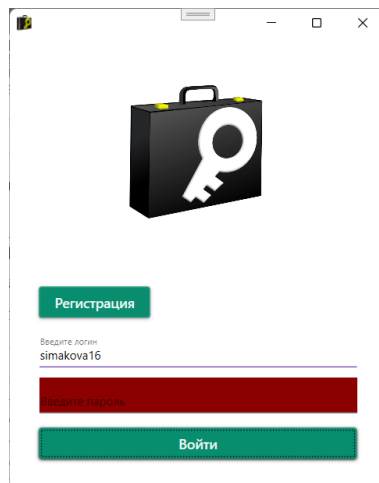


Рисунок 33 – ПФ.1.3

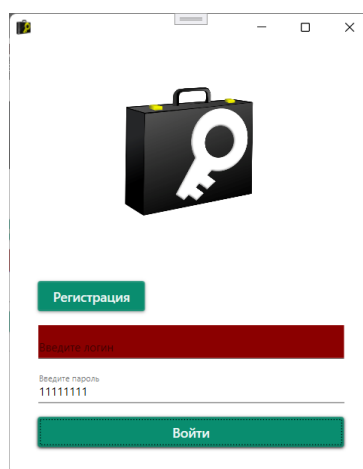


Рисунок 34 – ПФ.1.4

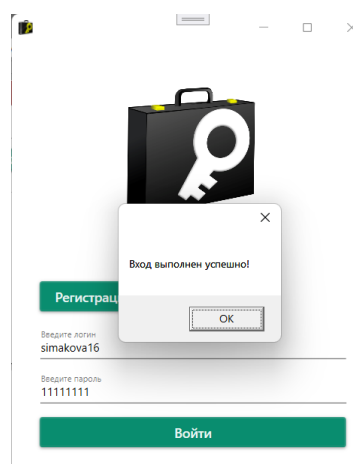


Рисунок 35 – ПФ.1.5

Изм.	Лист	№ докум.	Подпись	Дата

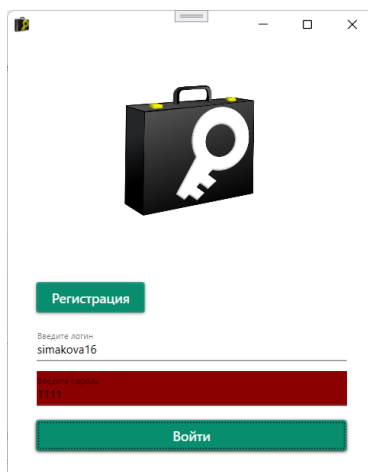


Рисунок 36 – ПФ.1.6

Таблица 2 – Тестирование ввода данных в окно MainWindow

Проверяется: Текстовые поля окна MainWindow					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.2.1	Проверить ввод значения только числового формата в текстовое поле «Введите логин»	1234	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Текстовое поле «Введите логин» выделится красным цветом	Совпадает с ожидаемым
ПФ.2.2	Проверить ввод четырехзначного значения текстового формата в текстовое поле «Введите логин»	abcd	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Текстовое поле «Введите логин» выделится красным цветом	Совпадает с ожидаемым
ПФ.2.3	Проверить ввод четырехзначного	simakova16	Ввести данные в текстовое поле,	Текстовое поле	Совпадает с ожидаемым

Проверяется: Текстовые поля окна MainWindow

Документ: Глава 3, пункт 1

№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
	значения в текстовое поле «Введите пароль»	1111	нажать кнопку «Зарегистрироваться»	«Введите пароль» выделится красным цветом	
ПФ.2.4	Проверить ввод неодинаковых значений в поля «Введите пароль» и «Повторите пароль»	111111 11 111111 1	Ввести данные в текстовые поля, нажать кнопку «Зарегистрироваться»	Текстовое поле «Повторите пароль» выделится красным цветом	Совпадает с ожидаемым
ПФ.2.5	Проверить ввод значения, не содержащего знак @, в поле «Email»	simako va16	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Текстовое поле «Email» выделится красным цветом	Совпадает с ожидаемым
ПФ.2.6	Проверить поведение программы при корректно заполненных полях	simako va16 1111111 1 1111111 1 simakova@mail.ru	Ввести данные в текстовые поля, нажать кнопку «Зарегистрироваться»	Появится окно с уведомлением «Регистрация прошла успешно!»	Совпадает с ожидаемым

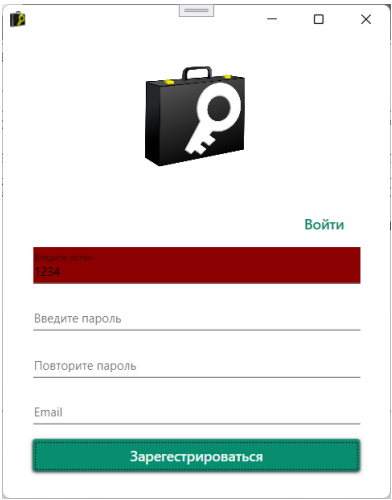


Рисунок 37 – ПФ.2.1

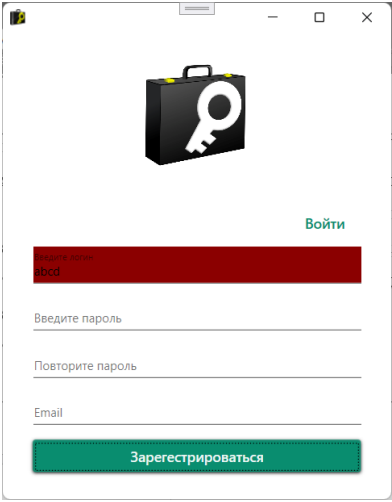


Рисунок 38 – ПФ.2.2

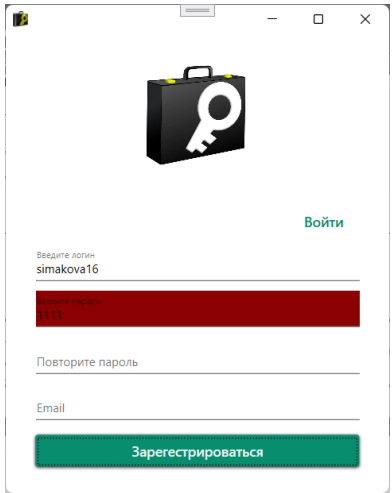


Рисунок 39 – ПФ.2.3

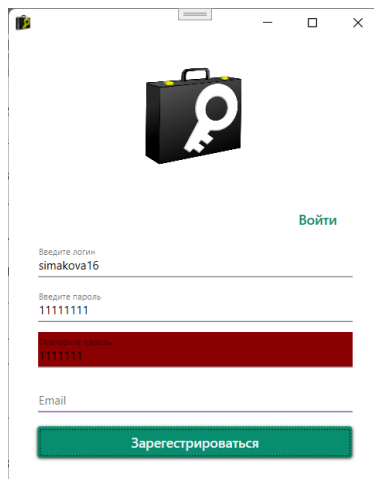


Рисунок 40 – ПФ.2.4

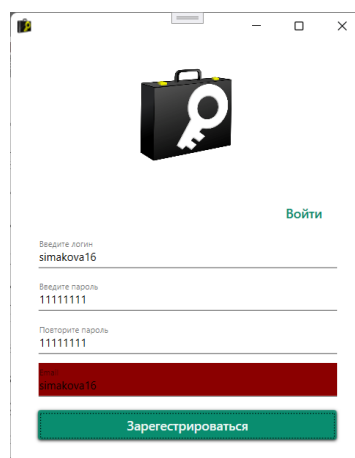


Рисунок 41 – ПФ.2.5

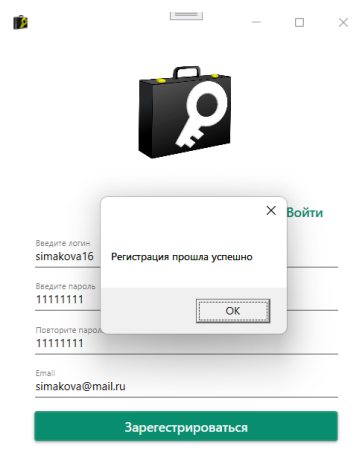


Рисунок 42 – ПФ.2.6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

49

В результате выполненных тестов видно, что исключения при вводе некорректных значений в текстовые поля обработаны, что помогает пользователю правильно внести свои данные.

1.3.1.2 Тестирование целостности файлов при шифровании

Ниже приведена таблица 3, включающая набор тестов, и результаты, расположенные после нее на рисунках 43, 44.

Таблица 3 – Тестирование целостности файлов при шифровании

Проверяется: Целостность файлов					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.3.1	Сравнить размер исходного файла, зашифрованного и расшифрованного	test.doc x	Зашифровать исходный файл, расшифровать получившийся зашифрованный файл	Размер файлов будет одинаковым	Совпадает с ожидаемым
ПФ.3.2	Сравнить содержимое исходного и расшифрованного файла	test.doc x	Зашифровать исходный файл, расшифровать получившийся зашифрованный файл	Содержимое файлов будет одинаковым	Совпадает с ожидаемым

test.decrypted	25.05.2022 19:37	Файл "DECRYPTED"	13 КБ
test.docx	30.04.2022 22:56	Документ Microso...	13 КБ
test.encrypted	25.05.2022 19:37	Файл "ENCRYPTED"	13 КБ
test.manifest.xml	25.05.2022 19:37	Документ XML	2 КБ

Рисунок 43 – ПФ.3.1

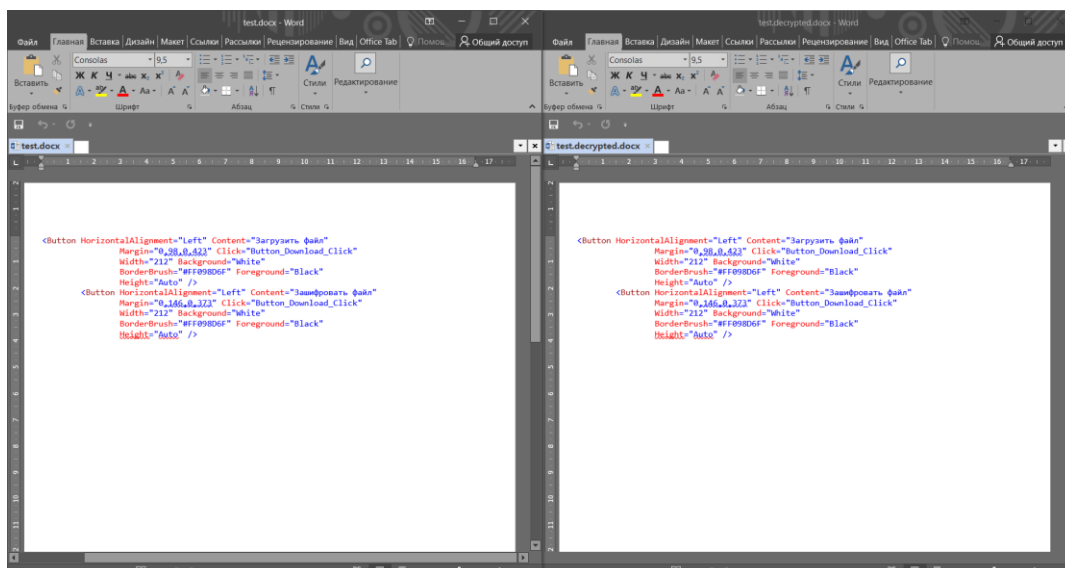


Рисунок 44 – ПФ.3.2

Исходя из результатов выполненных тестов, можно отметить, что при шифрации и дешифрации документов целостность содержимого не нарушается.

1.3.1.3 Тестирование скорости шифрования документов

В таблице 4 расположены тесты, направленные на сравнение скорости алгоритмов программ. Результаты проверок расположены ниже таблицы на рисунках 45-47.

Таблица 4 – Тестирование скорости шифрования документов

Проверяется: Скорость шифрования документов					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ.4.1	Сравнить скорость шифрования документа разработанной программой и	Файл образа диска размер 3,66 ГБ:	Зашифровать исходный файл, зафиксировать начало и окончание шифрования	Время шифрования разработанной программой	Совпадает с ожидаемым

Проверяется: Скорость шифрования документов					
Документ: Глава 3, пункт 1					
№ теста	Идея теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
	программами аналогами	ru_windows_10_multiple_editions_version_1511_x64_dvd_7223785.iso		будет меньше, чем программам и аналогам	

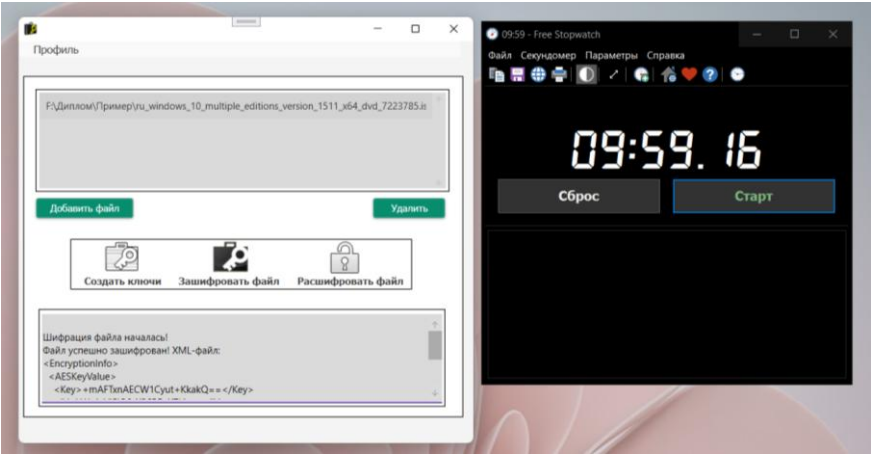


Рисунок 45 – ПФ.4.1

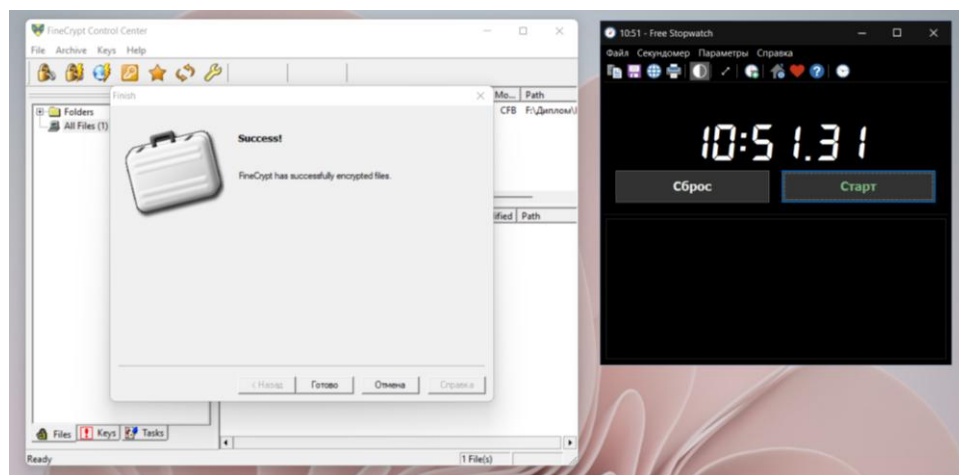


Рисунок 46 – ПФ.4.1

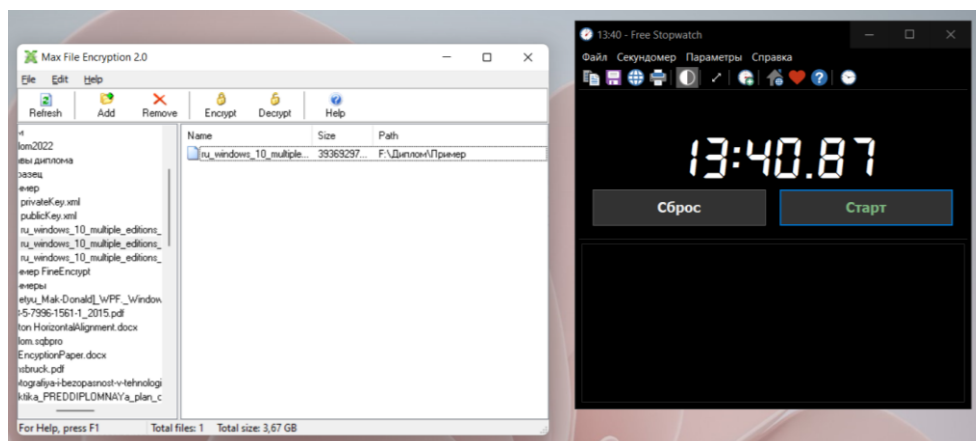


Рисунок 47 – ПФ.4.2

Полученный результат в ходе сравнения стоит рассматривать с учётом погрешности в ± 10 с. В результате, можно сделать вывод, что, учитывая погрешность измерений, применение комбинированного шифрования, уменьшает скорость работы алгоритма, что приводит к меньшим затратам времени при работе с программой.

2 Безопасность жизнедеятельности

Реализация любого проекта, будь то полноценный производственный процесс или работа над конкретной задачей, требует не только экономических затрат и умственных или физических усилий, но и обеспечения труда в безопасных для здоровья работников условиях. Для того, чтобы предоставить равные условия для работы, была разработана система правил и мер, которая легла в основу целой науки, основной целью которой стали систематизация знаний и обучение квалифицированных работников охране труда. Безопасность жизнедеятельности представляет собой свод правил, важнейшей целью которых является разработка и обновление мер по предупреждению производственного травматизма и заболеваемости.

В случае работы на ЭВМ, основными факторами, влияющими на работников, будут особенности организации рабочего пространства, удобство локации, а также самого места работы, условиями внутреннего расположения человека в помещении (освещение рабочего места, уровень шума, качеством техники и др.), общий характер работы.

В данной главе будут рассмотрены опасности, которые могут возникнуть в процессе реализации данного проекта, пожарные и санитарные нормы, применяемые к помещению, где должен реализовываться проект, санитарно-гигиеническая характеристика места разработки проекта, а также основные требования электробезопасности.

2.1 Характеристика потенциальных опасностей и вредностей

Для того, чтобы определить основные потенциальные опасности, присутствующие при разработке данного проекта, нужно определить, в каких условиях может производиться данный проект. Данные опасности можно определить, проведя анализ рисков, составленный на основе практического опыта. Так, среди главных опасностей для здоровья

человека согласно ГОСТу 12.0.003-04 были определены факторы, часть из которых может особенно иметь влияние на исполнителя. Среди них:

- повышенная или пониженная температура/влажность окружающей среды, повышенный уровень шума;

- повышенное напряжение в электросети;

- повышенный уровень электромагнитных излучений - электромагнитное и электростатическое поля, акустический шум, изменение ионного состава воздуха и параметров микроклимата в помещении также могут влиять на процесс работы;

- недостаточная освещенность рабочего места и зрительное напряжение;

- монотонность трудового процесса и нервно-эмоциональные перегрузки - нервно-эмоциональное напряжение при работе за персональным компьютером возникает вследствие дефицита времени, большого объема и плотности информации;

- длительное нахождение в одном положении, зачастую неправильном - повышенные статические и динамические нагрузки у пользователей ПК приводят к жалобам на боли в спине, шейном отделе позвоночника и руках. Из всех недомоганий, обусловленных работой на компьютерах, чаще встречаются те, которые связаны с использованием клавиатуры и неправильном положении при сидячей работе.

2.2 Санитарно-гигиеническая характеристика производства

Одними из важнейших требований при организации рабочего места сотрудника, независимо от его должности, являются обеспечение безопасных, подходящих под конкретного человека, условий для работы, а также недопущение возникновения заболеваний и несчастных случаев на рабочем месте. Общие требования к организации рабочих мест регулируются Трудовым кодексом, санитарно-эпидемиологическими

правилами и нормативами (СанПиН), а также другими правовыми документами.

В настоящее время действуют СанПиН 2.2.4.335916 «Санитарно-эпидемиологические требования к физическим факторам на рабочих местах», утвержденные Постановлением Главного государственного санитарного врача РФ от 21.06.2016 № 81 (предназначены для граждан, состоящих в трудовых отношениях, индивидуальных предпринимателей и юридических лиц). Этим документом установлены санитарно-эпидемиологические требования к физическим факторам на рабочих местах и источникам этих физических факторов, а также требования к организации контроля, методам измерения физических факторов на рабочих местах и мерам профилактики вредного воздействия физических факторов на здоровье работающих.

В отношении офисных работников действуют также следующие санитарные нормы и правила:

- при работе с компьютером с плоским монитором рабочее место должно иметь площадь не менее 4,5 кв.м;
- при размещении рабочих мест с ПЭВМ расстояние между рабочими столами с видеомониторами должно быть не менее 2м; размещать рабочие столы следует таким образом, чтобы естественный свет падал на дисплей преимущественно слева;
- экран видеомонитора должен находиться на расстоянии 600 — 700мм от глаз пользователя, но не ближе 500мм с учетом размеров алфавитно-цифровых знаков и символов;
- рабочий стул (кресло) должен быть подъемно-поворотным, регулируемым по высоте и углам наклона сиденья и спинки, а также по расстоянию спинки от переднего края сиденья. При этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию. При выборе типа рабочего стула (кресла) следует

учитывать рост пользователя, характер и продолжительность работы с ПЭВМ. Поверхности сиденья, спинки и других элементов стула (кресла) должны быть полумягкими, с нескользящим, слабо электризующимся и воздухопроницаемым покрытием, обеспечивающим легкую очистку от загрязнений;

– для сидячих рабочих мест оптимальными параметрами микроклимата на рабочем месте являются: температура воздуха в холодный период года — от 22 до 24 °С, в теплый период года — от 23 до 25 °С; относительная влажность воздуха на постоянных рабочих местах — 40 — 60%, скорость движения воздуха — 0,1м/с.

2.3 Электробезопасность

При рассмотрении вопроса обеспечения электробезопасности разработчика необходимо выделить три основных фактора:

- электроустановки рабочего места программиста;
- вспомогательное электрооборудование;
- окружающая среда помещения.

Помещения, оборудованные вычислительной техникой, как правило, относятся к категории помещений без повышенной опасности так как:

- относительная влажность воздуха не превышает 75%;
- нет токопроводящей пыли и токопроводящих полов;
- температура не превышает длительное время 30 °С;
- отсутствует возможность одновременного прикосновения человека с имеющими соединение с землей металлическими конструкциями.

Таким образом, для предотвращения электротравматизма пользователя, необходимо соблюдать требования безопасности, как при работе с обычной бытовой техникой.

2.4 Пожарная безопасность

Меры пожарной безопасности к помещениям, которые по своему назначению можно отнести к офисам или кабинетам, определяют следующие нормы, правила, изложенные как в законодательной, так и в нормативно-технической базе, например, в Федеральном законе номер 69 «О пожарной безопасности», а также различных ГОСТах, СНиПах, НПБ и СП.

Помещение, в котором ведется разработка программного обеспечения, классифицируется как офисное. К причинам возникновения пожара электрического характера относятся короткие замыкания, перегрузки, большие переходные сопротивления, искрение и электрические дуги, статическое электричество. Основным фактором, от которого зависит пожарная безопасность помещения, является автоматическое средство обнаружения пожаров. В помещении офиса должны быть также установлены: пожарные извещатели автоматической сигнализации, а также иметься воздушно-пенные, воздушно-эмульсионные, порошковые огнетушители.

2.5 Расчетно-аналитическая часть

Расчет освещенности рабочего места сводится к выбору системы освещения, определению необходимого числа светильников, их типа и размещения. Процесс работы программиста происходит в таких условиях, когда естественное освещение недостаточно или отсутствует. Исходя из этого, рассчитаем параметры искусственного освещения.

Расчет освещения производится для комнаты площадью 42 м²: длина - 7 м, ширина - 6 м, высота - 3 м. Равномерное освещение горизонтальной рабочей поверхности достигается при определенных отношениях расстояния между центрами светильников L , м ($L=1,75 \cdot H$) к высоте их подвеса над рабочей поверхностью H , м.

Изм.	Лист	№ докум.	Подпись	Дата

$$L = 1,75 * H = 1,75 * 3 = 5,25 \text{ м.}$$

Число светильников с люминесцентными лампами (ЛЛ):

$$N = \frac{S}{L * M},$$

где S - площадь помещения, м;

M - расстояние между параллельными рядами, м.

$$N = \frac{6 * 7}{5,25 * 2} = 4.$$

Для достижения равномерной горизонтальной освещенности светильники с ЛЛ рекомендуется располагать сплошными рядами, параллельно стенам с окнами или длинным сторонам помещения.

Для расчета общего равномерного освещения горизонтальной рабочей поверхности используют метод светового потока, учитывающий световой поток, отраженный от потолка и стен.

Для определения количества светильников определим световой поток, падающий на поверхность по формуле:

$$F_p = \frac{E_n * K * S * Z}{N * \mu},$$

где F_p - рассчитываемый световой поток, Лм;

E_n - нормированная минимальная освещенность, Лк (определяется по таблице). Работу программиста, в соответствии с этой таблицей, можно отнести к разряду точных работ, следовательно, минимальная освещенность будет $E = 300$ Лк при газоразрядных лампах;

Z - отношение средней освещенности к минимальной (обычно принимается равным 1.1-1.2, пусть $Z = 1.1$);

K - коэффициент запаса, учитывающий уменьшение светового потока лампы в результате загрязнения светильников в процессе эксплуатации (его значение определяется по таблице коэффициентов запаса для различных помещений и в нашем случае $K = 1.5$);

μ - коэффициент использования, (выражается отношением светового потока, падающего на расчетную поверхность, к суммарному потоку всех ламп и исчисляется в долях единицы; зависит от характеристик светильника, размеров помещения, окраски стен и потолка, характеризуемых коэффициентами отражения от стен (P_c) и потолка (P_n)), значение коэффициентов P_c и P_n определим по таблице зависимостей коэффициентов отражения от характера поверхности: $P_c=30\%$, $P_n=50\%$.

Значение μ определим по таблице коэффициентов использования различных светильников. Для этого вычислим индекс помещения по формуле:

$$I = \frac{S}{H(A + B)},$$

где S - площадь помещения, $S = 42 \text{ м}^2$;

h - расчетная высота подвеса, $h = 3 \text{ м}$;

A - ширина помещения, $A = 6 \text{ м}$;

B - длина помещения, $B = 7 \text{ м}$.

Подставив значения получим:

$$I = \frac{42}{3 * (6 + 7)} = 1,08.$$

Зная индекс помещения I , P_c и P , по таблице находим $\mu = 0.3$
Подставим все значения в формулу для определения светового потока F :

$$F = \frac{75 * 1,5 * 42 * 1,1}{4 * 0,3} = 4331,25.$$

Учитывая, что в светильнике по 4 лампы, световой поток одной лампы будет равен. Для освещения выбираем люминесцентные лампы типа ЛБ20, световой поток которых $F = 1180 \text{ Лм}$.

Рассчитаем действительное освещение. В рабочем помещении имеется 4 светильника, в каждом из которых используется по 4 лампы ЛДЦ20. Рассчитаем световой поток:

$$F = 820 * 4 * 4 = 3280 \text{ лк.}$$

Затем рассчитаем значение нормированной освещенности:

$$E = \frac{F * N * z}{S * Z * K} = 227 \text{ лк.}$$

Полученное значение нормированной освещенности оказалось меньше нормированной минимальной освещенности. Рекомендуется заменить используемые лампы на лампы с большим световым потоком.

					ПГУ 1.090301.37.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

Заключение

В рамках данной выпускной квалификационной работы было разработано программное обеспечение для шифрования данных. Реализованное приложение позволяет зашифровывать и расшифровывать данные, применяя комбинированное шифрование, использующее алгоритмы *AES-128* и *RSA-2048*.

Приложение написано на языке программирования *C#*, с использованием технологии *Windows Presentation Form*, которое имеет интуитивно понятный графический интерфейс

В ходе работы был получен опыт применения объектно-ориентированного подхода программирования и проектирования приложения с многоуровневой архитектурой. Была изучена документация по применению пространства имен *System.Security.Cryptography* языка *C#* на основе фреймворка *.NET Framework* для реализации алгоритмов шифрования. Также была освоена платформа для разработки *UI Windows Presentation Form* и декларативный язык разметки *XAML*. Был получен навык работы с базой данных *SQLite* вместе со сторонней программой *DB Browser*.

В дальнейшем приложение можно улучшить посредством расширения функционала, добавив шифрование не только отдельных документов, но и папок с вложенными файлами. Также можно добавить возможность создания сейфов, т.е. специальных зашифрованных контейнеров, которые после монтирования можно использовать как обычный диск. Можно реализовать функцию подписания файлов цифровой подписью и возможность проверки подписи файлов. Также можно реализовать возможность кодирования съемных носителей, таких как диски и *USB-накопители*.

Список используемых источников

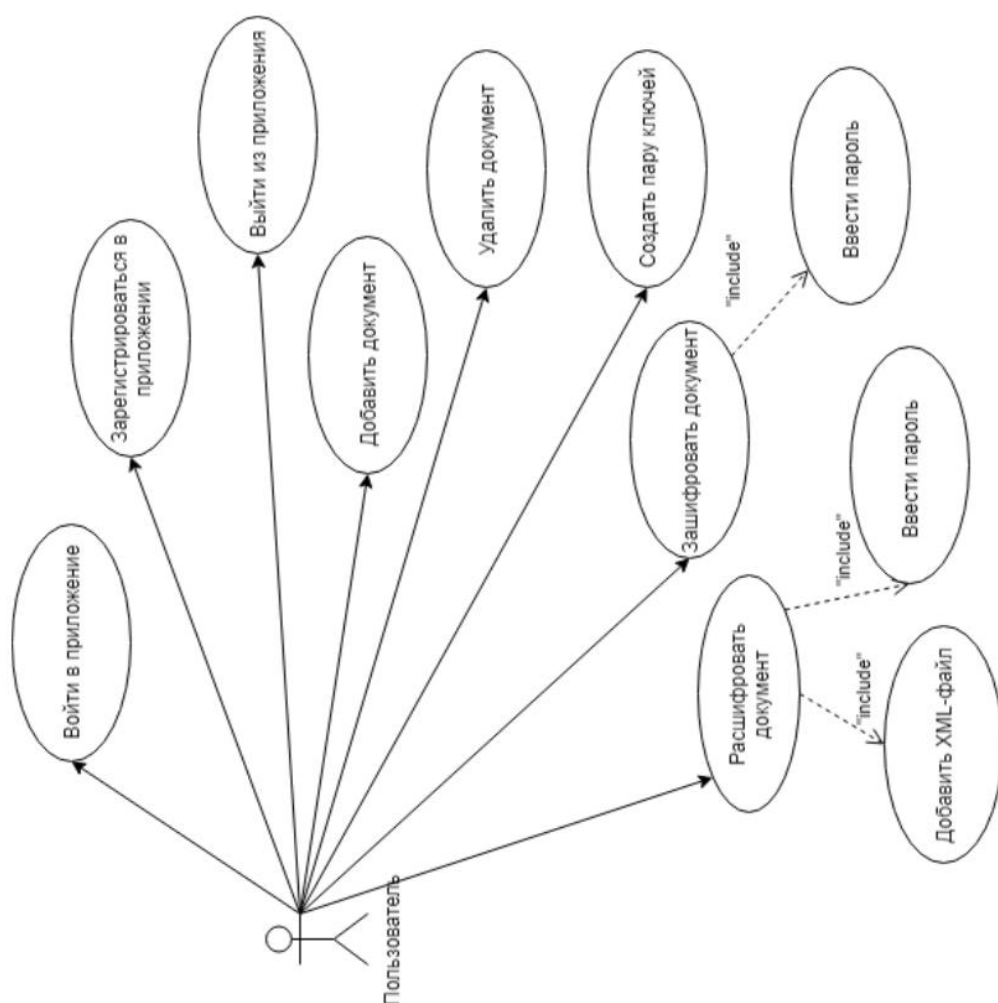
1. Спиричева, Н. Р., Алгоритмы блочной криптографии учебно-методическое пособие / Н. Р. Спиричева. - Екатеринбург : Изд-во Урал, ун-та, 2013.-78 с.
2. Разинков Е.В. Теория чисел и асимметричная криптография / Е.В. Разинков. – Казань: Казан. ун-т, 2020. – 53 с.
3. Документация по C#: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>.
4. Торстейнсон П, Криптография и безопасность в технологии .NET / П. Торстейнсон, Г. А. Ганеш ; пер. с англ.—4-е изд., электрон.—М. : Лаборатория знаний, 2020.—482 с.
5. Мак-Доналд, М. WPF: *Windows Presentation Foundation* в .NET 4.5 / М. Мак-Доналд. – : Вильямс Издательский дом, 2013. – 142-158 с. URL: <https://booksee.org/book/1330069>
6. Учебник по XAML [Электронный ресурс] / *CoderLessons* URL: <https://coderlessons.com/tutorials/microsoft-technologies/vyuchit-xaml/uchebnik-ro-xaml>, свободный. (Дата 06.04.2021 г.)

Приложение А

(Обязательное)

Программные обеспечение для шифрования данных
UML-диаграммы

Приложение А.1 – UML – диаграмма прецедентов



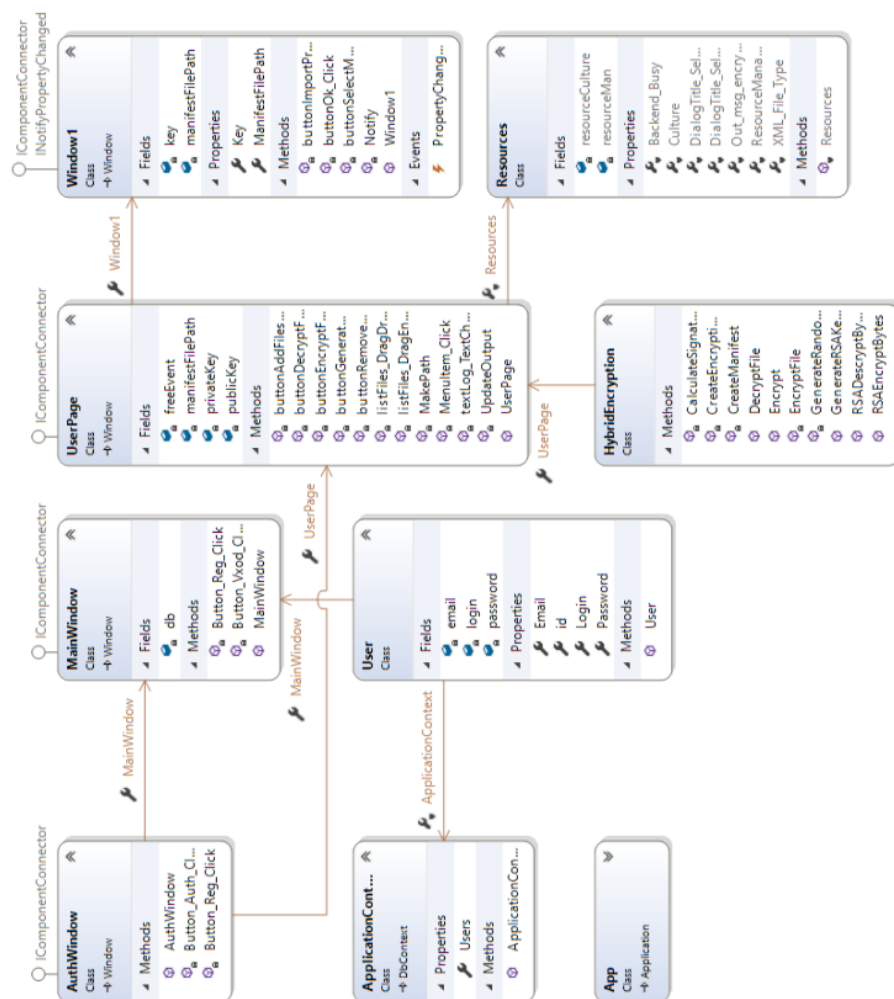
Приложение А.2 – UML – диаграмма классов 1

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

65



Приложение А.3 – UML – диаграмма классов 2



Приложение А.4 – UML – диаграмма классов 3

Изм.	Лист	№ докум.	Подпись	Дата



Приложение Б

(Обязательное)

Программные обеспечение для шифрования данных

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

68

Файл HybridEncryption.cs

```
namespace Diplom2022.Encrypt
{
    class HybridEncryption
    {
        public UserPage UserPage
        {
            get => default;
            set
            {
            }
        }

        // RSA key pair
        public static void GenerateRSAKeyPair(out string
publicKey, out string privateKey)
        {
            RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(2048);

            publicKey = rsa.ToXmlString(false);
            privateKey = rsa.ToXmlString(true);
        }

        //Encrypt a file
        //Returns encryption information including
        symmetrical keys for data encryption and signature
        public static string Encrypt(
            string plainFilePath,
            string encryptedFilePath,
            string manifestFilePath,
            string rsaKey)
        {
            //генерируем ключи
            byte[] signatureKey = GenerateRandom(64);
            byte[] encryptionKey = GenerateRandom(16);
            byte[] encryptionIV = GenerateRandom(16);

            EncryptFile(plainFilePath, encryptedFilePath,
encryptionKey, encryptionIV);

            byte[] signature =
CalculateSignature(encryptedFilePath, signatureKey);

            CreateManifest(signature, signatureKey,
encryptionKey, encryptionIV, rsaKey, manifestFilePath);
        }
    }
}
```

```

        return CreateEncryptionInfoXml(signatureKey,
encryptionKey, encryptionIV);
    }

    //Create encryption information in the form of xml
string
    //Returns xml string containing key informations
    private static string CreateEncryptionInfoXml(byte[]
signatureKey, byte[] encryptionKey, byte[] encryptionIV)
    {
        string template = "<EncryptionInfo>" +
            "<AESKeyValue>" +
            "<Key/>" +
            "<IV/>" +
            "</AESKeyValue>" +
            "<HMACSHAKeyValue/>" +
            "</EncryptionInfo>";

        XDocument doc = XDocument.Parse(template);

        doc.Descendants("AESKeyValue").Single().Descendants("Key").Single().Value = Convert.ToBase64String(encryptionKey);

        doc.Descendants("AESKeyValue").Single().Descendants("IV").Single().Value = Convert.ToBase64String(encryptionIV);

        doc.Descendants("HMACSHAKeyValue").Single().Value = Convert.ToBase64String(signatureKey);
        return doc.ToString();
    }

    //Generate random byte array
    //Returns random byte array
    private static byte[] GenerateRandom(int length)
    {
        byte[] bytes = new byte[length];
        using (RNGCryptoServiceProvider random = new RNGCryptoServiceProvider())
        {
            random.GetBytes(bytes);
        }

        return bytes;
    }

    //Encrypt a file with AES
    private static void EncryptFile(
        string plainFilePath,
        string encryptedFilePath,

```

```

        byte[] key,
        byte[] iv)
    {
        using (AesCryptoServiceProvider aes = new
AesCryptoServiceProvider())
        {
            aes.KeySize = 128;
            aes.Key = key;
            aes.IV = iv;

            ICryptoTransform encryptor =
aes.CreateEncryptor(aes.Key, aes.IV);

            using (FileStream plain =
File.Open(plainFilePath, FileMode.Open, FileAccess.Read,
FileShare.Read))
            {
                using (FileStream encrypted =
File.Open(encryptedFilePath, FileMode.Create,
FileAccess.Write, FileShare.None))
                {
                    using (CryptoStream cs = new
CryptoStream(encrypted, encryptor, CryptoStreamMode.Write))
                    {
                        plain.CopyTo(cs);
                    }
                }
            }
        }
    }

//Decrypt a file with AES
public static void DecryptFile(
    string plainFilePath,
    string encryptedFilePath,
    byte[] key,
    byte[] iv)
{
    using (AesCryptoServiceProvider aes = new
AesCryptoServiceProvider())
    {
        aes.KeySize = 128;
        aes.Key = key;
        aes.IV = iv;
        ICryptoTransform decryptor =
aes.CreateDecryptor(aes.Key, aes.IV);
        using (FileStream plain =
File.Open(plainFilePath, FileMode.Create, FileAccess.Write,
FileShare.None))

```

```

        {
            using (FileStream encrypted =
File.Open(encryptedFilePath, FileMode.Open, FileAccess.Read,
FileShare.Read))
            {
                using (CryptoStream cs = new
CryptoStream(plain, decryptor, CryptoStreamMode.Write))
                {
                    encrypted.CopyTo(cs);
                }
            }
        }
    }

    //Encrypt byte array with RSA
    //Returns encrypted array
    public static byte[] RSAEncryptBytes(byte[] datas,
string keyXml)
    {
        byte[] encrypted = null;
        using (RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(2048))
        {
            rsa.FromXmlString(keyXml);
            encrypted = rsa.Encrypt(datas, true);
        }

        return encrypted;
    }

    //Decrypt byte array with RSA
    //Returns decrypted array
    public static byte[] RSADecryptBytes(byte[] datas,
string keyXml)
    {
        byte[] decrypted = null;
        using (RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(2048))
        {
            rsa.FromXmlString(keyXml);
            decrypted = rsa.Decrypt(datas, true);
        }

        return decrypted;
    }

    //Calculate file signature
    //Returns signature array

```

```

        private static byte[] CalculateSignature(string
filePath, byte[] key)
        {
            byte[] sig = null;
            using (HMACSHA256 sha = new HMACSHA256(key))
            {
                using (FileStream f = File.Open(filePath,
FileMode.Open, FileAccess.Read, FileShare.Read))
                {
                    sig = sha.ComputeHash(f);
                }
            }
            return sig;
        }

//Create manifest file of a encrypted package, used
for backend parsing
private static void CreateManifest(
    byte[] signature,
    byte[] signatureKey,
    byte[] encryptionKey,
    byte[] encryptionIv,
    string rsaKey,
    string manifestFilePath)
{
    string template = "<DataInfo>" +
        "<Encrypted>True</Encrypted>" +
        "<KeyEncryption algorithm='RSA-2048'>" +
        "</KeyEncryption>" +
        "<DataEncryption algorithm='AES-128'>" +
        "<AESEncryptedKeyValue>" +
        "<Key/>" +
        "<IV/>" +
        "</AESEncryptedKeyValue>" +
        "</DataEncryption>" +
        "<DataSignature algorithm='HMACSHA256'>" +
        "<Value />" +
        "<EncryptedKey />" +
        "</DataSignature>" +
        "</DataInfo>";

    XDocument doc = XDocument.Parse(template);

    doc.Descendants("DataEncryption").Single().Descendants("AESE
ncryptedKeyValue").Single().Descendants("Key").Single().Valu
e =
    System.Convert.ToBase64String(RSAEncryptBytes(encryptionKey,
rsaKey));

    doc.Descendants("DataEncryption").Single().Descendants("AESE

```

```

ncryptedKeyValue").Single().Descendants("IV").Single().Value
=
System.Convert.ToBase64String(RSAEncryptBytes(encryptionIv,
rsaKey));

doc.Descendants("DataSignature").Single().Descendants("Value
").Single().Value =
System.Convert.ToBase64String(signature);

doc.Descendants("DataSignature").Single().Descendants("Encry
ptedKey").Single().Value =
System.Convert.ToBase64String(RSAEncryptBytes(signatureKey,
rsaKey));

        doc.Save(manifestFilePath);
    }
}
}

```

Файл UserPage.xaml.cs

```

namespace Diplom2022
{
    /// <summary>
    /// Interaction logic for UserPage.xaml
    /// </summary>
    public partial class UserPage : Window
    {
        public UserPage()
        {
            InitializeComponent();

            this.freeEvent = new EventWaitHandle(true,
EventResetMode.ManualReset);
        }

        public Window1 Window1
        {
            get => default;
            set
            {
            }
        }

        internal Properties.Resources Resources
        {
            get => default;
            set
            {
            }
        }
    }
}

```

```

    }

    private EventWaitHandle freeEvent;
    private string privateKey;
    private string publicKey;
    private string manifestFilePath;

    private void buttonAddFiles_Click(object sender,
RoutedEventArgs e)
    {
        Microsoft.Win32.OpenFileDialog openFileDialog =
new Microsoft.Win32.OpenFileDialog();

        openFileDialog.Title = "Select your File(s)";
        openFileDialog.CheckFileExists = true;
        openFileDialog.CheckPathExists = true;
        openFileDialog.Multiselect = true;

        if (openFileDialog.ShowDialog() == true)
        {
            var files = openFileDialog.FileNames;

            if (files != null && files.Length > 0)
            {
                foreach (var filePath in files)
                {
                    var items = listFiles.Items;
                    if (!items.Contains(filePath))
                        listFiles.Items.Add(filePath);
                    else
                        UpdateOutput(textLog, "Файл уже
был добавлен!", true);
                }
            }
        }

        private void buttonRemoveFiles_Click(object sender,
RoutedEventArgs e)
        {
            var selectedIndex = listFiles.SelectedIndex;
            if (selectedIndex != -1)
            {
                listFiles.Items.RemoveAt(selectedIndex);

                listFiles.SelectedIndex = selectedIndex <
listFiles.Items.Count ? selectedIndex : selectedIndex - 1;
                listFiles.Focus();
            }
        }
    }

```

```

        private void buttonEncryptFiles_Click(object sender,
RoutedEventArgs e)
        {
            var select = listFiles.SelectedItem;
            if (select != null)
            {
                if (!freeEvent.WaitOne(0))
                {

System.Windows.MessageBox.Show("Подождите пока закончится
шифрование файла!", Properties.Resources.Backend_Busy);
                    return;
                }

                if (string.IsNullOrEmpty(publicKey))
                {
                    UpdateOutput(textLog, "Для начала
шифрования файла создайте ключи!", true);
                    return;
                }

                string selectFilePath =
(string)listFiles.SelectedItem;
                string encryptedFilePath =
MakePath(selectFilePath, ".encrypted");
                string manifestFilePath =
MakePath(selectFilePath, ".manifest.xml");

                UpdateOutput(textLog, "Шифрация файла
началась!", true);

                var t = Task.Factory.StartNew(() =>
                {
                    freeEvent.Reset();
                    string s = HybridEncryption.Encrypt(
                        selectFilePath,
                        encryptedFilePath,
                        manifestFilePath,
                        publicKey);

                    freeEvent.Set();
                    UpdateOutput(textLog, "Файл успешно
зашифрован! XML-файл: " +
Properties.Resources.Out_msg_encrypt_success + "\r\n" + s,
true);
                });
            }
            else

```

```

        UpdateOutput(textLog, "Выберите файл!",
true);
    }

    private void buttonDecryptFiles_Click(object sender,
RoutedEventArgs e)
    {
        var select = listFiles.SelectedItem;
        if (select != null)
        {
            Window1 decryptPage = new Window1();
            decryptPage.ManifestFilePath =
manifestFilePath;
            Nullable<bool> result =
decryptPage.ShowDialog();

            if (result == true)
            {
                privateKey = decryptPage.Key;
                manifestFilePath =
decryptPage.ManifestFilePath;
            }

            if (!freeEvent.WaitOne(0))
            {

System.Windows.MessageBox.Show("Подождите пока закончиться
дешифрация файла!", Properties.Resources.Backend_Busy);
                return;
            }

            string encryptedFile =
(string)listFiles.SelectedItem;
            string rsaKey = privateKey;
            string plainFile = MakePath(encryptedFile,
".decrypted");
            string manifestFile = manifestFilePath;

            XDocument doc =
XDocument.Load(manifestFile);
            XElement aesKeyElement =
doc.Root.XPathSelectElement("./DataEncryption/AESEncryptedKe
yValue/Key");
            byte[] aesKey =
HybridEncryption.RSADecryptBytes(Convert.FromBase64String(a
esKeyElement.Value), rsaKey);
            XElement aesIvElement =
doc.Root.XPathSelectElement("./DataEncryption/AESEncryptedKe
yValue/IV");

```

```

        byte[] aesIv =
HybridEncryption.RSADecryptBytes(Convert.FromBase64String(a
esIvElement.Value), rsaKey);

        UpdateOutput(textLog, "Дешифрация файла
началась.", true);

        var t = Task.Factory.StartNew(() =>
        {
            freeEvent.Reset();
            HybridEncryption.DecryptFile(plainFile,
encryptedFile, aesKey, aesIv);
            freeEvent.Set();

            UpdateOutput(textLog, "Файл успешно
расшифрован! Путь до файла: "+ plainFile, true);
        });
    }
    else
        UpdateOutput(textLog, "Выберите файл!",
true);
}

    private static string MakePath(string plainFilePath,
string newSuffix)
    {
        string encryptedFileName =
System.IO.Path.GetFileNameWithoutExtension(plainFilePath) +
newSuffix;
        return
System.IO.Path.Combine(System.IO.Path.GetDirectoryName(plain
FilePath), encryptedFileName);
    }

    private void
UpdateOutput(System.Windows.Controls.TextBox tb, string
message, bool append)
    {
        this.Dispatcher.Invoke(new Action(() =>
        {
            string newMessage = append ? tb.Text +
"\r\n" + message : message;
            tb.Text = newMessage;
        })), null);
    }

    private void buttonGenerate_Click(object sender,
RoutedEventArgs e)
    {

```

```

        Microsoft.Win32.SaveFileDialog fbd = new
Microsoft.Win32.SaveFileDialog();
        fbd.FileName = "publicKey.xml";
        Nullable<bool> result = fbd.ShowDialog();
        if (result == true)
        {
            HybridEncryption.GenerateRSAKeyPair(out
publicKey, out privateKey);
            using (StreamWriter sw =
File.CreateText(fbd.FileName))
            {
                sw.Write(publicKey);
            }
        }

        Microsoft.Win32.SaveFileDialog dlg = new
Microsoft.Win32.SaveFileDialog();
        dlg.FileName = "privateKey.xml";
        Nullable<bool> resultTwo = dlg.ShowDialog();
        if (result == true)
        {
            using (StreamWriter sw =
File.CreateText(dlg.FileName))
            {
                sw.Write(privateKey);
            }
        }

        private void textLog_TextChanged(object sender,
TextChangedEventArgs e)
        {
            textLog.IsReadOnly = true;
        }

        private void MenuItem_Click(object sender,
RoutedEventArgs e)
        {
        }

        private void listFiles_DragDrop(object sender,
System.Windows.DragEventArgs e)
        {
            var fileList =
e.Data.GetData(System.Windows.DataFormats.FileDrop, false);

            var paths = fileList as IEnumerable<string>;
            if (paths != null)
                foreach (var path in paths)

```

```

        {
            listFiles.Items.Add(path);
        }
    }

    private void listFiles_DragEnter(object sender,
System.Windows.DragEventArgs e)
    {
        if
(e.Data.GetDataPresent(System.Windows.DataFormats.FileDrop))
            e.Effects =
System.Windows.DragDropEffects.Copy;
        else
            e.Effects =
System.Windows.DragDropEffects.None;
    }
}

```

Файл DecryptPage.xaml.cs

```

namespace Diplom2022
{
    /// <summary>
    /// Логика взаимодействия для Window1.xaml
    /// </summary>
    public partial class Window1 : Window,
INotifyPropertyChanged
    {
        public Window1()
        {
            InitializeComponent();
            this.DataContext = this;
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler
PropertyChanged;
        void Notify(string propName)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new
PropertyChangedEventArgs(propName));
        }
        #endregion

        private string key;
        private string manifestFilePath;
    }
}

```

```

public string ManifestFilePath
{
    get
    {
        return this.manifestFilePath;
    }
    set
    {
        if (value != this.manifestFilePath)
        {
            this.manifestFilePath = value;
            Notify("ManifestFilePath");
        }
    }
}

public string Key
{
    get
    {
        return this.key;
    }
    set
    {
        if (value != this.key)
        {
            this.key = value;
            Notify("Key");
        }
    }
}

private void buttonSelectManifest_Click(object
sender, RoutedEventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.DefaultExt = ".xml";
    ofd.Filter =
Diplom2022.Properties.Resources.XML_File_Type;
    ofd.Title =
Diplom2022.Properties.Resources.DialogTitle_SelectManifest;
    Nullable<bool> result = ofd.ShowDialog();
    if (result == true)
    {
        this.ManifestFilePath = ofd.FileName;
    }
}

private void buttonImportPrivateKey_Click(object
sender, RoutedEventArgs e)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.DefaultExt = ".xml";
            ofd.Filter =
Diplom2022.Properties.Resources.XML_File_Type;
            ofd.Title =
Diplom2022.Properties.Resources.DialogTitle_SelectKey;
            Nullable<bool> result = ofd.ShowDialog();
            if (result == true)
            {
                using (StreamReader sr =
File.OpenText(ofd.FileName))
                {
                    this.Key = sr.ReadToEnd();
                }
            }
        }

        private void buttonOk_Click(object sender,
RoutedEventArgs e)
        {
            if (textBoxManifest.Text == null)
            {
                MessageBox.Show("Добавьте файл XML
зашифрованного файла!");
            }
            else
            {
                this.DialogResult = true;
            }
        }
    }
}

```

Файл ApplicationContext.cs

```

namespace Diplom2022
{
    class ApplicationContext : DbContext
    {
        public DbSet<User> Users { get; set; }

        public ApplicationContext() :
base("DefaultConnection") { }
    }
}

```

Файл User.cs

```

namespace Diplom2022
{
    //класс модели для таблицы бд
    class User
    {
        public int id { get; set; }
        private string login, password, email;

        public string Login
        {
            get { return login; }
            set { login = value; }
        }

        public string Password
        {
            get { return password; }
            set { login = password; }
        }

        public string Email
        {
            get { return email; }
            set { email = value; }
        }

        public User(string login, string password, string
email)
        {
            this.login = login;
            this.password = password;
            this.email = email;
        }

        public MainWindow MainWindow
        {
            get => default;
            set
            {
            }
        }

        internal ApplicationContext ApplicationContext
        {
            get => default;
            set
            {
            }
        }
    }
}

```

```
}
```

Файл AuthWindow.xaml.cs

```
namespace Diplom2022
{
    /// <summary>
    /// Interaction logic for AuthWindow.xaml
    /// </summary>
    public partial class AuthWindow : Window
    {
        public AuthWindow()
        {
            InitializeComponent();
        }

        public MainWindow MainWindow
        {
            get => default;
            set
            {
            }
        }

        public UserPage UserPage
        {
            get => default;
            set
            {
            }
        }

        private void Button_Auth_Click(object sender,
            RoutedEventArgs e)
        {
            string login = textBoxLogin.Text.Trim();
            string password = textBoxPassword.Text.Trim();

            //ЛОГИН меньше 5 СИМВОЛОВ
            if (login.Length < 5)
            {
                textBoxLogin.ToolTip = "Неккоректный логин";
                textBoxLogin.Background = Brushes.DarkRed;
            }

            //пароль меньше 8 СИМВОЛОВ
            else if (password.Length < 8 ||
                password.Contains("%") || password.Contains("!") ||
```

Изм.	Лист	№ докум.	Подпись	Дата

```

password.Contains("#") || password.Contains("$") ||
password.Contains("^") || password.Contains("&"))
    {
        textBoxPassword.ToolTip = "Некорректный
пароль";
        textBoxPassword.Background =
Brushes.DarkRed;
    }
    else
    {
        textBoxLogin.ToolTip = "";
        textBoxLogin.Background =
Brushes.Transparent;

        textBoxPassword.ToolTip = "";
        textBoxPassword.Background =
Brushes.Transparent;

        MessageBox.Show("Вход выполнен успешно!");

        User authUser = null;

        using (ApplicationContext db = new
ApplicationContext())
        {
            authUser = db.Users.Where(b => b.Login
== login && b.Password == password).FirstOrDefault();
        }

        if (authUser != null)
        {
            MessageBox.Show("Good");
            UserPage userPage = new UserPage();
            userPage.Show();
            Hide();
        }

        else
            MessageBox.Show("Not Good");
    }
}

private void Button_Reg_Click(object sender,
RoutedEventArgs e)
{
    MainWindow mainWindow = new MainWindow();
    mainWindow.Show();
    Hide();
}

```

```

    }
}

```

Файл MainWindow.xaml.cs

```

namespace Diplom2022
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        ApplicationContext db;
        public MainWindow()
        {
            InitializeComponent();

            db = new ApplicationContext();
        }

        private void Button_Reg_Click(object sender,
RoutedEventArgs e)
        {
            string login = textBoxLogin.Text.Trim();
            string password = textBoxPassword.Text.Trim();
            string password_2 =
textBoxPassword_2.Text.Trim();
            string email =
textBoxEmail.Text.Trim().ToLower();

            //логин меньше 5 символов
            if (login.Length < 5)
            {
                textBoxLogin.ToolTip = "Некорректный логин";
                textBoxLogin.Background = Brushes.DarkRed;
            }
            //пароль меньше 8 символов
            else if (password.Length < 8 ||
password.Contains("%") || password.Contains("!") ||
password.Contains("#") || password.Contains("$") ||
password.Contains("^") || password.Contains("&") )
            {
                textBoxPassword.ToolTip = "Некорректный
пароль";
                textBoxPassword.Background =
Brushes.DarkRed;
            }
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.37.001 ПЗ

Лист

86

```

        //пароли не совпадают
        else if (password != password_2)
        {
            textBoxPassword_2.ToolTip = "Пароли не
совпадают";
            textBoxPassword_2.Background =
Brushes.DarkRed;
        }
        //эмейл меньше 5 символов, в эмейле нет @ и .
        else if (email.Length < 5 ||
!email.Contains("@") || !email.Contains("."))
        {
            textBoxEmail.ToolTip = "Неккоректный email";
            textBoxEmail.Background = Brushes.DarkRed;
        }
        //если всё верно
        else
        {
            textBoxLogin.ToolTip = "";
            textBoxLogin.Background =
Brushes.Transparent;

            textBoxPassword.ToolTip = "";
            textBoxPassword.Background =
Brushes.Transparent;

            textBoxPassword_2.ToolTip = "";
            textBoxPassword_2.Background =
Brushes.Transparent;

            textBoxEmail.ToolTip = "";
            textBoxEmail.Background =
Brushes.Transparent;

            MessageBox.Show("Регистрация прошла успешно
");

            //объект класса
            User user = new User(login, password,
email);

            //добавить пользователя в бд
            db.Users.Add(user);
            db.SaveChanges();

            UserPage userPage = new UserPage();
            userPage.Show();
            Hide();
        }
    }

```

```

    }

    private void Button_Vxod_Click(object sender,
RoutedEventArgs e)
    {
        AuthWindow authWindow = new AuthWindow();
        authWindow.Show();
        Hide();
    }
}
}

```

Приложение В

(Обязательное)

Программное обеспечение для шифрования данных

Презентация

					ПГУ 1.090301.37.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

Программное обеспечение для шифрования данных

Выполнила:
студент группы 188В2
Симакова В.А.
Руководитель:
к.т.н. Дорошенко И.Н.

Цель проекта

- ▶ Разработать программное обеспечение, позволяющее пользователю зашифровывать данные с помощью метода комбинированного шифрования, и расшифровывать их.

Актуальность

Актуальность разработки ПО обусловлена:

- ▶ Внедрением практик удаленной работы для сотрудников
- ▶ Потребностью в безопасном хранении данных на ПК
- ▶ Потребностью в конфиденциальности данных и защите их целостности
- ▶ Отсутствии аналогов приложений, направленных на русскоговорящих пользователей

Изм.	Лист	№ докум.	Подпись	Дата

Постановка задачи

- ▶ Возможность защищенной регистрации и дальнейшей работы с выбранным профилем пользователя
- ▶ Шифрование документов различных форматов посредством совместного применения алгоритмов *AES-128* и *RSA-2048*
- ▶ Возможность дешифрации документов различных форматов
- ▶ Генерация пары ключей для реализации *RSA* алгоритма
- ▶ Создание *XML*-файла для хранения информации
- ▶ Создание хэш-функции для проверки целостности *XML*-файла с помощью алгоритма *SHA-256*

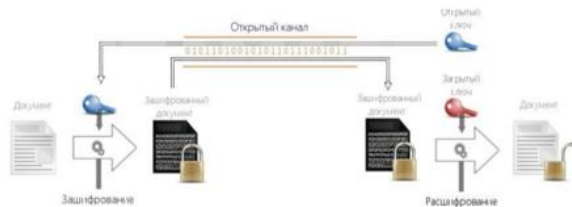
Анализ предметной области

- ▶ Симметричный алгоритм



Анализ предметной области

- ▶ Ассиметричный алгоритм



Анализ предметной области

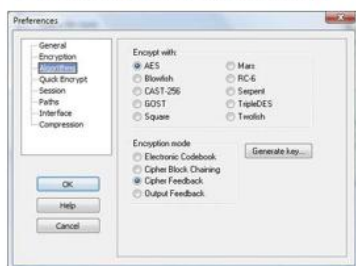
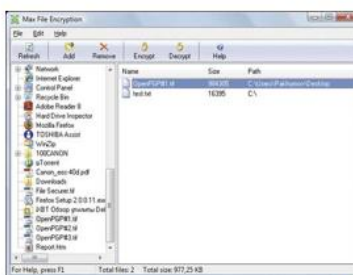
► Метод комбинированного шифрования

Комбинированная (или гибридная) криптосистема - это система шифрования, которая совмещает безопасность алгоритмов с открытым ключом с производительностью симметричных криптосистем.

В разработанном проекте метод реализуется с помощью двух криптостойких алгоритмов: AES-128 и RSA-2048.

Обзор программных средств шифрования

► Finecrypt и Max File Encryption



Используемые программные средства

- Язык разработки - C#
- Платформа - .NET Framework
- Пространство имен System.Security.Cryptography
- Технология разработки пользовательского интерфейса - Windows Presentation Form
- Языки разметки: XAML
- СУБД - SQLite

Архитектура решения

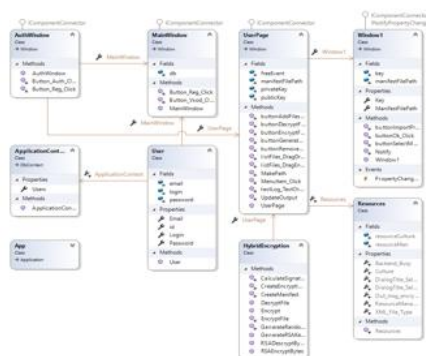
- Для разработки программного обеспечения была выбрана многоуровневая архитектура. Это широко применяемая архитектура разработки ПО, в которой приложения разделены на три физических и логических уровня: уровень представления, уровень приложения и уровень данных.



UML-диаграмма вариантов использования

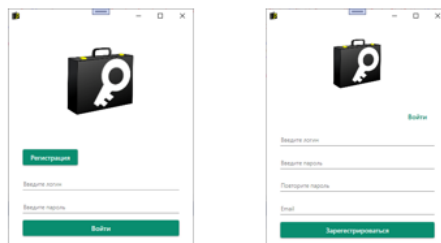


UML-диаграмма классов



Графический интерфейс приложения

- ▶ Стартовые окна для выполнения входа в программу и для регистрации нового пользователя.



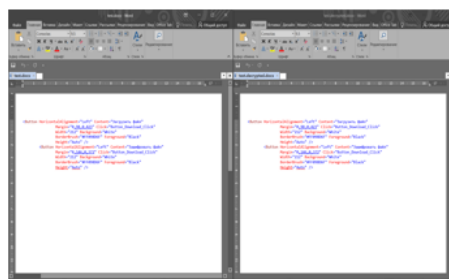
Графический интерфейс приложения

- ▶ Главное окно программы.



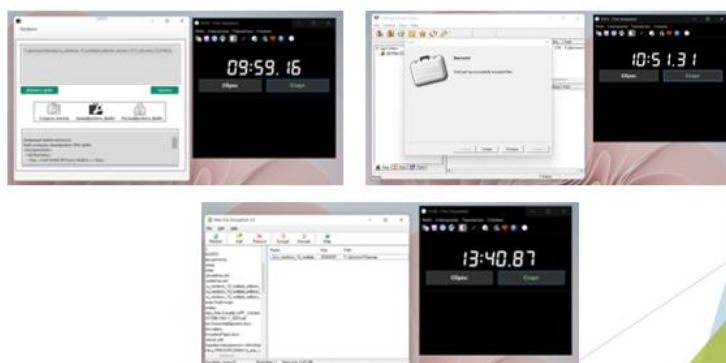
Результаты тестирования

- ▶ Тестирование целостности содержимого файла при шифровании



Результаты тестирования

► Тестирование скорости шифрования документов



Заключение

- В рамках данной выпускной квалификационной работы было разработано программное обеспечение для шифрования данных. Реализованное приложение позволяет зашифровывать и расшифровывать данные, применяя комбинированное шифрование, использующее алгоритмы AES-128 и RSA-2048.
- В дальнейшем приложение можно улучшить посредством расширения функционала, добавив шифрование не только отдельных документов, но и папок с вложенными файлами. Также можно добавить возможность создания сейфов, т.е. специальных зашифрованных контейнеров, которые после монтирования можно использовать как обычный диск. Можно реализовать функцию подписания файлов цифровой подписью и возможность проверки подписи файлов. Также можно реализовать возможность кодирования съемных носителей, таких как диски и USB-накопители.

Спасибо за внимание!

Изм.	Лист	№ докум.	Подпись	Дата