

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 ВТ и автоматизированных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРА

на тему

Клиент-серверное приложение шифрования
данных

Студент

Анастасин 03.06.20 Анастасин Вадим Дмитриевич
(подпись, дата)

Руководитель

Карамышева 03.06.20 Карамышева Н.С.
(подпись, дата)

Нормоконтролёр

Кучин Кучин А.В.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 г. № 12)

Заведующий кафедрой

Митрохин
(подпись)

М.А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от 11.06.20 № 5)

Секретарь ГЭК

Гурин
(подпись)

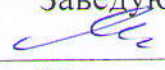
Е.И. Гурин

Пенза 2020

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Анастасин Вадим Дмитриевич гр. 16вв2 факультета вычислительной техники.
2. Тема работы «Клиент-серверное приложение шифрования данных».

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020 г.

3. Руководитель бакалаврской работы к.т.н., доцент каф ВТ ПГУ Карамышева Н.С.
4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение разработки- клиент-серверное приложение, предназначенное для шифрования файлов.
5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать языки программирования для клиентского и серверного приложения, выбрать способ взаимодействия между клиентом и сервером, разработать протокол взаимодействия.

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	30.04.2020	
2.	Выбор архитектуры приложения, языков программирования, способа взаимодействия	10	04.05.2020	
3.	Разработка протокола взаимодействия	10	10.05.2020	
4.	Разработка серверной части приложения	20	15.05.2020	
5.	Разработка клиентской части приложения	20	20.05.2020	
6.	Тестирование и отладка	10	23.05.2020	
7.	Оформления выпускной работы	20	31.05.2020	

Руководитель бакалаврской работы Карл 30.04.20 Карамышева Н.С.
(подпись, дата)

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

РЕФЕРАТ

Пояснительная записка содержит 98 листов, 33 рисунка, 5 приложений. КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ПРОТОКОЛ ВЗАИМОДЕЙСТВИЯ TCP/IP, АЛГОРИТМ ШИФРОВАНИЯ XOR, АЛГОРИТМ ШИФРОВАНИЯ RSA, АЛГОРИТМ ШИФРОВАНИЯ AES.

В рамках данной выпускной квалификационной работы было создано клиент-серверное приложение шифрования данных.

Цель работы – разработать клиент-серверное приложение для отправки файла от клиента серверу, с использованием алгоритмов шифрования различной сложности.

Объектом разработки будет являться клиент-серверное приложение, представляющие собой простой кластер. Клиент и сервер образуют единую распределенную вычислительную сеть для шифрования/расшифрования данных, взаимодействие которых осуществляется через протокол TCP/IP.

В результате проведенной работы был реализован проект, который удовлетворяет условию поставленной задачи, а именно, создано клиент-серверное приложение шифрования данных, которое представляет собой единую распределенную вычислительную сеть.

Разработка проводилась в среде программирования Visual Studio 2019. Исползуованные языки программирования: C#. Для шифрования данных при передаче файла от клиента серверу, были выбраны алгоритмы: XOR, RSA и AES.

Созданное приложение может быть внедрено на предприятие, где требуется обрабатывать файлы с помощью алгоритмов различной сложности.

Содержание

РЕФЕРАТ	4
Содержание.....	5
Введение.....	8
1 Анализ предметной области для решения поставленной задачи	10
1.1 Постановка задачи.....	10
1.2 Актуальность использования шифрования данных.....	11
1.3 Обзор алгоритмов шифрования	12
1.3.1 Симметричные алгоритмы шифрования.....	15
1.3.2 Ассиметричные алгоритмы шифрования.....	16
1.4 Определение архитектуры приложения	17
2. Выбор языка программирования и среды разработки.....	19
2.1 Язык программирования для клиентского приложения.....	19
2.2 Язык программирования для клиентского приложения.....	20
2.3 Выбор среды разработки для клиент-серверного приложения.....	21
2.4 Взаимодействие клиента и сервера	22
3 Разработка клиент-серверного приложения	24
3.1 Разработка серверной части приложения	27
3.2 Разработка клиентской части приложения	32
4 Описание работы приложения	38
4.1 Руководство пользователя.....	38
4.2 Описание работы серверного приложения.....	44
4.3 Описание работы клиентского приложения.....	45
4.4 Описание алгоритмов шифрования.....	46
4.4.1 Алгоритм шифрования XOR.....	47
4.4.2 Алгоритм шифрования RSA+AES.....	47
5 Безопасность жизнедеятельности.....	49
Заключение.....	51
Список используемых источников	53
Приложение А.....	54
Листинги программы сервера.....	54

Приложение А.1.....	54
Файл «ClientConnection.cs».....	55
Приложение А.2.....	59
Файл «Program.cs».....	59
Приложение А.3.....	59
Файл «ServerCore.cs».....	59
Приложение А.4.....	61
Файл «ServerForm.cs».....	61
Приложение А.5.....	63
Файл «ServerForm.Designer.cs».....	63
Приложение А.6.....	66
Файл «TableDataRow.cs».....	66
Приложение А.7.....	68
Файл «DataClientToServer.cs».....	68
Приложение А.8.....	68
Файл «DataServerToClient.cs».....	68
Приложение А.9.....	69
Файл «EncriptionType.cs».....	69
Приложение А.10.....	69
Файл «QueryClientType.cs».....	69
Приложение В.....	70
Листинги программы клиента.....	70
Приложение В.1.....	70
Файл «Program.cs».....	70
Приложение В.2.....	71
Файл «TcpConnection.cs».....	71
Приложение В.3.....	75
Файл «DataClientToServer.cs».....	75
Приложение В.4.....	76
Файл «DataServerToClient.cs».....	76
Приложение В.5.....	76
Файл «EncriptionType.cs».....	76

Приложение В.6.....	77
Файл «QueryClientType.cs».....	77
Приложение С	
Результат работы программы.....	78
Приложение С.1 – Вид начального экрана сервера.....	78
Приложение С.2 – Вид экрана с выбором алгоритма шифрования.....	78
Приложение С.3 – Вид экрана с подключением нового клиента.....	79
Приложение С.4 – Вид экрана с выбором файла.....	79
Приложение С.5 – Вид экрана состояния переданных файлов.....	80
Приложение С.6 – Вид экрана полученных файлов от клиентов.....	81
Приложение С.7 – Вид сообщения с ошибкой при подключении.....	81
Приложение С.8 – Вид сообщения с ошибкой при разрыве соединения.....	81
Приложение D.....	82
UML-диаграммы приложения.....	82
Приложение D.1 – UML-диаграмма классов сервера.....	82
Приложение D.2 – UML-диаграмма классов клиента.....	83
Приложение D.3 – UML-диаграмма вариантов использования сервера.....	85
Приложение D.4 – UML-диаграмма вариантов использования клиента.....	86
Приложение D.5 – UML-диаграмма деятельности клиента.....	87
Приложение D.6 – UML-диаграмма деятельности сервера.....	88
Приложение D.7 – UML-диаграмма последовательности.....	89
Приложение D.8 – UML-диаграмма развертывания.....	90
Приложение Е	91
Презентация.....	91

Введение

За века своего существования человечество придумало множество способов хранения тайны. В Древнем Риме, например, чтобы тайно передать письмо, брили голову рабу, писали на коже сообщение, а потом, дождавшись, когда волосы отрастут, отправляли его к адресату. Примерами старых алгоритмов также выступают: «шифр Цезаря», «шифр Виженера» и т.д. Но это старые и простые алгоритмы, со временем появлялись новые алгоритмы, которые стали сложнее и одновременно с этим эффективнее. Необходимость шифрования данных получила актуальность и в годы войны.

Во время Первой Мировой Войны криптография использовалась в ведении боевых действий. Лидерами в этой области были Россия и Франция, но свои наработки имели и другие страны, в том числе Англия и Германия. Возможно, самым известным случаем применения является телеграмма Циммермана. Она была перехвачена и расшифрована британским криптографическим отделом “Комната №40”. Текст этой телеграммы позволил США обосновать объявление войны Германии и вступить в боевые действия на стороне союзников. В годы Второй мировой войны, немцы использовали шифровальную машину «Энигма» для шифровки/дешифровки передаваемых сообщений между частями вермахта, что позволило обеспечить продвижение Германии на первых этапах войны, однако в противовес Энигме, вскоре появилась другая дешифровальная машина «Bombe», которая смогла расшифровать передаваемые сообщения и обеспечить тем самым победу союзников в войне с Германией.

В 21 веке развитие вычислительной техники достигло небывалых высот. Появились настольные компьютеры, планшеты, смартфоны и многие другие средства вычислительной техники, которые так плотно вошли в нашу жизнь, что стали её неотъемлемой частью. Они позволяют нам хранить огромные объемы данных и проводить невероятно большие вычисления за малый промежуток времени. Т.к. компьютеры являются средствами хранения

информации и используются повсеместно, включая военные и государственные сферы, то возникла и потребность в хранении и передаче её, используя различные средства защиты, например, шифрование.

В наше время шифрование получило значительное развитие. Появилась наука называемая «Криптографией». Криптография – это наука, изучающая способы сокрытия данных и обеспечения их конфиденциальности. Это одна из старейших наук и ее история насчитывает четыре тысячелетия.

За это время алгоритмы шифрования тоже сильно усложнились. Они разделились на две большие группы: симметричные (XOR, AES, ГОСТ, Blowfish, CAST, DES) и асимметричные (RSA, El-Gamal). Отличия между ними заключаются в том, что симметричные алгоритмы шифрования используют один ключ для зашифровывания информации и для ее дешифрования, а асимметричные алгоритмы используют два ключа – один для зашифровывания, другой для дешифрования. Все эти алгоритмы требуют затраты времени и вычислительных мощностей в зависимости от различных факторов: объема шифруемой информации и алгоритма шифрования.

В данной работе мной была поставлена задача разработать клиент-серверное приложение шифрования данных. Приложение должно представлять собой единую распределенную вычислительную систему. Пользователь должен иметь возможность выбрать файл для шифрования/расшифрования и выбрать один из предложенных алгоритмов для обработки данных.

1 Анализ предметной области для решения поставленной задачи

1.1 Постановка задачи

Требуется разработать клиент-серверное приложение, позволяющее пользователю шифровать или расшифровать выбранный файл.

К приложению предъявляются следующие требования:

1. Клиент – серверное приложение должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему.
2. Клиентское приложение должно обеспечивать подключение к серверу.
3. Клиентское приложение не должно работать без подключения к серверной части.
4. Серверное приложение должно иметь понятный графический интерфейс.
5. В окне серверного приложения должна быть возможность выбора алгоритма шифрования, по которому необходимо зашифровать выбранный файл.
6. Клиентское приложение должно быть консольным.
7. Приложение должно выполнять шифрование файла согласно заданному алгоритму.
8. В приложение должны присутствовать 2 алгоритма шифрования данных.
9. Взаимодействие между клиентом и сервером должно выполняться с использованием протокола TCP/IP.
10. Взаимодействие пользователя с клиент-серверным приложением должно осуществляться с помощью клавиатуры и мыши.
11. Операционная система – MS Windows 10 64 bit.

1.2 Актуальность использования шифрования данных

В наши дни обеспечение информационной безопасности – это первостепенный вопрос, встающий не только перед владельцами крупных и небольших компаний, но и перед обычными пользователями. Мы перестали хранить наличные в книгах, а важные данные не записываем больше ручкой в блокноты, но так ли надежны современные сервисы, облачные хранилища, и папки с файлами на персональных компьютерах?

Конечно, от злоумышленников порой тяжело спрятать конфиденциальную информацию, но обеспечить дополнительную защиту данных позволяет шифрование - искажение исходной информации посредством особых алгоритмов. Кодирование текстов начали использовать задолго до того, как был изобретен первый флеш – накопитель или было отправлено первое email-сообщение. Но что изменилось с тех пор и зачем нужно шифрование сегодня при наличии межсетевых экранов, антивирусов и других эффективных средств?

Шифрование – это ключевой способ защиты информации, от злоумышленников, позволяющий не только передавать ее по сети без рисков, но и гарантировать целостность данных. Все существующие методы кодирования основаны на криптографии – науке об обеспечении секретности сообщений. Конечно, современные способы заключаются не просто в преобразовании символов и используются не только в личной переписке. Сегодня оно требуется для работы абсолютно со всеми типами данных, которые используются в бизнес-сфере. Без шифрования информация легко может быть прочитана хакерами.

Защита информации методом шифрования обладает рядом преимуществ:

- обеспечение целостности и блокировка корпоративных данных, хранящихся в электронном виде;
- охрана баз данных, почты и других систем от несанкционированного доступа;
- защита информации от копирования и обнародования;

- повышение уровня корпоративной этики за счет обеспечения безопасности обмена личными сообщениями.

Утечка корпоративных данных может происходить при пересылке информации через интернет, и при копировании файлов сотрудниками, и при несанкционированном внедрении, и из-за неумышленных ошибок персонала. Исходя из этого можно сказать, что шифрование данных как и много сотен лет назад, остаётся актуальным средством сокрытия информации и по сей день.

1.3 Обзор алгоритмов шифрования

Для того, чтобы разобрать алгоритмы шифрования, для начала необходимо рассмотреть предметную область и понять, что необходимо для реализации поставленной задачи.

На протяжении многих веков человечество старалось сохранить личную информацию в тайне от чужих глаз и прибегало к различным способам. В следствие чего родилась целая наука, именуемая как криптография. Перед тем как перейти непосредственно к криптографии, для начала обратимся к её истокам.

Криптография – это наука об обеспечении конфиденциальности (невозможности прочтения информации посторонним). В истории криптографии условно можно выделить три периода.

Первый период – эра донаучной криптографии, когда она являлась скорее не научной дисциплиной, а искусством, доступным узкому кругу посвященных лиц, умевших так записывать тексты, чтобы они становились непонятны посторонним.

Второй период начинается с конца XIX – начала XX в., когда происходит коренное изменение способов кодирования и обработки информации, появляются телеграф, телефон, электромеханические печатные машины и иная техника. Кульминацией этого этапа явился 1949 г., когда появилась

работа известного американского математика, одного из основоположников кибернетики К. Шеннона «Теория связи в секретных системах». В ней впервые математически строго была сформулирована задача защиты информации, показаны условия невскрываемости шифров. Это событие послужило началом развития современной научной симметричной криптографии, основанной на переосмыслении и развитии принципов донаучной криптографии. В техническом плане этому содействовало развитие электронной вычислительной техники, что сделало возможным реализацию таких сложных алгоритмов, которые ранее невозможно было реализовать, используя ручные или электромеханические методы.

Третий период начинается с появлением в 1976 г. революционной работы двух других американских математиков У. Диффи и М. Хеллмана «Новые направления в криптографии», где было показано, что секретная связь возможна и без предварительной передачи секретного ключа по физически защищенному каналу связи. Эта работа послужила началом второй основной ветви современной криптографии – асимметричной криптографии. В техническом плане дальнейшему развитию криптографии содействовало бурное развитие информационно-телекоммуникационных систем, что сделало возможным реализацию сложных криптографических протоколов. Несмотря на то, что два направления различны по возрасту: симметричная криптография берет свое начало из древнего и средневекового искусства тайнописи, а асимметричной всего около 40 лет, сегодня они имеют примерно равное значение на практике. Основным фактам из истории донаучной криптографии и этапам становления современной криптографической науки и посвящена эта глава.

Следует также отметить, что криптография не занимается защитой от обмана, шантажа и подкупа, кражи ключей и других угроз для информации, которые могут возникнуть.

В данной работе основной акцент будет делаться именно на обеспечении конфиденциальности информации, т.е. на защите данных от посторонних лиц, которые не должны иметь к ним доступ. Поэтому для обеспечения конфиденциальности информации необходимо рассмотреть шифрование, как один из основных инструментов защиты информации.

Шифрование – это преобразование информации, делающее ее нечитаемой для посторонних. При этом доверенные лица могут провести дешифрование и прочесть исходную информацию. Существует множество способов шифрования/дешифрования, но секретность данных основана не на тайном алгоритме, а на том, что ключ шифрования (пароль) известен только доверенным лицам.

С помощью шифрования обеспечиваются три состояния безопасности информации:

- Конфиденциальность, т.е. сокрытие информации от неавторизованных пользователей при передаче или хранении.
- Целостность, т.е. предотвращение модификации информации при передаче или хранении.
- Идентифицируемость, т.е. аутентификация источника информации и предотвращение отказа отправителя информации от того факта, что данные были отправлены именно им.

Алгоритмы шифрования данных делятся на две группы: симметричные и асимметричные.

1.3.1 Симметричные алгоритмы шифрования

Симметричные алгоритмы шифрования используют для зашифровывания и для расшифровывания один и тот же ключ. Отсюда и происходит название «симметричные».

Ключ заранее выбирается и известен обеим сторонам. Но из-за этого возникают основной недостаток этого метода: необходимость сохранения ключа в секретности обеими сторонами, в частности уязвимым местом является первоначальная передача этого ключа другой стороне. К тому же некоторые симметричные алгоритмы (например, алфавитные алгоритмы шифрования) поддаются математическому анализу и могут быть относительно легко расшифрованы. Но степень этих недостатков варьируются в зависимости от используемого алгоритма, что в свою очередь может выступать критерием при выборе того или иного алгоритма шифрования.

Пример симметричного алгоритма шифрования DES, Triple DES и т.д.



Рисунок 1 - Схема симметричного шифрования

Схема симметричного шифрования представлена на рисунке 1. На ней изображен процесс передачи открытого ключа.

1.3.2 Ассиметричные алгоритмы шифрования

Ассиметричные системы также называют криптосистемами с открытым ключом. Это такой способ шифрования данных, при котором открытый ключ передается по открытому каналу (не скрывается) и используется для проверки электронной подписи и для шифрования данных. Для дешифровки же и создания электронной подписи используется второй ключ, секретный.

Само устройство асимметричных криптосистем использует идею односторонних функций $f(x)$, в которых несложно найти x , зная значение самой функции но почти невозможно найти саму $f(x)$, зная только значение x . Примером такой функции может служить телефонный справочник большого города, в котором легко найти номер человека, зная его фамилию и инициалы, но крайне сложно, зная номер, вычислить владельца.



Рисунок 2 - Схема асимметричного шифрования

Схема асимметричного шифрования представлена на рисунке 2. На ней изображен процесс передачи открытого ключа, а также наличие закрытого ключа.

1.4 Определение архитектуры приложения

Исходя из требования к задаче, была выбрана клиент серверная архитектура. Клиент – сервер это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически [клиент](#) и [сервер](#) – это [программное обеспечение](#). Обычно эти программы расположены на разных [вычислительных машинах](#) и взаимодействуют между собой через [вычислительную сеть](#) посредством [сетевых протоколов](#), но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, [загрузка файлов](#) посредством [HTTP](#), [FTP](#), [BitTorrent](#), [потокковое мультимедиа](#) или работа с [базами данных](#)) или в виде сервисных функций (например, работа с [электронной почтой](#), общение посредством [систем мгновенного обмена сообщениями](#) или просмотр [web-страниц](#) во [всемирной паутине](#)). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют [сервером](#), а машины, выполняющие клиентские программы, соответственно, клиентами.

К преимуществам данной архитектуры являются:

- Отсутствие дублирования кода программы-сервера программами-клиентами.
- Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются.

- Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Недостатками данной архитектуры являются:

- Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть, если он выходит из строя.
- Поддержка работы данной системы требует отдельного специалиста – [системного администратора](#).
- Высокая стоимость серверного оборудования.

2. Выбор языка программирования и среды разработки

2.1 Язык программирования для клиентского приложения

Серверная часть приложения должна обладать графическим интерфейсом и уметь реагировать на действия со стороны пользователя. Поэтому нам необходимо выбрать язык программирования, который позволяет относительно легко создавать графический интерфейс для удобства подключения клиента. Также необходимо, чтобы была возможность использовать классы при написании программы. Это значительно повысит наглядность и читаемость исходного кода, а также позволит выделить детали необходимые для грамотной реализации программы.

С учетом всего вышеперечисленного был выбран язык программирования C#.

Данный язык является объектно-ориентированным языком программирования, в котором присутствует такое понятие как классы. Класс – в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов. За счет этого можно относительно легко разбить программу на сущности, которые будут представлять собой независимые части.

Графический интерфейс серверной части приложения можно реализовать за счет такого инструмента в языке C#, как WinForms.

Windows Forms – это визуальная поверхность, на которой выводится информация для пользователя. Обычно приложение Windows Forms строится путем помещения элементов управления на форму и написания кода для реагирования на действия пользователя, такие как щелчки мыши или нажатия клавиш. Элемент управления – это отдельный элемент

пользовательского интерфейса, предназначенный для отображения или ввода данных.

2.2 Язык программирования для клиентского приложения

Клиентская часть приложения должна выполнять шифрование файла по заданному сервером алгоритму. Для реализации поставленной задачи необходимо создать дополнительные нити и обеспечить синхронизацию разделяемых данных. Вышеописанные действия позволят обнаружить и исправить возникшие ошибки для последующего их устранения. В дополнение к этому, необходима поддержка классов. Они повысят читаемость исходного кода и позволят выделить сущности необходимые для грамотной реализации программы. В свою очередь, это облегчит поддержку и развитие приложения.

С учетом выше сказанного был выбран язык программирования C#. Язык C# является объектно-ориентированным языком программирования, в котором присутствует такое понятие как классы. Класс-представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов. За счет этого можно относительно легко разбить программу на сущности, которые будут представлять собой независимые части.

Главная задача клиента будет заключаться в шифровании подзадачи, полученной от сервера, по заданному алгоритму. Это подразумевает, что необходимо обрабатывать большой объем данных по заданному алгоритму шифрования, который может представлять собой некоторую математическую задачу.

Так как клиентское приложение представляет собой консольную программу, то не требуется никаких дополнительных средств, для организации интерфейса и взаимодействия пользователя с ним.

2.3 Выбор среды разработки для клиент-серверного приложения

В качестве среды разработки клиент-серверного приложения была выбрана MS Visual Studio 2019. Microsoft Visual Studio 2019 – полный набор инструментов, с помощью которых процесс реализации инновационных замыслов разработчика легко воплощается в жизнь.

Visual Studio предоставляет минимальный функционал, при помощи которого можно создавать уникальные программные продукты.

В MS Visual Studio 2019 появились новые возможности, а именно:

- Новый интерфейс создания проектов;
- Новые возможности установки обновлений – теперь можно выбрать способ установки обновлений Visual Studio и полностью контролировать его («Сервис -> Параметры»), а также обновления Visual Studio загружаются в фоновом режиме, только когда компьютер бездействует;
- Доступна совместная работа с другими пользователями с помощью среды Visual Studio Live Share, которая устанавливается по умолчанию;
- Переработан интерфейс среды, в частности увеличено вертикальное пространство для кода;
- Добавлен вид расширения: «Бесплатное», «Платное» и «Пробная версия» – теперь гораздо легче распознать версию расширения, также добавлены соответствующие теги, для более удобного поиска;
- Система управления версиями – теперь можно временно сохранить изменения, чтобы работать над другими задачами, это возможно благодаря поддержке временного хранилища Git средствами Git в Team Explorer;
- Очистка кода одним щелчком – теперь отреагировать на предупреждения и рекомендации среды можно одним нажатием кнопки;

- Усовершенствованный поиск – теперь поиск выдает более точные результаты (а также поиск стал возможен в окнах отладки);
- Индикатор хода выполнения сборки – теперь мы можем наблюдать более подробные сведения о ходе выполнения сборки, полезно, если проект крупный и на его сборку затрачивается много времени.

Исходя из перечисленных достоинств данной версии VS была выбрана MS Visual Studio 2019.

2.4 Взаимодействие клиента и сервера

При разработке клиент-серверного приложения, передача информации между клиентом и сервером должна осуществляться на основе протокола TCP/IP.

TCP/IP- [сетевая модель](#) передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Название TCP/IP происходит из двух важнейших протоколов семейства — [Transmission Control Protocol](#) (TCP) и [Internet Protocol](#) (IP).

Протокол TCP гарантирует передачу информации от источника к приемнику, поэтому необходимо организовать взаимодействия через этот протокол, для обеспечения должной степени надежности. Для этого будет использована библиотека Winsock, которая позволяет организовать взаимодействия за счет такой абстракции как сокет. Сокет ([англ.](#) socket – разъём) – название [программного интерфейса](#) для обеспечения обмена данными между [процессами](#). Процессы при таком обмене могут выполняться как на одной [ЭВМ](#), так и на различных ЭВМ, связанных между собой [сетью](#). Сокет – [абстрактный](#) объект, представляющий конечную точку соединения.

Следует различать [клиентские](#) и [серверные](#) сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами [телефонной сети](#), а серверные – с [коммутаторами](#). Клиентское приложение (например, [браузер](#))

использует только клиентские сокеты, а серверное (например, [веб-сервер](#), которому браузер посылает запросы) – как клиентские, так и серверные сокеты.

Все рассмотренное выше, а именно: выбор архитектуры приложения, языки программирования, среда разработки приложения, протокол взаимодействия, позволит реализовать поставленную задачу в соответствии со всеми изложенными требованиями.

3 Разработка клиент-серверного приложения

Разработанное приложение шифрования данных состоит из двух частей: клиента и сервера. Они образуют единую распределенную вычислительную систему для шифрования файлов. Обмен данными между клиентом и сервером осуществляется через протокол TCP/IP. Пользователь взаимодействует только с серверной частью, поэтому для удобства работы был создан графический интерфейс. Клиент в свою очередь является консольным приложением, которое выполняет необходимые вычисления. Рассмотрим работу клиент-серверного приложения.

Для работы с клиент-серверным приложением, пользователю необходимо запустить серверную часть приложения, далее необходимо выбрать один из двух алгоритмов шифрования данных «RSA» или «XOR» и нажать на кнопку «Запустить Сервер». После ввода всех необходимых данных для работы приложения, сервер начинает прослушивание входящих соединений, после чего клиенты могут подключиться к серверу. Сервер будет находится в режиме ожидания, до тех пор, пока хотя бы один клиент не подключится. Для каждого подключенного клиента создается отдельная нить, это позволяет подключаться нескольким клиентам к серверу одновременно. После того как пользователь, выбрал необходимый для шифрования файл, клиентская часть приложения считывает этот файл в виде одного массива байт, помещает в объект и отправляет этот объект серверу. Далее сервер автоматически расшифровывает полученный файл.

Все состояния соединений выводятся в таблицу на форме. Также есть кнопка «Остановить сервер», которая прекращает работу сервера, после чего можно или закрыть приложение или выбрать еще раз алгоритм шифрования и запустить сервер заново. На форме есть кнопка «Показать папку с принятыми файлами» для удобного просмотра файлов, которые принял сервер от клиентов. Сервер автоматически расшифровывает файлы и

помещает их в эту папку. Поддержка нескольких подключений реализована с помощью потоков Thread.

UML-диаграмма последовательности для клиент-серверного приложения представлена на рисунке 3.

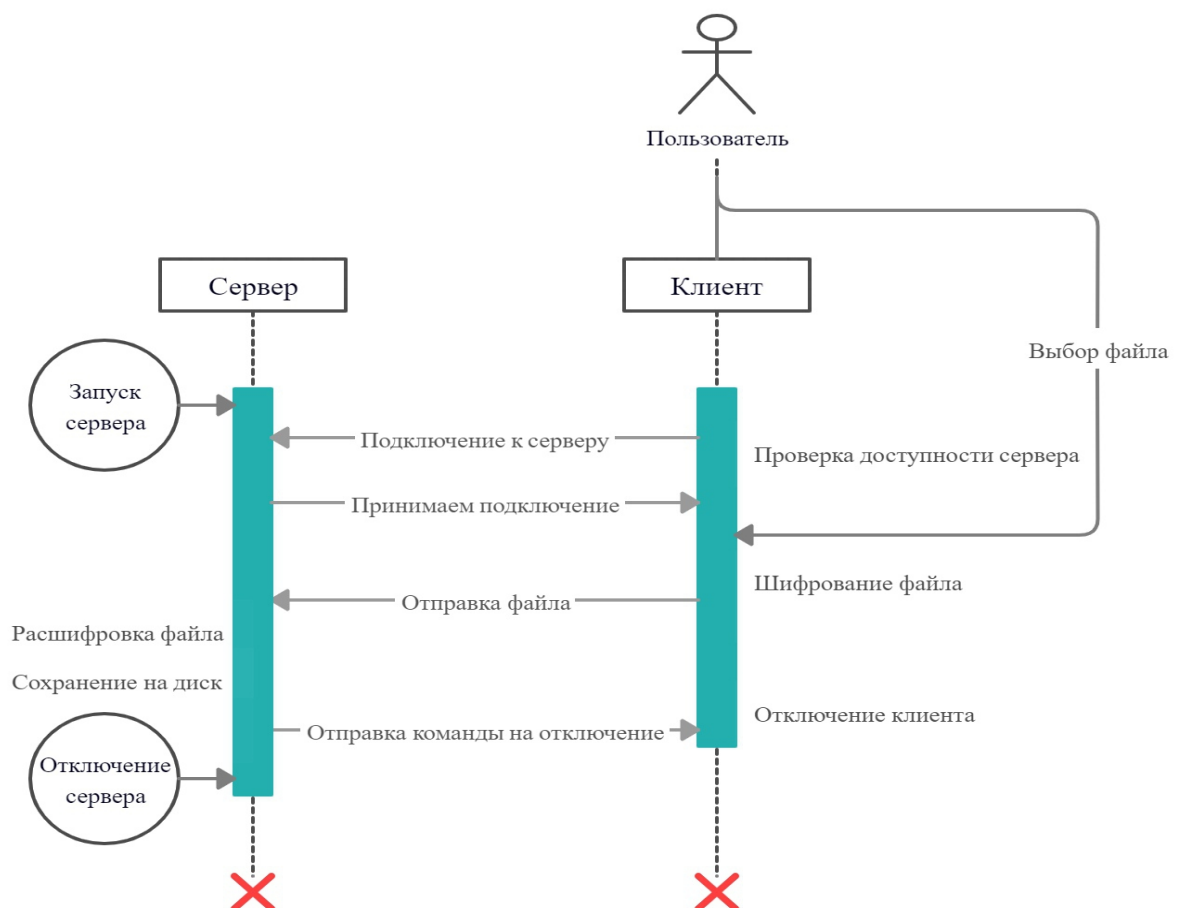


Рисунок 3 - UML-диаграмма последовательности

На основе изложенной выше информации рассмотрим клиентское и серверное приложение подробнее.

3.1 Разработка серверной части приложения

Серверное приложение является той частью, с которой непосредственно взаимодействует пользователь. Поэтому приложение имеет интуитивно понятный графический интерфейс для более комфортной работы. В нем пользователь может запустить или остановить работу сервера, выбрать один из двух алгоритмов шифрования данных, открыть папку с принятыми файлами, а также увидеть текущую статистику соединения с клиентами. Весь этот этап характеризуется тем, что пользователь устанавливает задачу. На основе этого можно составить UML-диаграмму вариантов использования, которая представлена на рисунке 4.

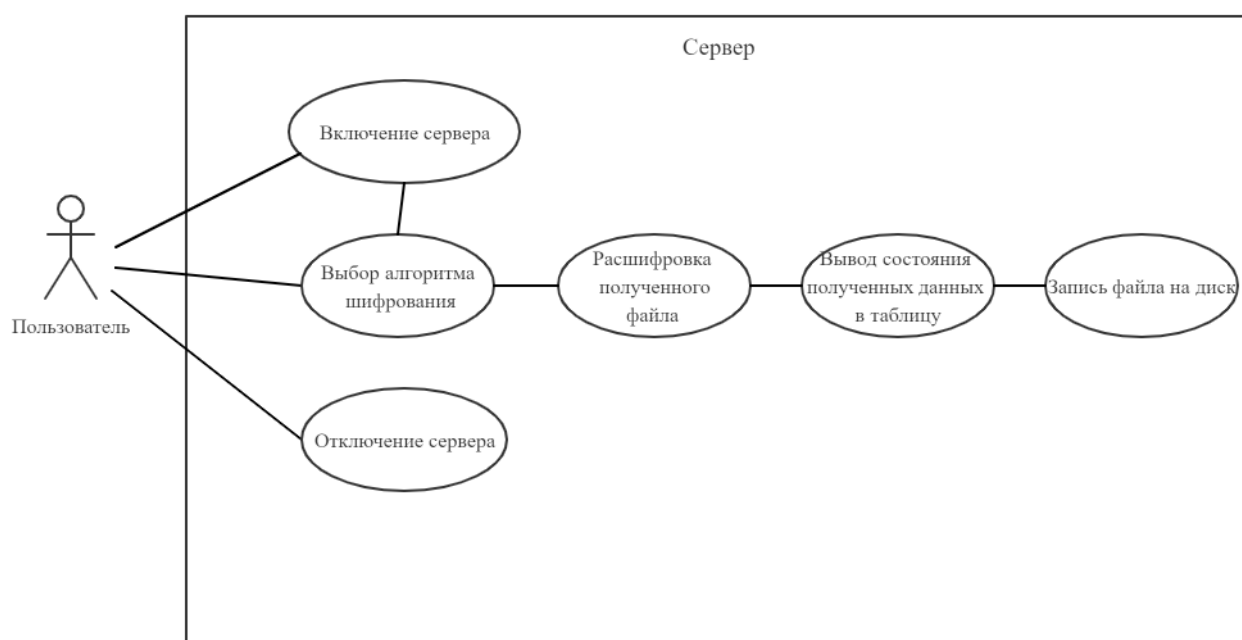


Рисунок 4 - UML-диаграмма вариантов использования сервера

После того как пользователь выбрал все необходимые параметры, сервер ожидает подключения клиентов для обработки задачи. Для каждого нового подключенного клиента создается отдельная нить, которая позволяет подключаться нескольким клиентам к серверу одновременно. После того как пользователь, выбрал необходимый для шифрования файл, клиентская часть

приложения считывает этот файл в виде одного массива байт и помещает в объект «DataClientToServer» и отправляет этот объект серверу. Далее сервер автоматически расшифровывает полученный файл. Если пользователь завершает работу, то сервер освобождает все используемые ресурсы. На основе вышеизложенного можно построить UML диаграмму деятельности сервера, представленную на рисунке 5.

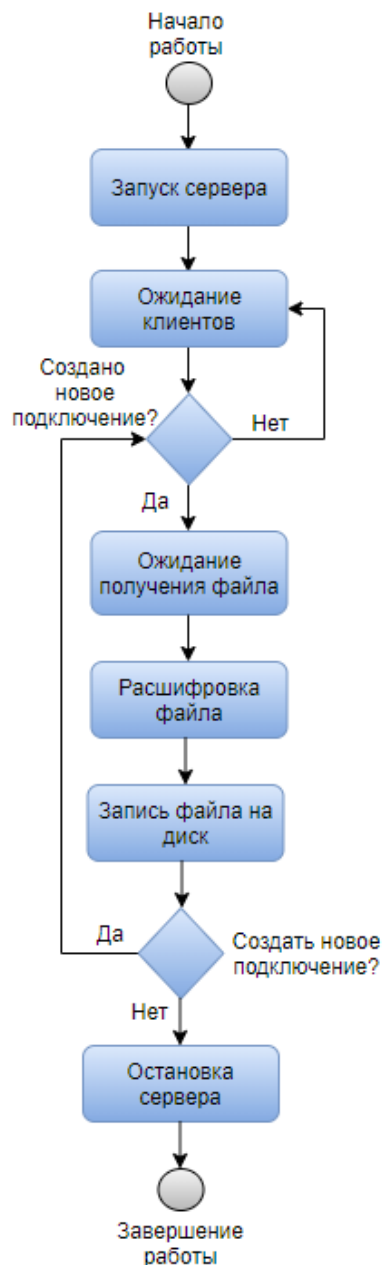


Рисунок 5 - UML-диаграмма деятельности сервера

Из диаграммы деятельности и описания работы можно выделить следующие состояния сервера:

1. Состояние ожидания входящих соединений. В этом состоянии сервер прослушивает порт TCP на входящие соединения.
2. Состояние ожидания получения файла. В этом состоянии сервер ожидает получения файла от клиента.
3. Состояние формирования результата. В этом состоянии сервер расшифровывает полученный файл.
4. Состояние, когда задача решена. В этом состоянии сервер обработал задачу и записывает полученные файлы на диск.
5. Состояние, завершения работы. В этом состоянии сервер прекращают свою работы и освобождает все занятые ресурсы.

Серверное приложение разрабатывается на языке программирования C#. C# поддерживает понятия класса, что позволяет повысить читаемость и наглядность исходного кода программы, а также выделить сущности предметной области. Каждая такая сущность является отдельным классом. Все они неким образом взаимодействуют между собой. Рассмотрим классы и методы которые понадобятся для реализации сервера.

ClientConnection – класс который управляет работой подключения между одним клиентом и сервером в отдельном потоке исполнения кода, выводит информацию о статусе подключения на основную форму.

Close – метод, который останавливает подключение между клиентом и сервером.

WorkClient - работа с клиентом, передает данные, получает файл, расшифровывает его

XorByteArrays - метод, который применяет побитовый xor для двух byte[]

GetBytesForXor - метод, который создает byte[] заданной длины

`RSAEncrypt` - метод, который дешифрует данные алгоритмом RSA по заданному `privateKey`

`Write` – метод, который отправляет объект на клиент.

`Read` – метод, который принимает объект с клиента.

`StartThread` – метод, который запускает отдельный поток исполнения кода для текущего соединения.

`Program` - класс который содержит в себе точку входа в приложение в которой создается экземпляр класса `TcpConnection`.

`Program.Main()` - Метод запускает основную и единственную форму `ServerForm`.

`ServerCore` - главный класс, который запускает сервер, слушает порт сервера на предмет новых подключений и при обнаружении подключения создает на основе его объект с типом `ClientConnection`.

`Start` - метод, который запускает прослушивание tcp сокета.

`Listen` - метод, который ищет новые соединения, строит на основе их объект с типом `ClientConnection` и добавляет его в `List` соединений.

`Close` – метод, который останавливает работу сервера.

`ServerForm` – основная форма управления сервером. Имеется две кнопки (классы `Button`), первая кнопка запускает или останавливает сервер, вторая открывает папку с принятыми файлами с помощью запуска процесса `Explorer.exe` с соответствующим параметром (метод `Process.Start()`). Также на форме есть выпадающий список (класс `ComboBox`) для выбора типа шифрования.

`StartServer_Click` - метод, который вызывается при событии клика по кнопке запуска сервера, запускает или останавливает сервер.

`AddRow` - метод, который добавляет в таблицу `DataGridViewRow`.

OpenFolder_Click - метод, который вызывается при событии клика по кнопке "показать принятые файлы", открывает папку в проводнике.

ServerForm.Designer – класс, предназначенный для графического интерфейса серверной части приложения.

TableDataRow - класс для удобной работы с DataGridViewRow.

DataGridViewRow- объект в котором отображается текущий статус подключения и остальная необходимая информация (имя отправляемого файла, тип шифрования).

DataClientToServer - класс на основе которого делается объект в котором передается информация от клиента к серверу.

DataServerToClient - класс на основе которого делается объект в котором передается информация от сервера к клиенту.

EncrptionType - перечисление, которое хранит в себе информацию о том, какое используется шифрование сервером (RSA или XOR).

QueryClientType - перечисление, которое хранит в себе информацию о том, какой сейчас статус подключения.

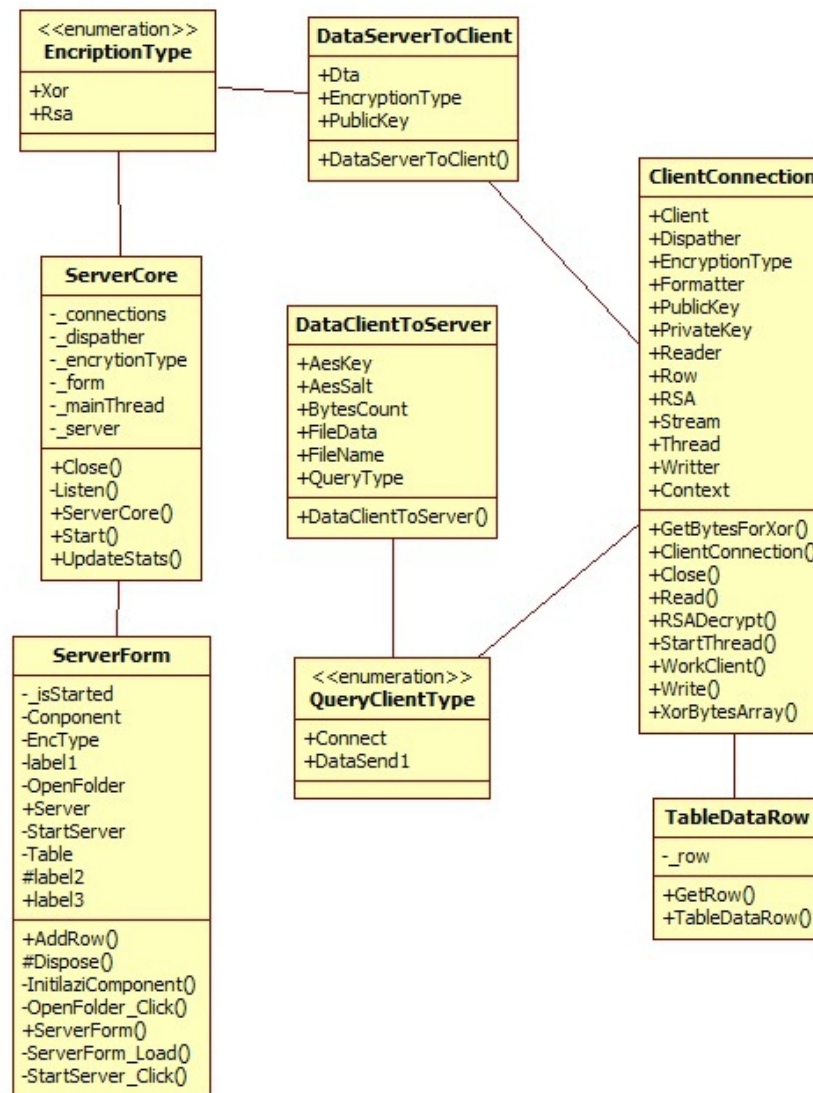


Рисунок 6 - Диаграмма классов серверного приложения

3.2 Разработка клиентской части приложения

Клиентское приложение является той частью, которая непосредственно выполняет обработку данных. Так как основная функция клиента – это выполнение обработки порции информации, то приложение является консольным. Оно подключается к серверу и ожидает задачу, которую необходимо обработать. UML-диаграмма вариантов использования для клиента представлена на рисунке 7.

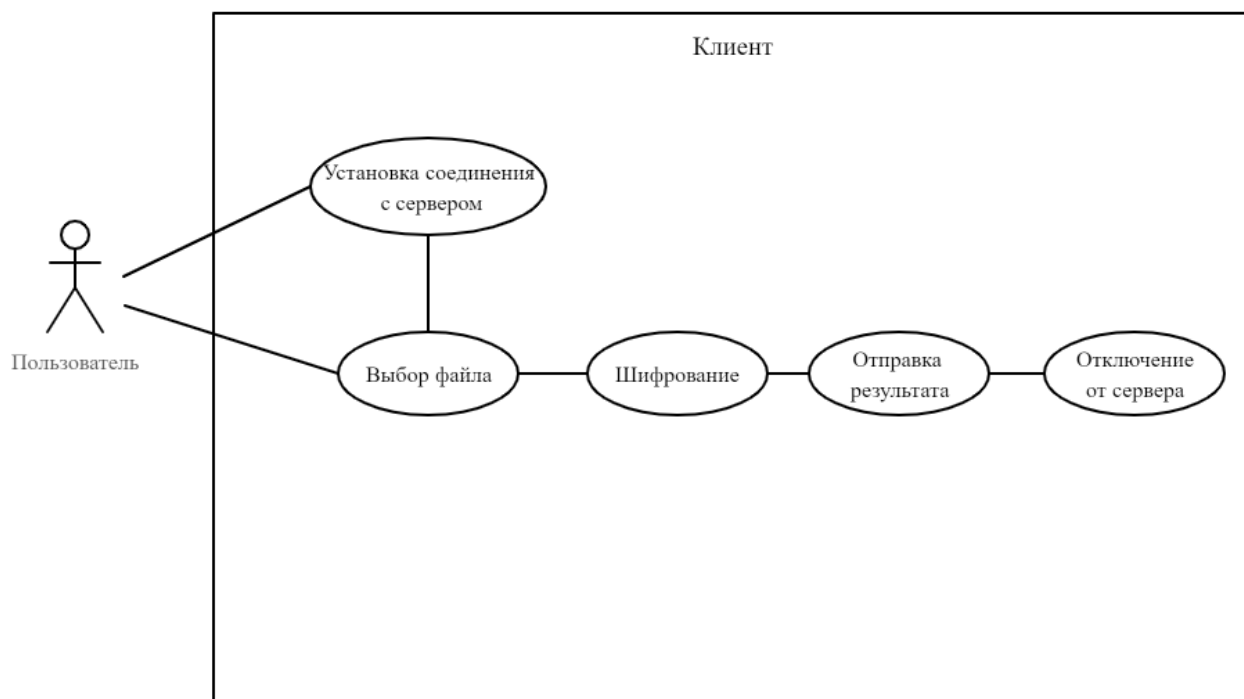


Рисунок 7 - UML-диаграмма вариантов использования клиента

При запуске приложения клиент пытается установить соединение с сервером. Если соединение установить не удалось, то выводится уведомление об ошибке. Как только задача получена, клиент, исходя из выбранного алгоритма, приступает к шифрованию файла. С целью повышения эффективности, клиент считывает файл в виде одного массива байт и помещает в объект «DataClientToServer» и отправляет этот объект серверу. Как только файл был передан серверу, в консольном окне клиента будет уведомление о успешной отправке файла. Так продолжается до тех пор, пока сервер присылает задачи. После отправки файла, клиент освобождает все ресурсы и завершает свою работу. UML-диаграмма деятельности клиента представлена на рисунке 8.

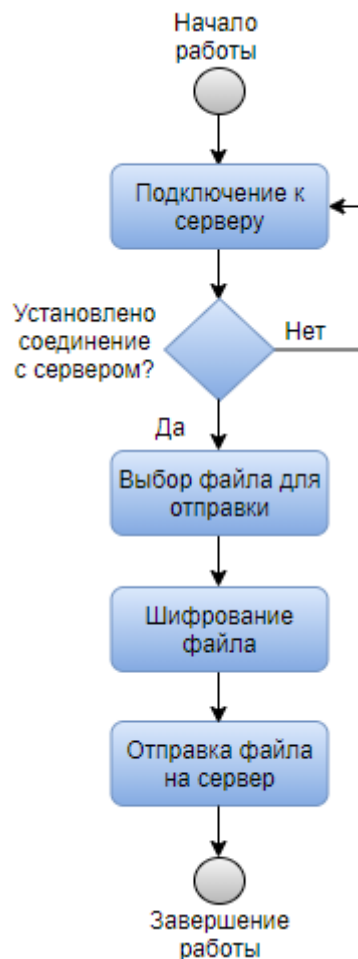


Рисунок 8 - UML-диаграмма деятельности клиента

Из диаграммы деятельности и описания работы можно выделить следующие состояния клиента:

- 1 Состояние подключение к серверу. В этом состоянии клиент пытается установить соединение с сервером.
- 2 Состояние выборки. В этом состоянии пользователь выбирает файл для шифрования и отправки на сервер.
- 3 Состояние обработки задачи. В этом состоянии клиент обрабатывает задачу.
- 4 Состояние отправки результата. В этом состоянии клиент отправляет результат обработки на сервер.
- 5 Состояние завершения работы. В этом состоянии клиент завершает работу.

Клиентская часть приложения разрабатывается на языке программирования C#. Язык C# поддерживает понятия класса, что позволяет повысить читаемость и наглядность исходного кода программы, а также выделить сущности предметной области. Каждая такая сущность является отдельным классом. Все они неким образом взаимодействуют между собой. Рассмотрим классы и методы, которые понадобятся для реализации работы клиента.

Program – класс, который содержит в себе точку входа в приложение в которой создается экземпляр класса TcpConnection.

ReadAllLines – метод, который открывает текстовый файл, считывает все строки файла, а затем закрывает его.

WriteAllLines - метод, который создает новый файл, записывает в него указанный массив строк и затем закрывает файл.

Show – метод, который отображает окно сообщения с заданным текстом, заголовком, кнопками и значком.

MessageBox – функция, которая создает, показывает на экране и использует окно сообщения. Окно сообщения содержит определяемое программой сообщение и заголовок, плюс любая комбинация предопределенных значков и командных кнопок.

MessageBoxButtons – функция, которая определяет кнопки, которые нужно отображать в окне MessageBox.

MessageBoxIcon – функция, задает константы, определяющие отображаемые сведения.

TcpConnection - основной класс для работы клиента с сервером. Он создает подключение к серверу, а также отправляет файл с нужным шифрованием.

Program.Main() - метод создает объект на основе класса TcpConnection, который и подключается к серверу для обмена с ним данными. После

создания объекта, он сразу же пытается подключиться к серверу, если у него получается это сделать он отправляет на сервер тестовый пакет и предлагает выбрать файл для отправки.

WriteLine – метод, который записывает заданное строковое значение, за которым следует текущий признак конца строки, в стандартный выходной поток.

GetStream – метод, который возвращает объект NetworkStream, используемый для отправки и получения данных.

WorkClient - метод, который соединяет клиент с сервером и обменивается с ним данными, отправляет файл.

ShowDialog – метод, который запускает общее диалоговое окно с заданным по умолчанию владельцем.

ReadAllBytes – метод, который открывает двоичный файл, считывает содержимое файла в массив байтов и затем закрывает файл.

GetFileName – метод, который возвращает имя файла и расширение указанной строки пути.

Create – метод, который создаёт криптографический объект, используемый для выполнения симметричного алгоритма.

GenerateIV – метод, который создаёт произвольный вектор инициализации для используемого алгоритма.

GenerateKey – метод, который генерирует произвольный ключ, используемый для алгоритма.

CreateEncryptor – метод, который создаёт симметричный объект-шифратор с заданным свойством и вектор инициализации.

ToArray – метод, который записывает содержимое потока в массив байтов.

XorByteArrays - метод, который применяет побитовый xor для двух byte[].

GetBytesForXor - метод, который создает byte[] заданной длины.

RSACrypt - метод, который шифрует данные алгоритмом RSA по заданному publicKey.

Write – метод, который отправляет объект на сервер.

Read – метод, который принимает объект с сервера.

Close – метод, который закрывает текущий поток и отключает все ресурсы.

Disconnect – метод, который закрывает подключение к сокету и позволяет повторно его использовать.

NextBytes – метод, который заполняет элементы указанного массива байтов случайными числами.

Serialize – метод, который сериализует объект или граф объектов с указанной вершиной в заданный поток.

Flush – метод, который очищает все буферы текущего модуля записи и вызывает немедленную запись всех буферизованных данных.

ReadInt32 – метод, который считывает целое число со знаком длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на 4 байта вперед.

Exit – метод, который завершает процесс работы.

DataClientToServer - класс на основе которого делается объект в котором передается информация от клиента к серверу.

DataServerToClient - класс на основе которого делается объект в котором передается информация от сервера к клиенту.

EncryptionType - перечисление, которое хранит в себе информацию о том, какое используется шифрование сервером (RSA или XOR).

QueryClientType - перечисление, которое хранит в себе информацию о том, какой сейчас статус подключения.

На рисунке 9 изображена UML-диаграмма классов клиента.

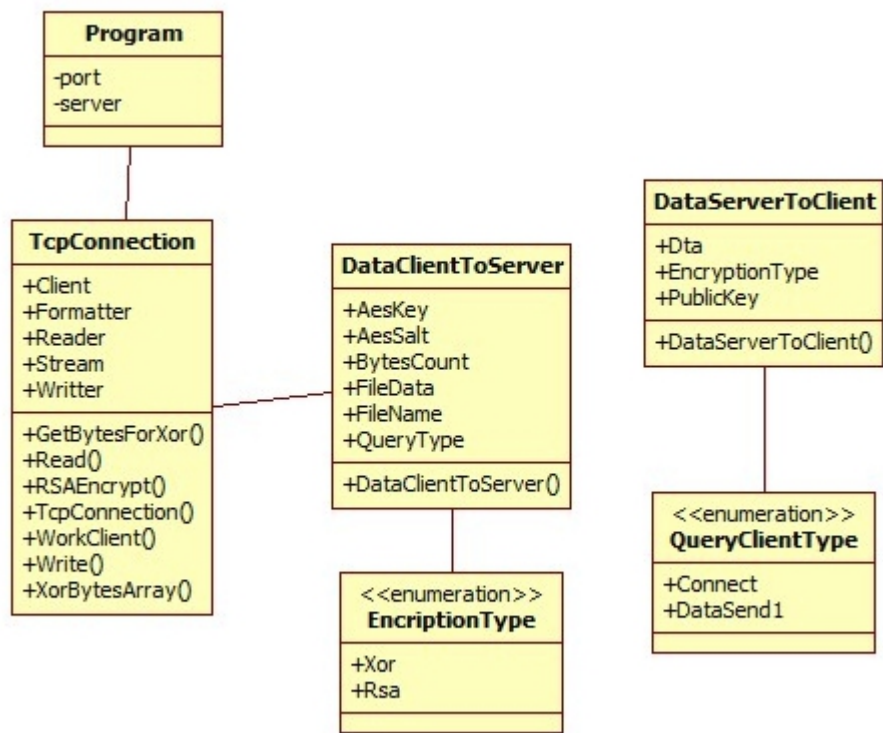


Рисунок 9 – Диаграмма классов клиентского приложения

4 Описание работы приложения

4.1 Руководство пользователя

Клиент-серверное приложение предназначено для зашифрования/расшифрования файлов. В основном, пользователь взаимодействует с серверным приложением, которое имеет интуитивно понятный графический интерфейс. Рассмотрим подробнее весь процесс взаимодействия пользователя с приложением.

Для начала работы с приложением, пользователю необходимо открыть сервер, интерфейс которого выполнен в графическом виде, представленном на рисунке 10.

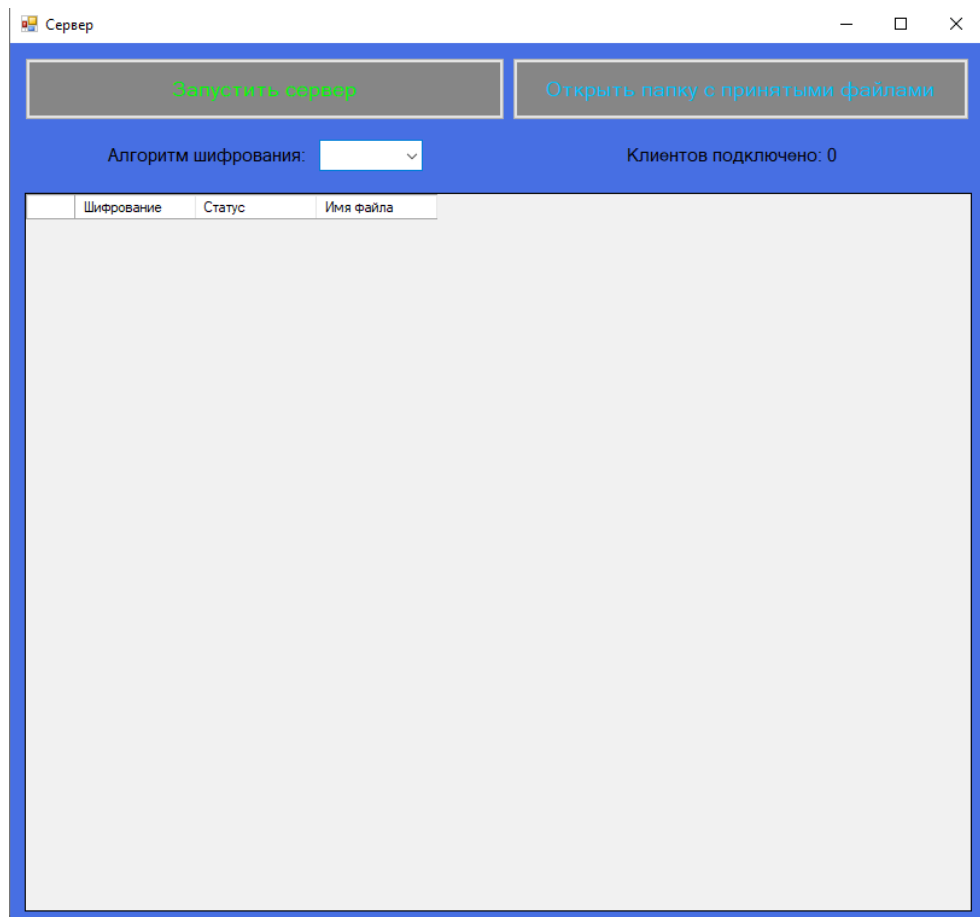


Рисунок 10 – Стартовое окно сервера

Перед тем как запустить сервер и перейти к работе с приложением, пользователь должен выбрать один из двух, предложенных алгоритмов шифрования данных, выбор которого изображен на рисунке 11.

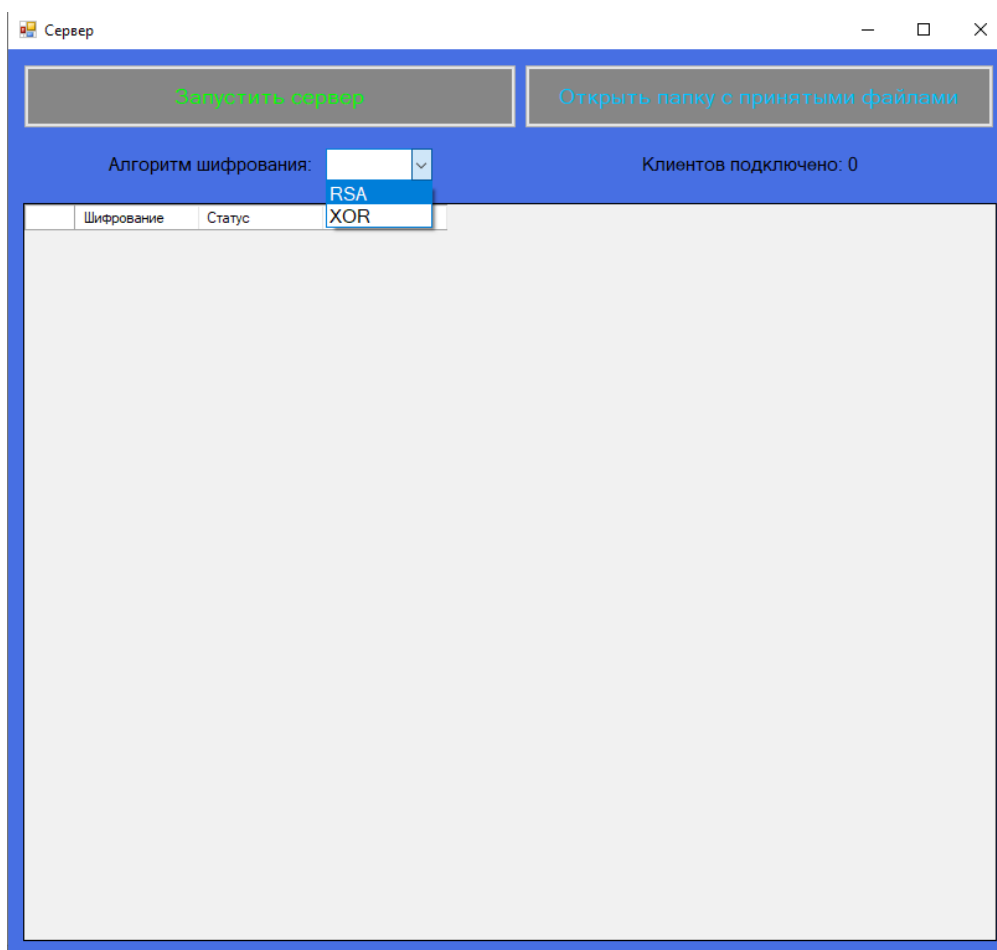


Рисунок 11 – Выбор алгоритма шифрования данных

После того как пользователь выбрал тип шифрования данных и запустил сервер, на следующем этапе работы с приложением, необходимо запустить новый клиент, информация о состоянии подключения клиента, изображена на рисунке 12.

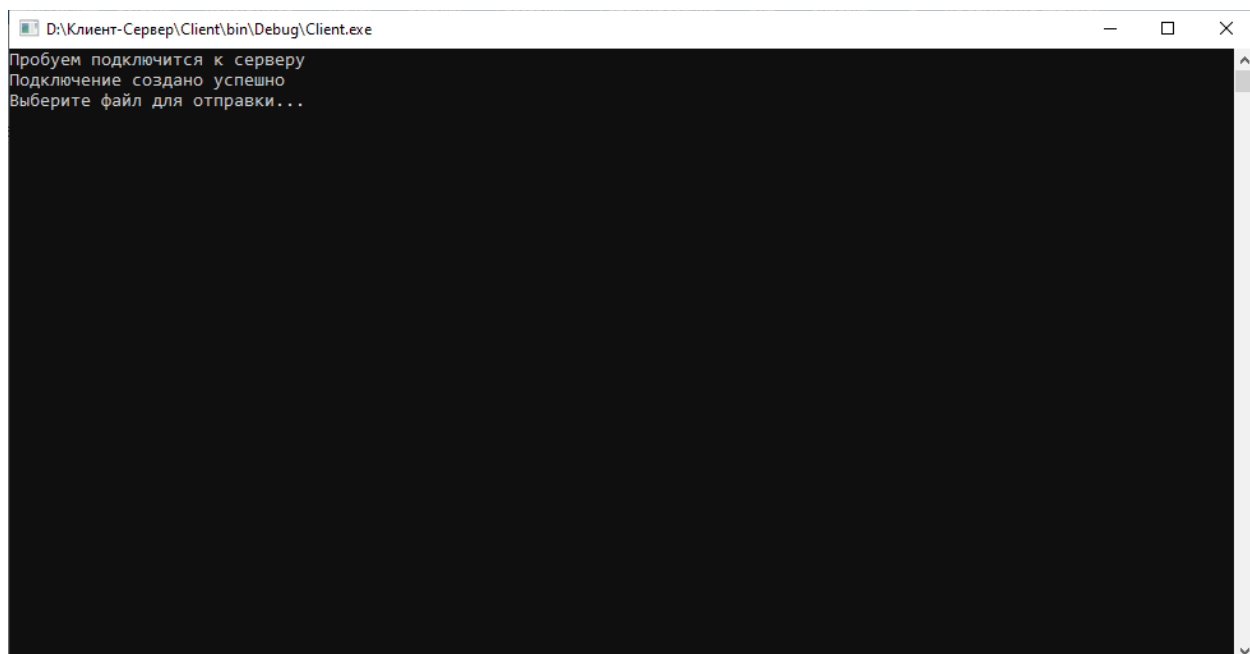


Рисунок 12 – Подключение нового клиента

Далее пользователь должен выбрать файл для шифрования данных, который необходимо передать серверу, рисунок 13.

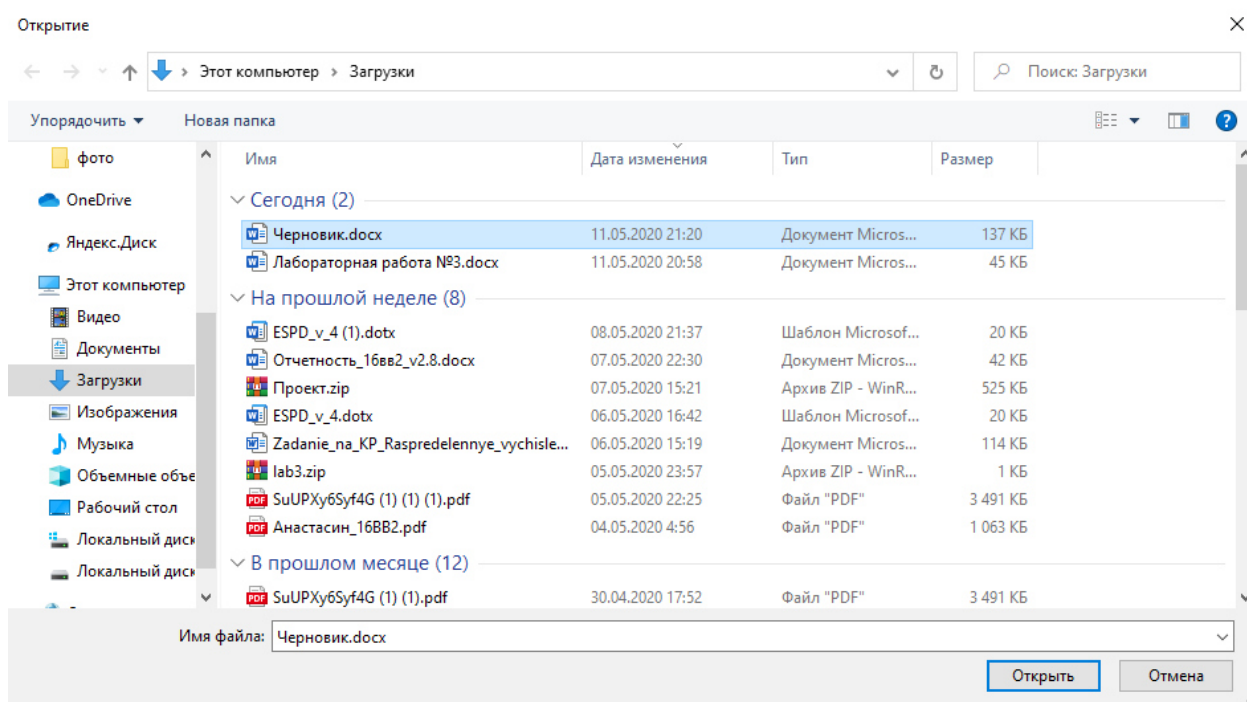


Рисунок 13 – Выбор файла для шифрования данных

В графическом окне сервера, пользователь может получить информацию о состоянии переданных файлов и способе их шифрования, а также узнать о количестве подключенных клиентов к серверу, рисунок 14.

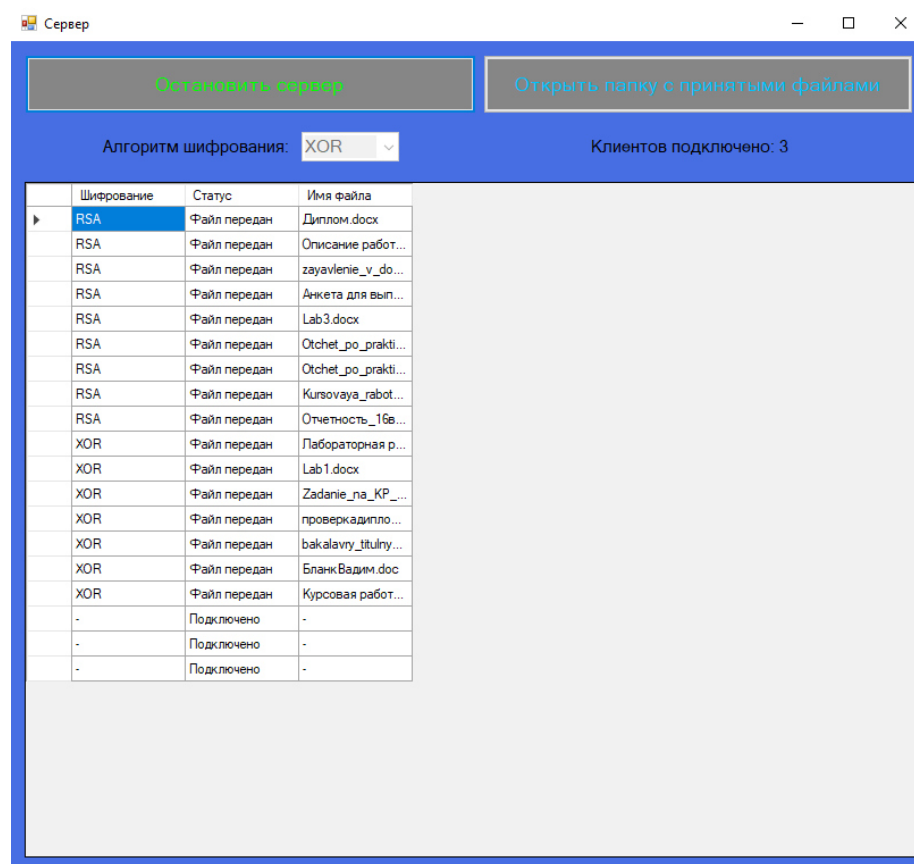


Рисунок 14 – Таблица с результатами переданных файлов

Переданные файлы от клиентов, пользователь может посмотреть, нажав на кнопку «Открыть папку с принятыми файлами», рисунок 15.

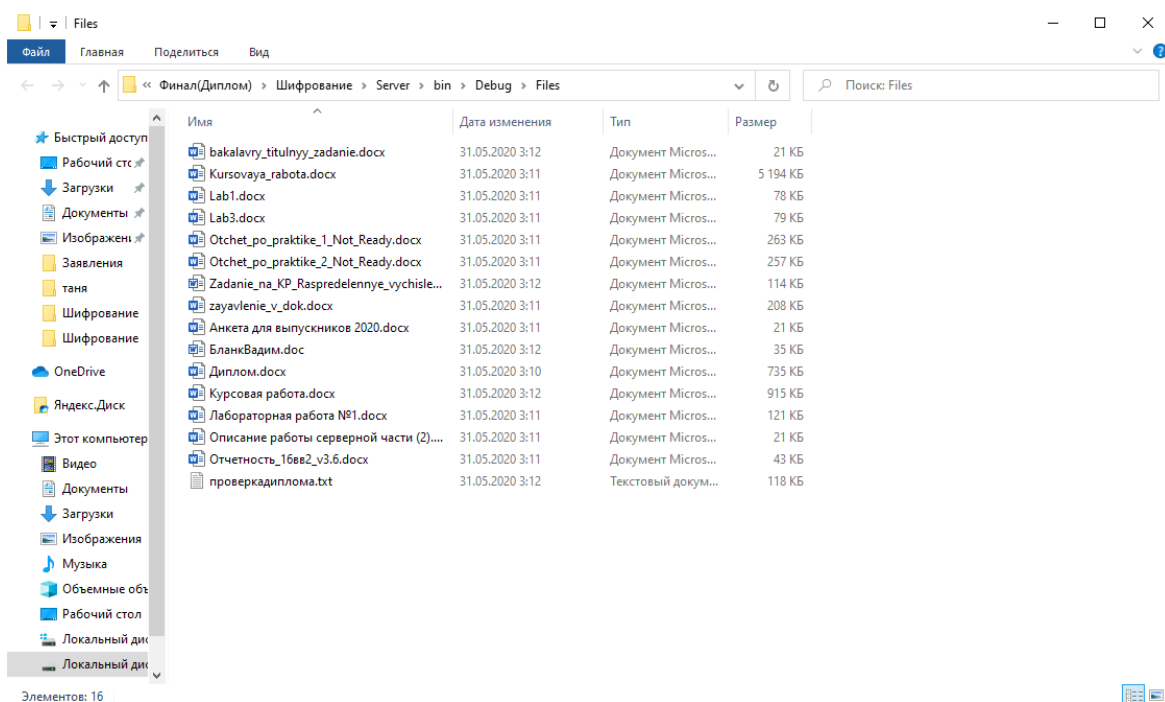


Рисунок 15 – Полученные файлы от клиентов.

Если клиент попытается подключиться без запуска сервера, то выдается сообщение об ошибке. Пример ошибки в консольном приложении клиента, показан на рисунке 16.

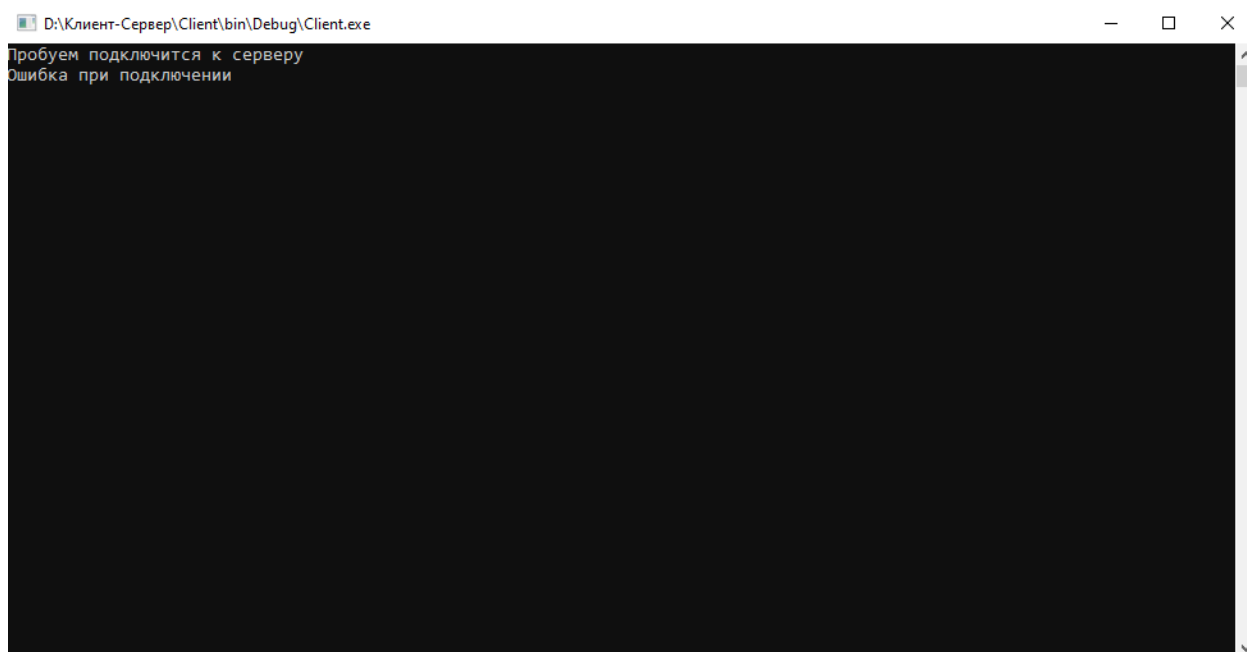


Рисунок 16 – Ошибка при подключении клиента

Если же во время подключения клиента к серверу, сервер прервет свою работу, то будет выдано следующее сообщение о ошибке подключения к

серверу. Пример ошибки в консольном приложении клиента, показан на рисунке 17.

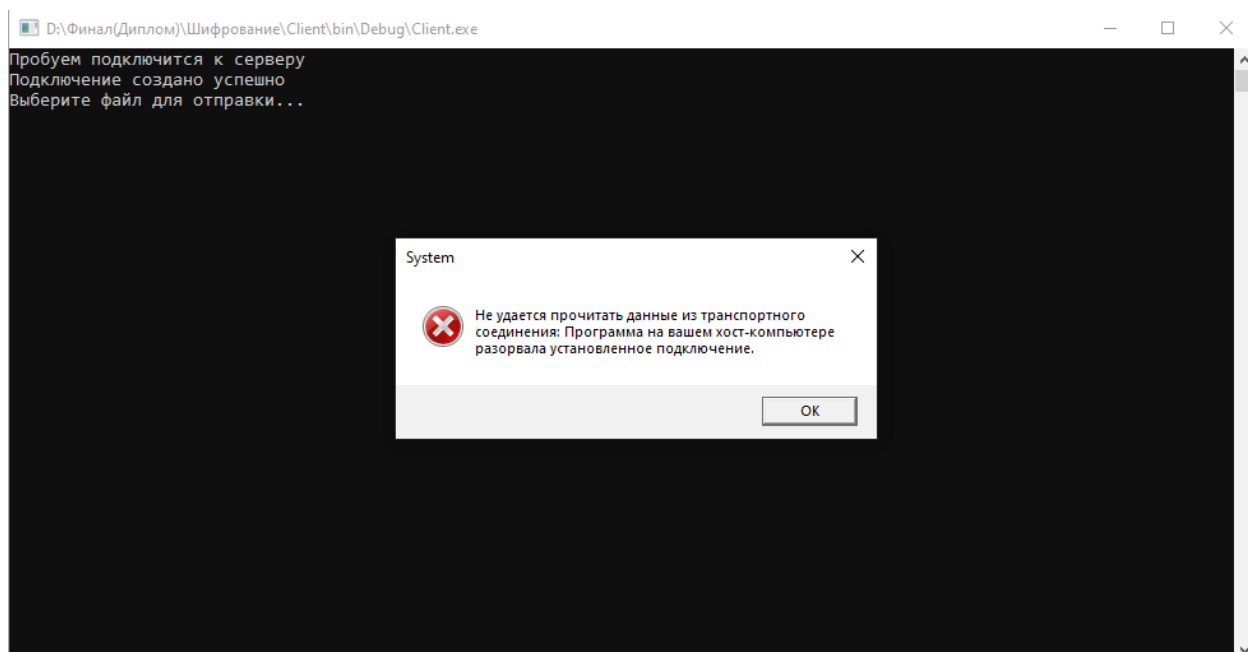


Рисунок 17 – Ошибка при разрыве соединения с сервером

4.2 Описание работы серверного приложения

При запуске сервера создается новый объект на основе класса `ServerCore`. Этот объект управляет всеми подключениями клиентов, принимает файлы, отправляет нужную информацию на сервер. Для запуска сервера, нужно создать объект с типом `ServerCore`, далее вызвать метод `Start()` этого объекта. При старте сервера создается новый поток исполнения кода (`Thread`), это сделано для того, чтобы основной UI-поток не перегружался и тем самым не вызывал ухудшение производительности прорисовки графического интерфейса. Для изменения данных графического интерфейса используется объект `Dispatcher`, который берется из основного окна и копируется во все потоки исполнения для доступа к интерфейсу из любого места. В новом потоке исполнения кода запускается бесконечный цикл `while`, который ждет подключения клиентов. Если клиент подключается, то сервер создает специально для этого пользователя отдельный поток, который уже будет обмениваться данными с клиентом. Для этого он создает объект с типом `ClientConnection`, который и работает с

клиентом. Этот объект добавляется в `List<ClientConnection>` для того, чтобы при желании пользователя остановить сервер, можно было бы закрыть все существующие соединения вызвав для каждого элемента из этого листа метод `Close()`, который в свою очередь завершает соответствующий ему поток исполнения кода, что разрывает подключение с клиентом.

`ClientConnection` – класс для работы с клиентом, выполняют всю работу с клиентом, принимает данные, расшифровывает их нужным алгоритмом и сохраняет расшифрованные данные на диск в виде файла. Сначала сервер принимает от клиента первый пакет, который не содержит никакой полезной информации, это нужно для того, чтобы проверить, работает ли подключение. В ответ на этот пакет сервер отправляет клиенту данные, которые говорят клиенту о том, какой алгоритм шифрования будет использоваться. Причем если используется алгоритм RSA, то в этом же пакете сразу отправляется дополнительно и `PublicKey` для того, чтобы клиент смог зашифровать AES ключ, которым он будет шифровать основные данные для отправки по открытому каналу.

На каждом событии отправки и принятия пакета обновляется статус подключения в основной форме. Для удобной работы с таблицей используется класс `TableDataRow`.

`TableDataRow` – класс для удобной работы с `DataGridViewRow`. Имеет три параметра – статус, имя файла, тип шифрования, которые меняются соответствующие клетки в строке. Так как код выполняется не в основном UI-поток. Для обновления `DataGridViewRow` используется `Dispatcher` основного окна.

4.3 Описание работы клиентского приложения

При запуске приложения клиент пытается установить соединение с сервером. Если соединение установить не удалось, то выводится уведомление об ошибке. Как только задача получена, клиент считывает данные из выбранного пользователем файла и принимает от сервера пакет с информацией о том, какой алгоритм шифрования нужно использовать для отправки файла по открытому каналу связи. В зависимости от алгоритма шифрования выполняются разные участки кода. Разберем оба варианта.

RSA+AES. Клиент вместе с первым пакетом, который пришел от сервера получает `PublicKey` для зашифровки `AESKEY`. Он генерирует случайный `AESKEY` и им шифрует данные из считанного файла. Далее клиент шифрует `AESKEY`. После чего отправляет имя файла, `AESKEY`, `AESSALT` (тоже генерируется случайным образом клиентом) и зашифрованные AES данные из файла серверу, который обладает всей информацией для того, чтобы расшифровать всю нужную информацию. После этого работа клиента прекращается.

Xor. Как только клиент получает информацию от сервера о том, что нужно использовать алгоритм шифрования XOR, он отправляет данные из файла, зашифрованные случайным ключом клиента. Дальше он принимает данные, которые были зашифрованы обоими ключами, расшифровывает своим и отправляет результат на сервер, который расшифровывает данные и записывает их на диск. После этого работа клиента прекращается.

4.4 Описание алгоритмов шифрования

4.4.1 Алгоритм шифрования XOR

Алгоритм основывается на том, что операция XOR коммутативна, ассоциативна и обратна сама себе. Это позволяет шифровать данные следующим алгоритмом:

Пусть данные, которые мы хотим зашифровать это A , они находятся у клиента.

Клиент придумывает случайную величину X и отправляет по каналу значение $A \text{ xor } X$. Т.е. по каналу не идет само значение A , поэтому его можно считать зашифрованным. Сервер не знает значение X , потому что оно находится у клиента, он делает следующую операцию: $A \text{ xor } X \text{ xor } Y$, Y сервер придумывает случайно. После это все обратно отправляется клиенту, клиенту приходит значение $A \text{ xor } X \text{ xor } Y$, он делает операцию: $(A \text{ xor } X \text{ xor } Y) \text{ xor } X$, X он знает потому что он его сгенерировал, у него из-за свойства операции xor получается значение $A \text{ xor } Y$, т.е. дважды xor X сам себя сократил. Клиент отправляет это значение серверу и тому для получения значения A остается только применить операцию xor Y . Теперь у сервера есть A , при этом по открытому каналу ни в один момент не ходили незашифрованные данные.

Сервер получает от клиента зашифрованное сообщение ключом клиента. Потом шифрует это сообщение ключом сервера, отправляет его на клиент, который дешифрует сообщение ключом клиента, отправляет на сервер, которому остается только дешифровать сообщение ключом сервера для получения исходного массива байт – данных файла. Дальше тем же способом, что и в RSA+AES данные сохраняются в файл

4.4.2 Алгоритм шифрования RSA+AES

Эти два алгоритма используются как правило в паре для быстрой работы и безопасности. AES – симметричный алгоритм шифрования. Т.е. для шифровки и расшифровки сообщения используется один и тот же ключ. Поэтому для зашифровки сообщения нужно сгенерировать этот ключ и

каким-то безопасным способом отправить по открытому каналу ключ для расшифровки. Для этого отлично подходит алгоритм RSA, который позволяет безопасно передать небольшие данные (ключ AES в данном случае маленький по сравнению с самим файлом). В основе алгоритма RSA лежит тот факт, что на текущем этапе развития технологий задача нахождения двух простых чисел P и Q во много раз проще, чем задача факторизации числа $N = P * Q$. Алгоритм опирается на некоторые факты из теории чисел, например малая теорема Ферма.

Сервер получает от клиента пакет с следующими данными: `FileName` – имя файла, `AESKEY` – зашифрованный ключ алгоритмом RSA и публичным ключом `PublicKey`, `AESSALT` – «соль» для алгоритма AES, `FileData` – массив зашифрованных байт алгоритмом AES. Серверу для расшифровки файла достаточно расшифровать `AESKEY` с помощью `PrivateKey`, который знает только сервер, а далее с помощью `AESKEY` и `AESSALT` расшифровать `FileData`. Потом сервер просто записывает расшифрованные байты в файл с именем `FileName`.

5 Безопасность жизнедеятельности

Безопасность жизнедеятельности это благоприятное, нормальное состояние окружающей человека среды, условий труда и учёбы, питания и отдыха, при которых снижена возможность возникновения опасных факторов, угрожающих его здоровью, жизни, имуществу, законным интересам; 2) наука о безопасном взаимодействии человека с окружающей средой; 3) учебная дисциплина в системе среднего профессионального и высшего образования, формирующая знания, умения и навыки обеспечения собственной безопасности, действий в условиях опасных, в том числе чрезвычайных ситуаций [1].

Основная цель БЖД как науки — защита человека в [техносфере](#) от негативных опасностей (воздействий) [антропогенного](#) и естественного происхождения и достижения комфортных или безопасных условий жизнедеятельности .

Воздействие [антропогенных](#) опасностей нарушает нормальную жизнедеятельность людей, вызывает аварии, приводящие к чрезвычайным ситуациям ([ЧС](#)) и катастрофам, в том числе экологическим. В настоящее время сформирована тревожная [тенденция](#) нарастания губительного воздействия опасных природных явлений и процессов. При всей специфике ситуаций в конкретных странах и регионах они обусловлены ростом народонаселения, концентрацией его и материальных богатств на сравнительно ограниченных территориях, а также изменением характера генезиса природных катастроф. Вторгаясь в природу и создавая все более мощные инженерные комплексы, человечество формирует новую, чрезвычайно сложную систему, включающую техносферу, закономерности развития которой пока неизвестны. Это приводит к увеличению неопределенности функционирования [техносферы](#), энтропийности протекающих в ней процессов, к риску возникновения технологических катастроф — крупномасштабных аварий в промышленности, энергетике, на

транспорте, загрязнению биосферы высокотоксичными и радиоактивными отходами производства, угрожающими здоровью миллионов людей.

Безопасность жизнедеятельности включает в себя систему [гражданской обороны](#).

Институт гражданской обороны начинает действовать после объявления военного положения, то есть в военное время. В России в мирное время всеми вопросами защиты граждан занимается Комиссия по ликвидации последствий чрезвычайных ситуаций и стихийных бедствий (РСЧС). Чтобы предотвратить гибель и травмирование граждан в чрезвычайных ситуациях, что является основным ущербом для государства, на территории России органами РСЧС разработаны, приняты и действуют регламентированные принципы и способы защиты населения.

Основные задачи:

- Идентификация опасностей (вид опасности, пространственные и временные координаты);
- Профилактика и защита от опасностей, исходя из концентрации и [остаточного риска](#), и ликвидация последствий воздействия опасности на человека.;
- Планирование и организация мероприятий в условиях возможных и реализованных [чрезвычайных ситуаций](#).
- Защита от опасности на основе сопоставления затрат и выгод.

[Опасность](#) — это любые явления, угрожающие жизни и здоровью человека, природной и окружающей среде; свойство живой и неживой материи, способной нанести [вред](#) человеку, привести к стойкой потере [трудоспособности](#).

Безопасность человека может быть достигнута за счет:

- снижения интенсивности потоков от источника [опасности](#);

- посредством совершенствования конструкции источников опасности;
- увеличения расстояния между источником опасности и человеком.

Уменьшения длительности их взаимодействия;

- применение различных средств защиты между человеком и источником опасности.

- Опасность — это любые явления, угрожающие жизни и здоровью человека, природной окружающей среде; свойство живой и неживой материи, способной нанести вред человеку, привести к стойкой потере трудоспособности. Опасности возникают, если потоки вещества, энергии, информации, действующие на человека или природную окружающую среду, превышают допустимые значения. Опасности действуют на человека, элементы техносферы и природную окружающую среду одновременно. Действие опасности сопровождается ущербом для человека, элементов техносферы и природной окружающей среды.

- Риск — вероятность (частота) реализации опасности.

- индивидуальный риск — характеризует опасность для отдельного человека;

- коллективный риск(социальный, групповой) — риск реализации опасности для определенного типа коллектива, определенной социальной или профессиональной группы людей.

Заключение

В данной работе было разработано клиент-серверное приложение, представляющий собой простой кластер, для шифрования файлов. Клиент и сервер образуют единую распределенную вычислительную систему. Сервер имеет интуитивно понятный графический интерфейс, в котором пользователь может выбирать алгоритм для зашифрования/расшифрования. После чего можно наблюдать за ходом выполнения работы. Клиент может подключаться к серверу, принимать от него задачу и отправлять результат обратно серверу.

Клиентская часть разрабатывалась с использованием объектно ориентированного языка программирования C#. Язык удовлетворяет требованию многомодульности. За счет наличия классов повысилась наглядность и читаемость кода. Также средства языка позволили относительно просто создавать процессы и нити.

Серверная часть разрабатывалась с использованием объектно ориентированного языка программирования C#, который также удовлетворяет требованию многомодульности. Код, написанный на C#, имеет высокую наглядность и читаемость. Язык позволил легко создать графический интерфейс и реагировать на действия пользователя с помощью WinForms.

Взаимодействие клиента и сервера реализовано по протоколу TCP/IP. Для реализации использовалась библиотека Winsock, которая позволила организовать обмен информации через сокеты.

В дальнейшем приложение можно улучшить, добавив новые алгоритмы шифрования, в частности симметричные. Можно реализовать службу на машине клиента, которая будет отслеживать запросы от сервера. Как только на сервере установили задачу, сервер будет отправлять запрос службе. Она, в свою очередь, будет запускать клиентов с заданными настройками. В качестве настроек может выступать максимальная загрузка

ЦП. Это позволит запускать больше клиентов на одной машине, которые одновременно могут работать с несколькими серверными приложениями. Можно добавить для клиентского приложения очередь, которая будет содержать задачи для обработки. Длина этой очереди будет определяться в настройках для клиента. Также можно создать специальное приложение для отдельной ЭВМ. Оно будет выступать как связующий элемент между клиентским и серверным приложением. К нему будут подключены все доступные для работы клиенты. Сервер будет взаимодействовать не с клиентами, а с этим промежуточным звеном. За счет этого с пользователя снимается необходимость контроля над подключением клиентов. Он просто задает задачу, а сервер направляет подзадачи к промежуточной ЭВМ. Она в свою очередь распределяет задачи между имеющимися клиентами. Такая архитектура получается более гибкой и позволяет обслуживать одновременно несколько серверных приложений. В этом случае удобно собирать статистику о работе клиентских машин с целью выявления потенциальных проблем и неполадок. Данная ЭВМ будет являться центральной частью, что делает её уязвимым местом. Поэтому необходимо либо организовать несколько таких узлов, работающих по определенному протоколу, либо использовать специальные средства для обеспечения надежности.

В ходе выполнения работы, были получены навыки разработки клиент серверного приложения. Улучшены навыки разработки в среде MS Visual Studio 2019. Получен дополнительный опыт и расширены знания по языку C#. Улучшены навыки работы с сокетами.

Список используемых источников

1. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен; Пер. с англ. Ю.Н. Артеменко. — М.: Вильямс, 2016. — 1312 стр.
2. Страуструп Б. «Программирование. Принципы и практика с использованием C#». 2016 г., 1328 стр.
3. Харви и Пол Дейтелы «Как программировать на C#». 2006 г., 1454 стр.
4. Хейлсберг, А. Язык программирования C#. Классика Computers Science / А. Хейлсберг, М. Торгерсен, С. Вилтамут. — СПб.: Питер, 2016. — 784 с.
5. Билл Вагнер «Наиболее эффективное программирование на C#. 50 способов улучшения кода». 2017 г., 240 стр.
6. Ишкова Э. А. Самоучитель C#. Начала программирования; Наука и техника - М., 2013. - 496 с.
7. Албахари Джозеф , Албахари Бен C# 6.0. Справочник. Полное описание языка; Вильямс - М., 2016. - 623 с.

Приложение А

Листинги программы сервера

Приложение А.1

Файл «ClientConnection.cs»

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Security.Cryptography;
using System.Threading;
using System.Windows.Forms;
using System.Windows.Threading;
using CommonClasses;

namespace Server
{
    public class ClientConnection
    {
        public Thread Thread;
        public TcpClient Client;
        public NetworkStream Stream;
        public BinaryWriter Writer;
        public BinaryReader Reader;
        public BinaryFormatter Formatter;
        public RSAParameters PrivateKey;
        public RSAParameters PublicKey;
        public RSACryptoServiceProvider RSA;
        public TableDataRow Row;
        public Dispatcher Dispatcher;
        public EncryptionType EncryptionType;
        public ServerCore Context;

        public ClientConnection(TcpClient client, Dispatcher dispatcher,
            EncryptionType encryptionType, ServerCore context)
        {
            Client = client;
            Stream = client.GetStream();
            Reader = new BinaryReader(new BufferedStream(Stream));
            Writer = new BinaryWriter(new BufferedStream(Stream));
            Formatter = new BinaryFormatter();
            RSA = new RSACryptoServiceProvider();
            Row = new TableDataRow(new DataGridViewRow());
            Dispatcher = dispatcher;
            Context = context;
            EncryptionType = encryptionType;

            PrivateKey = RSA.ExportParameters(true);
            PublicKey = RSA.ExportParameters(false);
        }
    }
}
```

```

StartThread();
}

public void Close()
{
    Client.Client.Disconnect(false);
    Stream.Close();
    Client.Close();
}

public void WorkClient()
{
    try
    {
        var connect = Read();

        if (EncryptionType == EncryptionType.Rsa)
        {
            Dispatcher.Invoke(() =>
            {
                Row.Status = QueryClientType.Connect;
                Row.EncryptionType = EncryptionType.Rsa;
            });

            Write(new DataServerToClient(PublicKey, EncryptionType.Rsa,
            null));

            var data = Read();

            Dispatcher.Invoke(() =>
            {
                Row.Status = QueryClientType.DataSend1;
                Row.EncryptionType = EncryptionType.Rsa;
            });

            var key = RSADecrypt(data.AesKey, PrivateKey, false);
            var iv = data.AesSalt;

            byte[] decryptedBytes;

            var aes = Aes.Create();
            aes.Key = key;
            aes.IV = iv;

            ICryptoTransform crypt = aes.CreateDecryptor(aes.Key, aes.IV);
            using (MemoryStream ms = new MemoryStream(data.FileData))
            {
                using (CryptoStream cs = new CryptoStream(ms, crypt,
                CryptoStreamMode.Read))
                {
                    using (var sr = new BinaryReader(cs))
                    {
                        decryptedBytes = sr.ReadBytes(data.BytesCount);
                    }
                }
            }
        }
    }
}

```

```

}
}
}

File.WriteAllBytes(Path.Combine(Directory.GetCurrentDirectory(),
"Files", data.Filename),
decryptedBytes);

Dispatcher.Invoke(() => { Row.FileName = data.Filename; });
}
else
{
Dispatcher.Invoke(() =>
{
Row.Status = QueryClientType.Connect;
Row.EncryptionType = EncryptionType.Xor;
});

Write(new DataServerToClient(PublicKey, EncryptionType.Xor,
null));

connect = Read();

var bytesCount = connect.BytesCount;

var xorBytes = GetBytesForXor(bytesCount);

var xorArray = XorByteArrays(connect.FileData, xorBytes);

Write(new DataServerToClient(PublicKey, EncryptionType.Xor,
xorArray));

var next = Read();

Dispatcher.Invoke(() =>
{
Row.Status = QueryClientType.DataSend1;
Row.EncryptionType = EncryptionType.Xor;
});

var file = XorByteArrays(next.FileData, xorBytes);
var filename = next.Filename;

var path = Path.Combine(Directory.GetCurrentDirectory(),
"Files", filename);

File.WriteAllBytes(path, file);

Dispatcher.Invoke(() => { Row.FileName = filename; });
}
}
catch
{

```

```

Row.FileName = "Ошибка";
}
Context._connections.Remove(this);
Context.UpdateStats();
}

public byte[] XorByteArrays(byte[] array, byte[] xorArray)
{
    var result = new byte[array.Length];

    for (int i = 0; i < array.Length; i++)
        result[i] = (byte)(array[i] ^ xorArray[i]);

    return result;
}

public byte[] GetBytesForXor(int bytesCount)
{
    var result = new byte[bytesCount];
    var rnd = new Random();

    rnd.NextBytes(result);

    return result;
}

public static byte[] RSADecrypt(byte[] DataToDecrypt,
    RSAParameters RSAKeyInfo, bool DoOAEPPEpadding)
{
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

    RSA.ImportParameters(RSAKeyInfo);

    return RSA.Decrypt(DataToDecrypt, DoOAEPPEpadding);
}

public void Write(DataServerToClient data)
{
    try
    {
        var memory = new MemoryStream();
        Formatter.Serialize(memory, data);

        var bytes = memory.ToArray();

        Writer.Write(bytes.Length);
        Writer.Write(bytes);
        Writer.Flush();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, e.Source, MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}

```

```

    }
}

public DataClientToServer Read()
{
    var byteCount = Reader.ReadInt32();

    var bytes = Reader.ReadBytes(byteCount);

    var obj = (DataClientToServer)Formatter.Deserialize(new
        MemoryStream(bytes));

    return obj;
}

public void StartThread()
{
    Thread = new Thread(WorkClient);
    Thread.IsBackground = true;
    Thread.Start();
}
}
}

```

Приложение A.2

Файл «Program.cs»

```

using System;
using System.Windows.Forms;

namespace Server
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new ServerForm());
        }
    }
}

```

Приложение A.3

Файл «ServerCore.cs»

```

using System.Collections.Generic;
using System.Net;

```

```

using System.Net.Sockets;
using System;
using System.Windows.Forms;
using System.IO;
using System.Threading;
using System.Windows.Threading;
using CommonClasses;

namespace Server
{
    public class ServerCore
    {
        public bool IsStarted { get; private set; }

        private TcpListener _server;
        private Thread _mainThread;
        public List<ClientConnection> _connections;
        private Dispatcher _dispatcher;
        private ServerForm _form;
        private EncryptionType _encryptionType;

        public ServerCore(Dispatcher dispatcher, ServerForm
form, EncryptionType encryptionType)
        {
            _dispatcher = dispatcher;
            _form = form;
            _encryptionType = encryptionType;
        }

        public void Start()
        {
            string[] cgf = new string[2];
            try
            {
                cgf = File.ReadAllLines("server.cfg");
            }
            catch
            {
                cgf[0] = "127.0.0.1";
                cgf[1] = "8888";
                File.WriteAllLines("server.cfg", cgf);
            }
            try
            {
                IsStarted = true;
                _server = new
TcpListener(IPAddress.Parse(cgf[0]), Convert.ToInt32(cgf[1]));
                _server.Start();
                _connections = new List<ClientConnection>();
                _mainThread = new Thread(Listen);
                _mainThread.IsBackground = true;
                _mainThread.Start();
            }
        }
    }
}

```

```

        catch (Exception e)
        {
            MessageBox.Show("Ошибка конфигурационного
файла\n" + e.Message, "Server", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }

    public void UpdateStats()
    {
        _form.label3.Text = _connections.Count.ToString();
    }

    private void Listen()
    {
        while (true)
        {
            try
            {
                var client = _server.AcceptTcpClient();
                var connection = new
ClientConnection(client, _dispatcher, _encryptionType, this);
                _connections.Add(connection);
                UpdateStats();
                _dispatcher.Invoke(() =>
{ _form.AddRow(connection.Row.GetRow()); });
            }
            catch
            {
                break;
            }
        }
    }

    public void Close()
    {
        foreach (var connection in _connections)
        {
            connection.Close();
        }
        _connections.Clear();
        IsStarted = false;
        _server.Stop();
        UpdateStats();
    }
}
}

```

Приложение A.4

Файл «ServerForm.cs»

```

using System;
using System.Diagnostics;

```

```

using System.IO;
using System.Windows.Forms;
using System.Windows.Threading;
using CommonClasses;

namespace Server
{
    public partial class ServerForm : Form
    {
        public ServerForm()
        {
            InitializeComponent();

            Table.Columns.Add("EncType", "Шифрование");
            Table.Columns.Add("Status", "Статус");
            Table.Columns.Add("File", "Имя файла");
        }

        public ServerCore Server;
        private bool _isStarted = false;

        private void StartServer_Click(object sender, EventArgs
e)
        {
            if (_isStarted)
            {
                Server.Close();
                _isStarted = false;
                EncType.Enabled = true;
                StartServer.Text = "Запустить сервер";
            }
            else
            {
                _isStarted = true;
                Server = new
ServerCore(Dispatcher.CurrentDispatcher, this,
EncType.SelectedIndex == 0 ? EncryptionType.Rsa :
EncryptionType.Xor);
                StartServer.Text = "Остановить сервер";
                EncType.Enabled = false;
                Server.Start();
            }
        }

        public void AddRow(DataGridViewRow row)
        {
            Table.Rows.Add(row);
        }

        private void OpenFolder_Click(object sender, EventArgs
e)
        {

```

```

        var path =
Path.Combine(Directory.GetCurrentDirectory(), "Files");
        Process.Start("explorer.exe", path);
    }
}
}

```

Приложение A.5

Файл «ServerForm.Designer.cs»

```

namespace Server
{
    partial class ServerForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components =
null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources
should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.Table = new
System.Windows.Forms.DataGridView();
            this.StartServer = new
System.Windows.Forms.Button();
            this.OpenFolder = new System.Windows.Forms.Button();
            this.EncType = new System.Windows.Forms.ComboBox();
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();

```

```

        ((System.ComponentModel.ISupportInitialize)(this.Table)).BeginInit();
        this.SuspendLayout();
        //
        // Table
        //
        this.Table.AllowUserToAddRows = false;
        this.Table.AllowUserToDeleteRows = false;
        this.Table.BackgroundColor =
System.Drawing.SystemColors.MenuBar;
        this.Table.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.Table.Location = new System.Drawing.Point(12,
124);
        this.Table.Name = "Table";
        this.Table.ReadOnly = true;
        this.Table.Size = new System.Drawing.Size(785, 595);
        this.Table.TabIndex = 0;
        //
        // StartServer
        //
        this.StartServer.BackColor =
System.Drawing.Color.Gray;
        this.StartServer.Font = new
System.Drawing.Font("Microsoft Sans Serif", 12.75F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
        this.StartServer.ForeColor =
System.Drawing.Color.Lime;
        this.StartServer.Location = new
System.Drawing.Point(12, 12);
        this.StartServer.Name = "StartServer";
        this.StartServer.Size = new System.Drawing.Size(398,
52);
        this.StartServer.TabIndex = 1;
        this.StartServer.Text = "Запустить сервер";
        this.StartServer.UseVisualStyleBackColor = false;
        this.StartServer.Click += new
System.EventHandler(this.StartServer_Click);
        //
        // OpenFolder
        //
        this.OpenFolder.BackColor =
System.Drawing.Color.Gray;
        this.OpenFolder.Font = new
System.Drawing.Font("Microsoft Sans Serif", 12.75F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
        this.OpenFolder.ForeColor =
System.Drawing.Color.DeepSkyBlue;
        this.OpenFolder.Location = new
System.Drawing.Point(416, 12);

```

```

        this.OpenFolder.Name = "OpenFolder";
        this.OpenFolder.Size = new System.Drawing.Size(379,
52);

        this.OpenFolder.TabIndex = 2;
        this.OpenFolder.Text = "Открыть папку с принятыми
файлами";

        this.OpenFolder.UseVisualStyleBackColor = false;
        this.OpenFolder.Click += new
System.EventHandler(this.OpenFolder_Click);
        //
        // EncType
        //
        this.EncType.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.EncType.FormattingEnabled = true;
        this.EncType.Items.AddRange(new object[] {
        "RSA",
        "XOR"});
        this.EncType.Location = new
System.Drawing.Point(256, 80);
        this.EncType.Name = "EncType";
        this.EncType.Size = new System.Drawing.Size(86, 26);
        this.EncType.TabIndex = 3;
        //
        // label1
        //
        this.label1.AutoSize = true;
        this.label1.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11F);
        this.label1.ForeColor = System.Drawing.Color.Black;
        this.label1.Location = new System.Drawing.Point(78,
83);

        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(172, 18);
        this.label1.TabIndex = 4;
        this.label1.Text = "Алгоритм шифрования:";
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.label2.Location = new System.Drawing.Point(508,
83);

        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(170, 18);
        this.label2.TabIndex = 5;
        this.label2.Text = "Клиентов подключено:";
        //

```

```

        // label3
        //
        this.label3.AutoSize = true;
        this.label3.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
        this.label3.Location = new System.Drawing.Point(674,
83);

        this.label3.Name = "label3";
        this.label3.Size = new System.Drawing.Size(16, 18);
        this.label3.TabIndex = 6;
        this.label3.Text = "0";
        //
        // ServerForm
        //
        this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
        this.BackColor = System.Drawing.Color.RoyalBlue;
        this.ClientSize = new System.Drawing.Size(807, 731);
        this.Controls.Add(this.label3);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.EncType);
        this.Controls.Add(this.OpenFolder);
        this.Controls.Add(this.StartServer);
        this.Controls.Add(this.Table);
        this.Name = "ServerForm";
        this.Text = "Сервер";
        ((System.ComponentModel.ISupportInitialize)(
(this.Table))).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.DataGridView Table;
    private System.Windows.Forms.Button StartServer;
    private System.Windows.Forms.Button OpenFolder;
    private System.Windows.Forms.ComboBox EncType;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    public System.Windows.Forms.Label label3;
}
}

```

Приложение A.6

Файл «TableDataRow.cs»

```

using System.Windows.Forms;
using CommonClasses;

namespace Server
{
    public class TableDataRow
    {
        private DataGridViewRow _row;

        public DataGridViewRow GetRow()
        {
            return _row;
        }

        public EncryptionType EncryptionType
        {
            set
            {
                var cell = _row.Cells[0];
                cell.Value = value == EncryptionType.Rsa ? "RSA"
: "XOR";
            }
        }

        public QueryClientType Status
        {
            set
            {
                var cell = _row.Cells[1];
                cell.Value = value == QueryClientType.Connect ?
"Подключено" : "Файл передан";
            }
        }

        public string FileName
        {
            set
            {
                var cell = _row.Cells[2];
                cell.Value = value;
            }
        }

        public TableDataRow(DataGridViewRow row)
        {
            _row = row;

            var type = new DataGridViewTextBoxCell();
            type.Value = "-";
            row.Cells.Add(type);
            type.ReadOnly = true;

            var status = new DataGridViewTextBoxCell();

```

```

        status.Value = "Подключено";
        row.Cells.Add(status);
        status.ReadOnly = true;

        var filename = new DataGridViewTextBoxCell();
        filename.Value = "-";
        row.Cells.Add(filename);
        filename.ReadOnly = true;
    }
}

```

Приложение A.7

Файл «DataClientToServer.cs»

```

using System;

namespace CommonClasses
{
    [Serializable]
    public class DataClientToServer
    {
        public QueryClientType QueryType;

        public byte[] AesKey;
        public byte[] AesSalt;

        public string Filename;

        public byte[] FileData;

        public int BytesCount;

        public DataClientToServer(QueryClientType queryType,
            byte[] aesKey,
            byte[] aesSalt, string filename, byte[] fileData,
            int bytesCount)
        {
            QueryType = queryType;
            AesKey = aesKey;
            AesSalt = aesSalt;
            Filename = filename;
            FileData = fileData;
            BytesCount = bytesCount;
        }
    }
}

```

Приложение A.8

Файл «DataServerToClient.cs»

```

using System;
using System.Security.Cryptography;

namespace CommonClasses
{
    [Serializable]
    public class DataServerToClient
    {
        public RSAParameters PublicKey;
        public EncryptionType EncryptionType;
        public byte[] Data;
        public DataServerToClient(RSAParameters publicKey,
EncryptionType encryptionType, byte[] data)
        {
            PublicKey = publicKey;
            EncryptionType = encryptionType;
            Data = data;
        }
    }
}

```

Приложение A.9

Файл «EncryptionType.cs»

```

using System;

namespace CommonClasses
{
    [Serializable]
    public enum EncryptionType
    {
        Xor,
        Rsa
    }
}

```

Приложение A.10

Файл «QueryClientType.cs»

```

using System;

namespace CommonClasses
{
    [Serializable]
    public enum QueryClientType
    {
        Connect,
        DataSend1
    }
}

```

Приложение В
Листинги программы клиента
Приложение В.1
Файл «Program.cs»

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using CommonClasses;

namespace Client
{
    public static class Program
    {
        [STAThread]
        public static void Main()
        {
            string[] cgf = new string[2];
            try
```

```

        {
            cgf = File.ReadAllLines("server.cfg");
        }
        catch
        {
            cgf[0] = "127.0.0.1";
            cgf[1] = "8888";
            File.WriteAllLines("server.cfg", cgf);
        }
        try
        {
            var connection = new TcpConnection(cgf[0],
Convert.ToInt32(cgf[1]));
        }
        catch (Exception e)
        {
            MessageBox.Show("Ошибка конфигурационного
файла\n" + e.Message, "Server", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
}
}

```

Приложение В.2

Файл «TcpConnection.cs»

```

using System;
using System.IO;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Security.Cryptography;
using System.Windows.Forms;
using CommonClasses;

namespace Client
{
    public class TcpConnection
    {
        public TcpClient Client;
        public BinaryWriter Writer;
        public BinaryReader Reader;
        public BinaryFormatter Formatter;
        public NetworkStream Stream;

        public TcpConnection(string server, int port)
        {
            try
            {
                Console.WriteLine("Попытка подключения к
серверу");
                Client = new TcpClient(server, port);
            }
            catch { }
        }
    }
}

```

```

        Console.WriteLine("Подключение создано
успешно");
    }
    catch
    {
        Console.WriteLine("Ошибка при подключении");
        return;
    }

    Stream = Client.GetStream();
    Writer = new BinaryWriter(new
BufferedStream(Stream));
    Reader = new BinaryReader(new
BufferedStream(Stream));
    Formatter = new BinaryFormatter();

    WorkClient();
}

public void WorkClient()
{
    var f = "";
    while (true)
    {
        Console.WriteLine("Выберите файл для
отправки...");

        var ofd = new OpenFileDialog();
        ofd.Multiselect = false;

        if (ofd.ShowDialog() == DialogResult.OK)
        {
            f = ofd.FileName;
            break;
        }
        else
            Console.WriteLine("Вы не выбрали файл");
    }

    var bytes = File.ReadAllBytes(f);
    var fileName = Path.GetFileName(f);

    Write(new
DataClientToServer(QueryClientType.Connect, null, null, null,
null, bytes.Length));

    var connect = Read();

    if (connect.EncryptionType == EncryptionType.Rsa)
    {
        var data = connect;
    }
}

```

```

        var publicKey = data.PublicKey;

        var aes = Aes.Create();
        aes.GenerateIV();
        aes.GenerateKey();
        var iv = aes.IV;
        var key = aes.Key;

        ICryptoTransform crypt =
aes.CreateEncryptor(aes.Key, aes.IV);
        byte[] encBytes;

        using (MemoryStream ms = new MemoryStream())
        {
            using (var cs = new CryptoStream(ms, crypt,
CryptoStreamMode.Write))
            {
                using (var sw = new BinaryWriter(cs))
                {
                    sw.Write(bytes, 0, bytes.Length);
                }
            }

            encBytes = ms.ToArray();
        }

        Write(new
DataClientToServer(QueryClientType.DataSend1, RSAEncrypt(key,
publicKey, false), iv, fileName, encBytes, encBytes.Length));

        //var encBytes = RSAEncrypt(bytes, publicKey,
false);

        //Write(new
DataClientToServer(QueryClientType.DataSend1,
EncryptionType.Rsa, fileName, encBytes, encBytes.Length));

        Console.WriteLine("Файл успешно отправлен");

        Stream.Close();
        Client.Close();
    }
    else
    {
        var xorArray = GetBytesForXor(bytes.Length);

        var xorBytes = XorByteArrays(bytes, xorArray);

        Write(new
DataClientToServer(QueryClientType.Connect, null, null, null,
xorBytes,
xorBytes.Length));
    }
}

```

```

        var next = Read();

        Write(new
DataClientToServer(QueryClientType.DataSend1, null, null,
fileName,
                    XorByteArrays(next.Data, xorArray),
next.Data.Length));

        Console.WriteLine("Файл успешно отправлен");

        Client.Client.Disconnect(false);
        Stream.Close();
        Client.Close();

    }
}

public byte[] XorByteArrays(byte[] array, byte[]
xorArray)
{
    var result = new byte[array.Length];

    for (int i = 0; i < array.Length; i++)
        result[i] = (byte)(array[i] ^ xorArray[i]);

    return result;
}

public byte[] GetBytesForXor(int bytesCount)
{
    var result = new byte[bytesCount];
    var rnd = new Random();

    rnd.NextBytes(result);

    return result;
}

public static byte[] RSAEncrypt(byte[] DataToEncrypt,
RSAParameters RSAKeyInfo, bool DoOAEPPadding)
{
    RSACryptoServiceProvider RSA = new
RSACryptoServiceProvider();

    RSA.ImportParameters(RSAKeyInfo);

    return RSA.Encrypt(DataToEncrypt, DoOAEPPadding);
}

public void Write(DataClientToServer data)
{
    try
    {

```

```

        var memory = new MemoryStream();
        Formatter.Serialize(memory, data);
        var bytes = memory.ToArray();
        Writer.Write(bytes.Length);
        Writer.Write(bytes);
        Writer.Flush();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, e.Source,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

public DataServerToClient Read()
{
    try
    {
        var bytesCount = Reader.ReadInt32();
        var bytes = Reader.ReadBytes(bytesCount);
        var obj =
        (DataServerToClient)Formatter.Deserialize(new
        MemoryStream(bytes));
        return obj;
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, e.Source,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        Environment.Exit(0xFFFF);
        return null;
    }
}
}
}
}

```

Приложение В.3

Файл «DataClientToServer.cs»

```

using System;

namespace CommonClasses
{
    [Serializable]
    public class DataClientToServer
    {
        public QueryClientType QueryType;

        public byte[] AesKey;
        public byte[] AesSalt;

        public string Filename;
    }
}

```

```

        public byte[] FileData;

        public int BytesCount;

        public DataClientToServer(QueryClientType queryType,
byte[] aesKey,
        byte[] aesSalt, string filename, byte[] fileData,
int bytesCount)
        {
            QueryType = queryType;
            AesKey = aesKey;
            AesSalt = aesSalt;
            Filename = filename;
            FileData = fileData;
            BytesCount = bytesCount;
        }
    }
}

```

Приложение B.4

Файл «DataServerToClient.cs»

```

using System;
using System.Security.Cryptography;

namespace CommonClasses
{
    [Serializable]
    public class DataServerToClient
    {
        public RSAParameters PublicKey;
        public EncryptionType EncryptionType;
        public byte[] Data;
        public DataServerToClient(RSAParameters publicKey,
EncryptionType encryptionType, byte[] data)
        {
            PublicKey = publicKey;
            EncryptionType = encryptionType;
            Data = data;
        }
    }
}

```

Приложение B.5

Файл «EncryptionType.cs»

```

using System;

namespace CommonClasses
{
    [Serializable]

```

```
public enum EncryptionType
{
    Xor,
    Rsa
}
}
```

Приложение В.6

Файл «QueryClientType.cs»

```
using System;

namespace CommonClasses
{
    [Serializable]
    public enum QueryClientType
    {
        Connect,
        DataSend1  }
}
```

Приложение С

Результат работы программы

Приложение С.1 – Вид начального экрана сервера

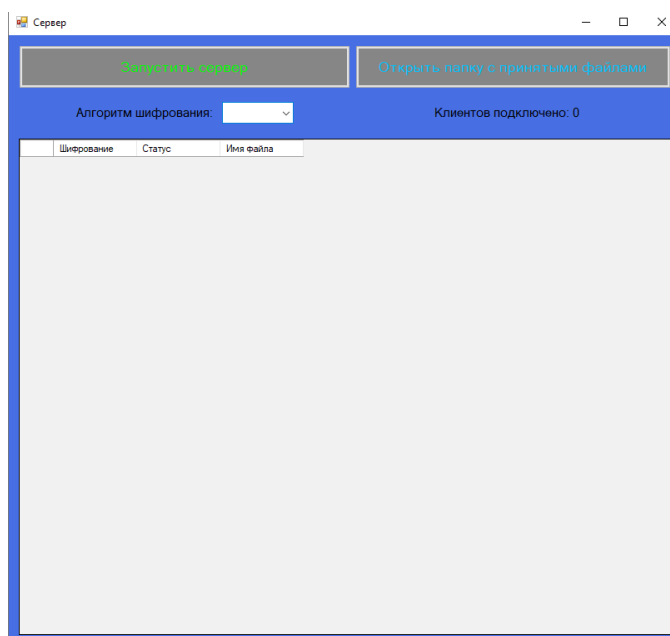


Рисунок 18 - Вид начального экрана сервера

Приложение С.2 – Вид экрана с выбором алгоритма шифрования

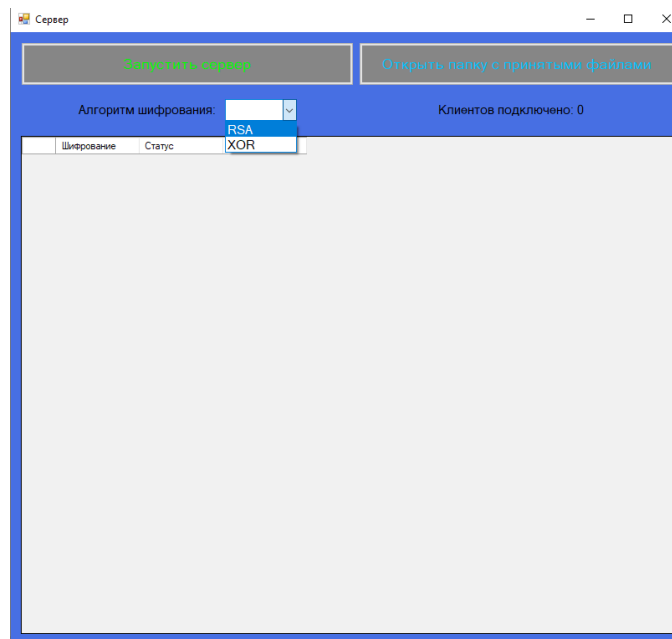


Рисунок 19 - Вид экрана с выбором алгоритма шифрования

Приложение С.3 – Вид экрана с подключением нового клиента

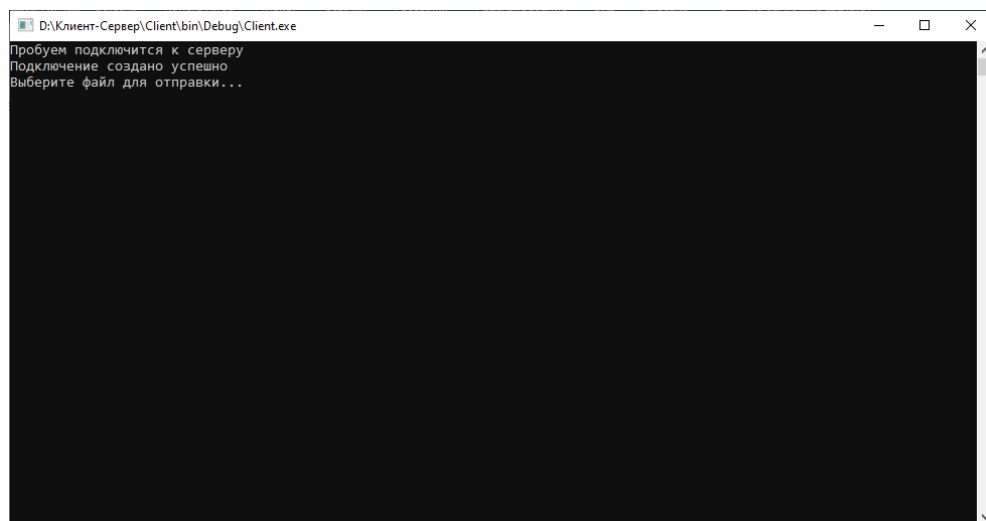


Рисунок 20 - Вид экрана с подключением нового клиента

Приложение С.4 – Вид экрана с выбором файла

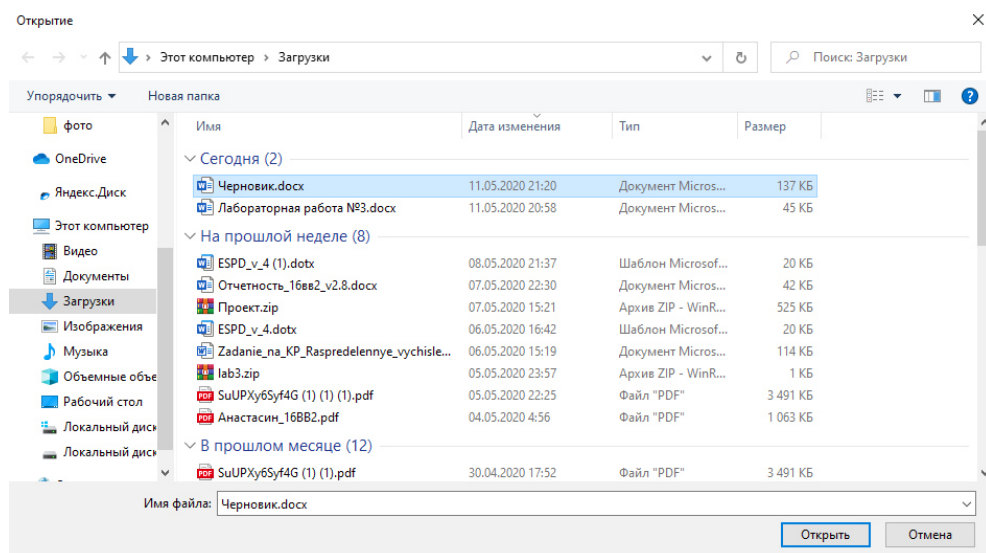


Рисунок 21 - Вид экрана с выбором файла

Приложение С.5 – Вид экрана состояния переданных файлов

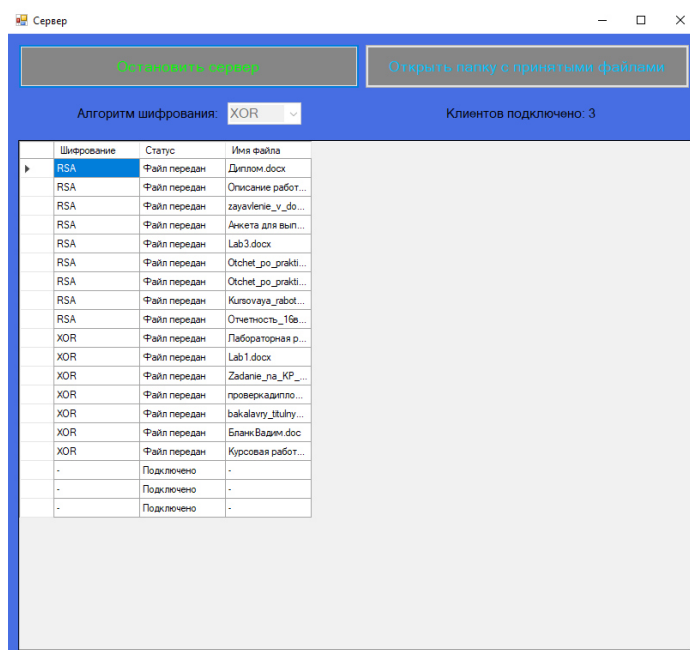


Рисунок 22 - Вид экрана состояние переданных файлов

Приложение С.6 – Вид экрана полученных файлов от клиентов

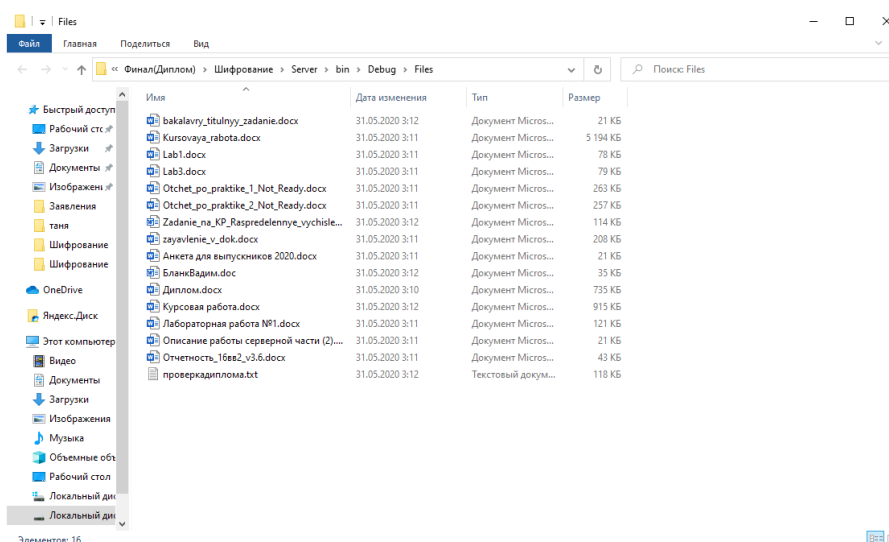


Рисунок 23 - Вид экрана полученных файлов от клиентов

Приложение С.7 – Вид сообщения с ошибкой при подключении

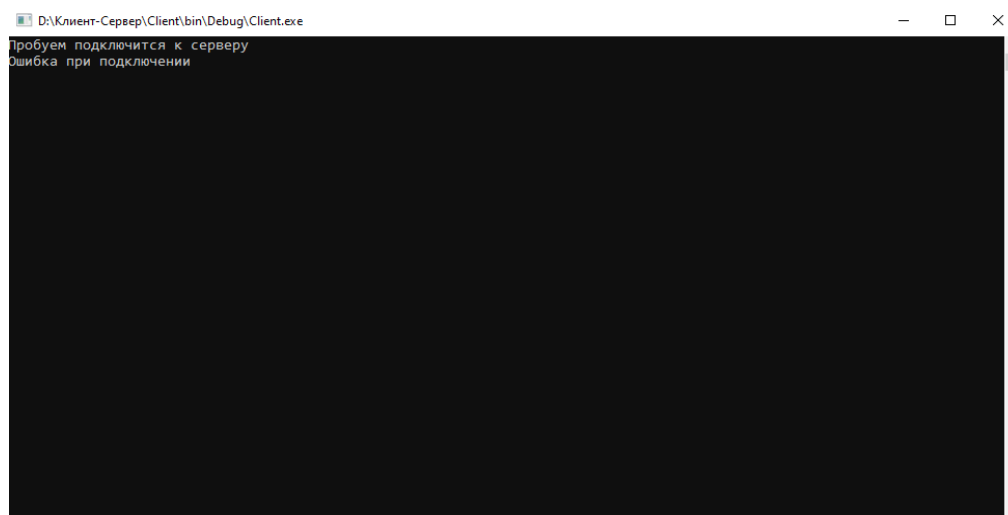


Рисунок 24 - Вид сообщения с ошибкой

Приложение С.8 – Вид сообщения с ошибкой при разрыве соединения

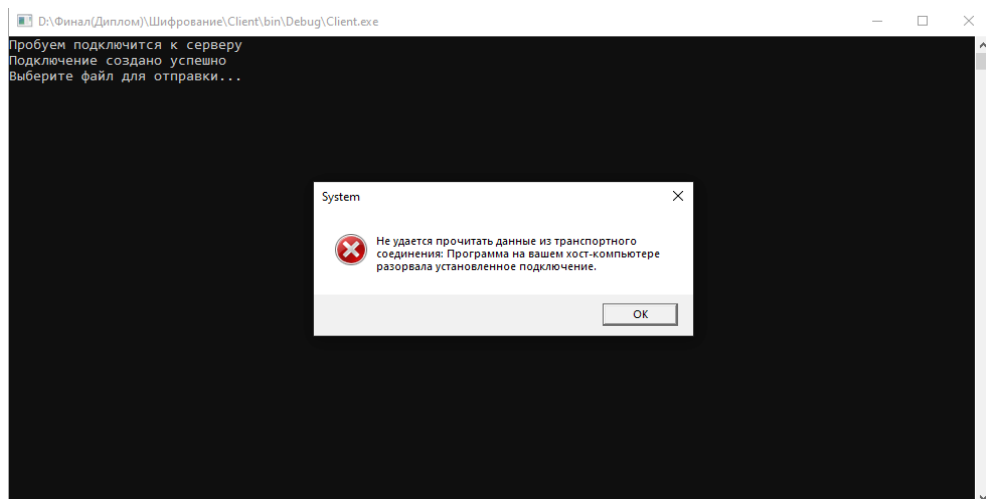


Рисунок 25 - Вид сообщения с ошибкой

Приложение D

UML-диаграммы приложения

Приложение D.1 – UML-диаграмма классов сервера

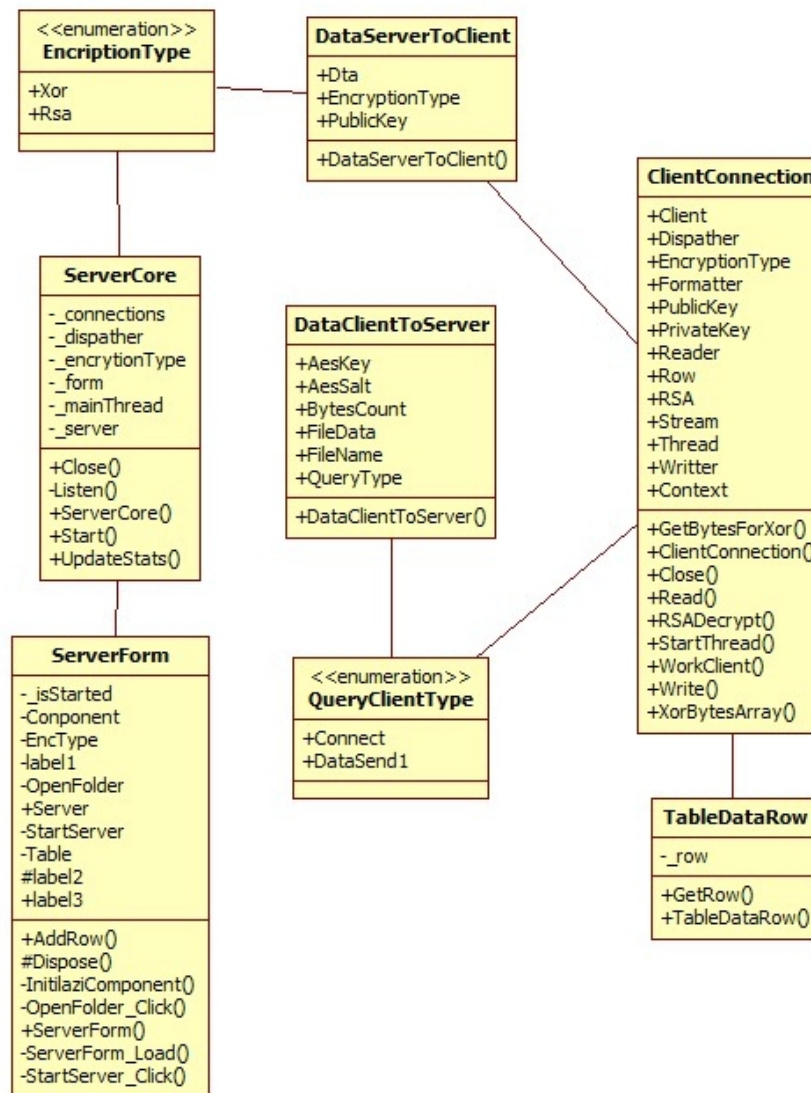


Рисунок 26 - UML-диаграмма классов сервера

Приложение D.2 – UML-диаграмма классов клиента

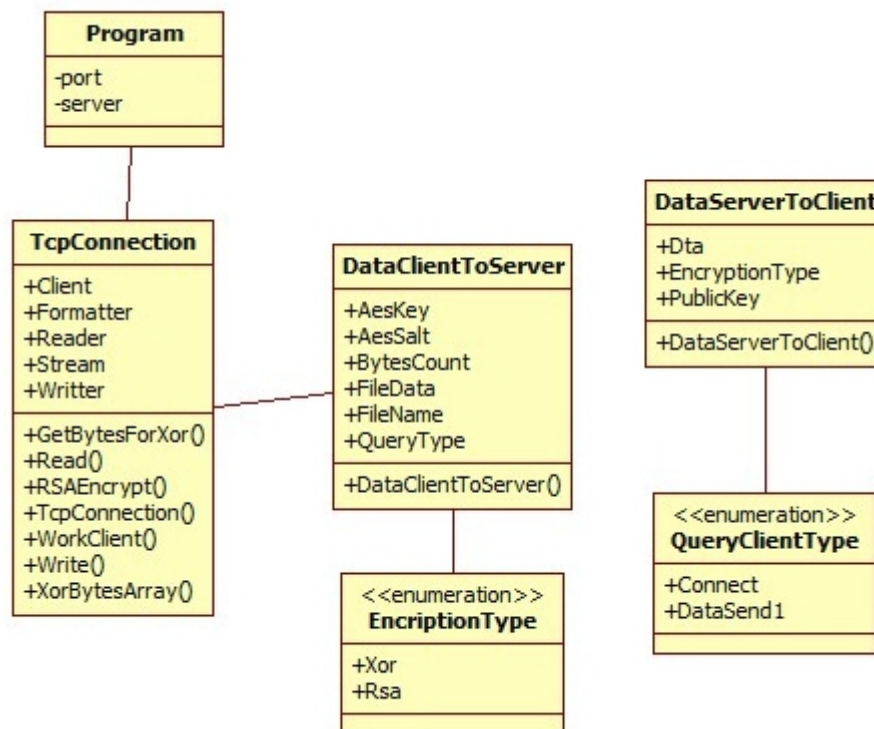


Рисунок 27 - UML-диаграмма классов клиента

Приложение D.3 – UML-диаграмма вариантов использования сервера

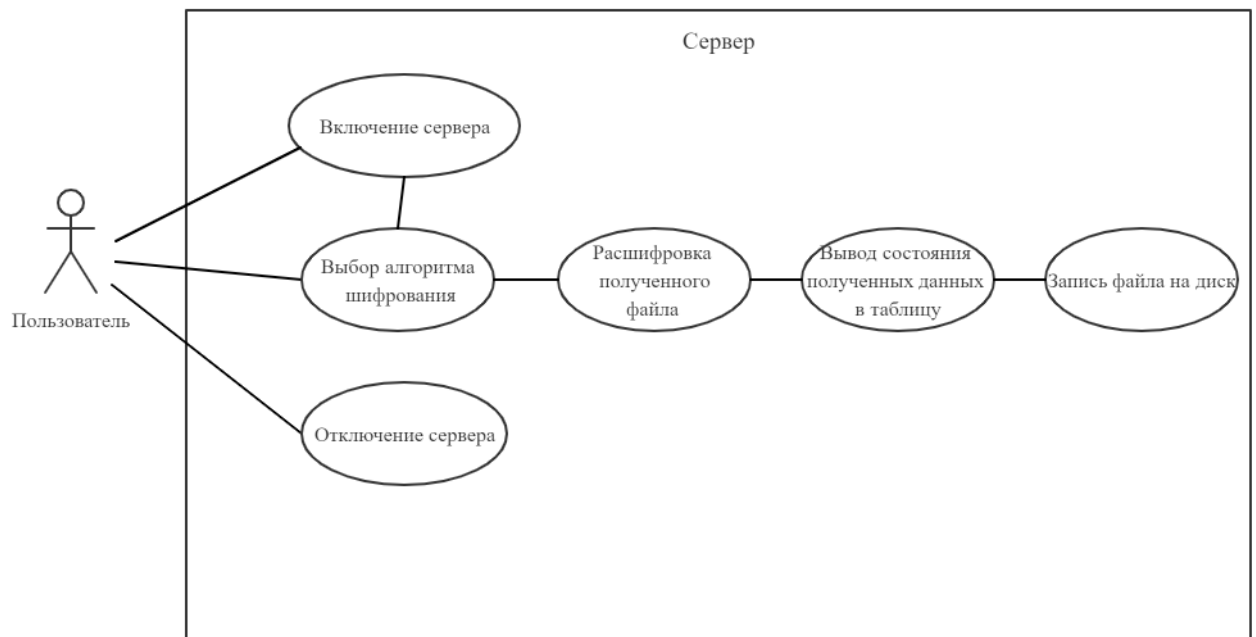


Рисунок 28 - UML-диаграмма вариантов использования сервера

Приложение D.4 – UML-диаграмма вариантов использования клиента

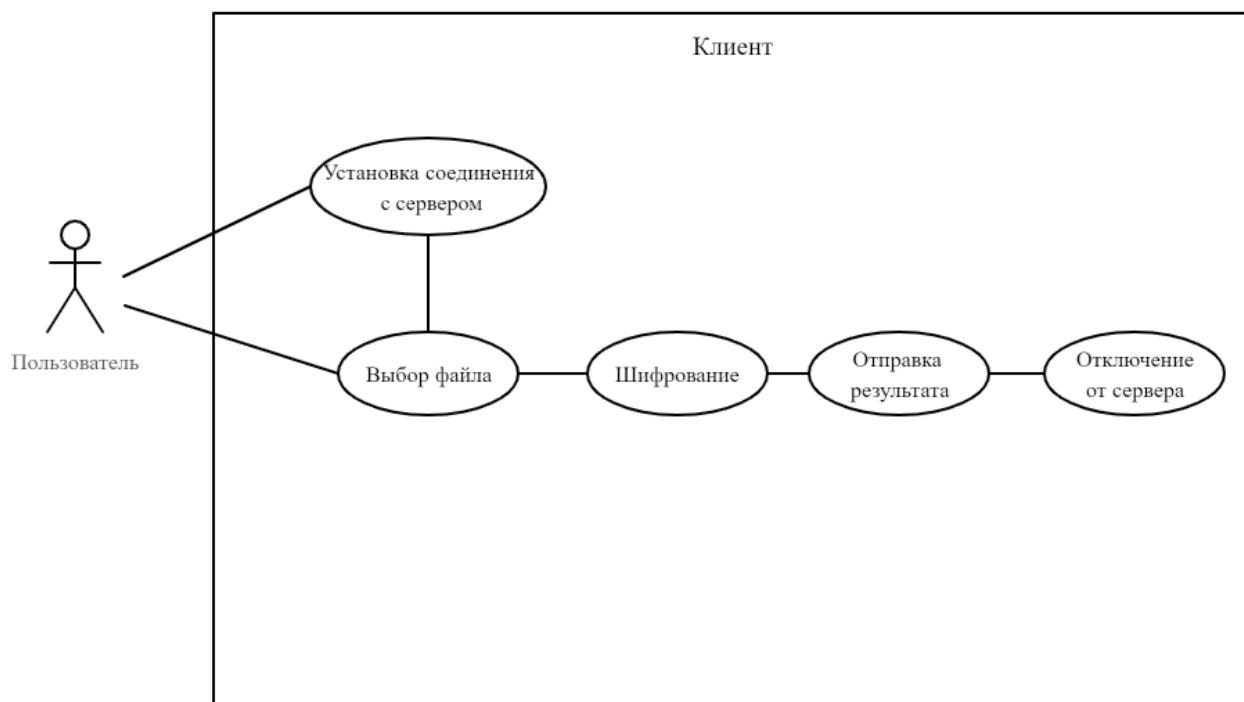


Рисунок 29 - UML-диаграмма вариантов использования клиента

Приложение D.5 – UML-диаграмма деятельности клиента

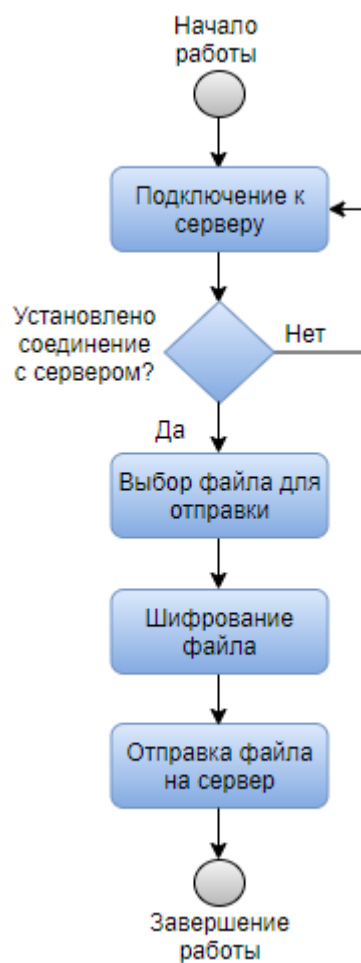


Рисунок 30 - UML-диаграмма деятельности клиента

Приложение D.6 – UML-диаграмма деятельности сервера

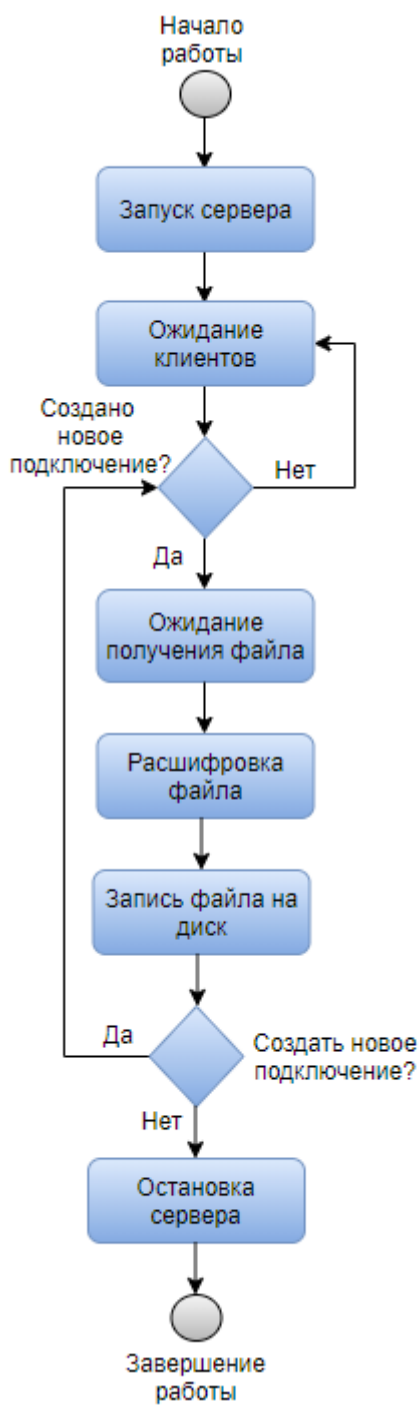


Рисунок 31 - UML-диаграмма деятельности сервера

Приложение D.7 – UML-диаграмма последовательности

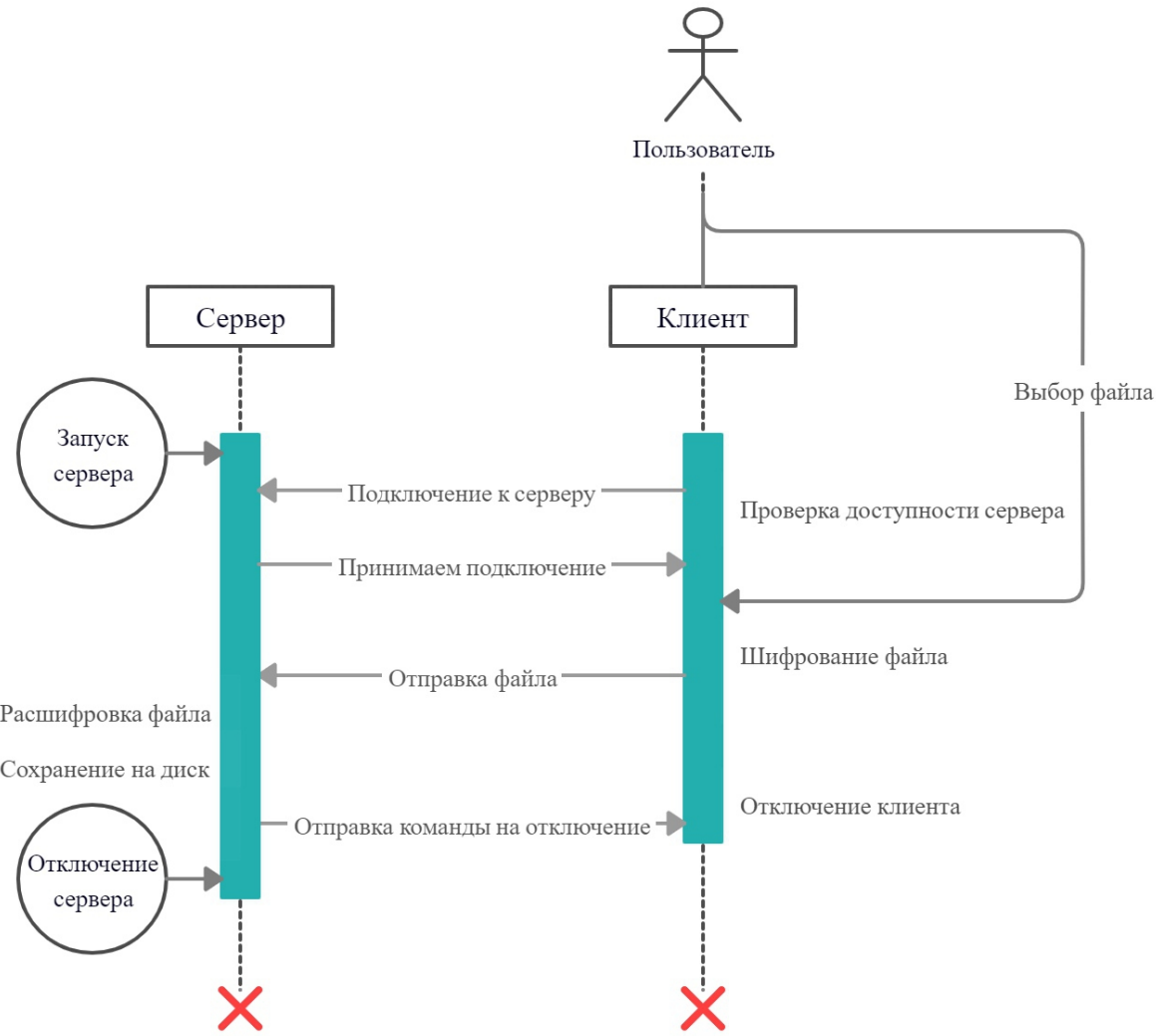


Рисунок 32 - UML-диаграмма последовательности

Приложение D.8 – UML-диаграмма развертывания

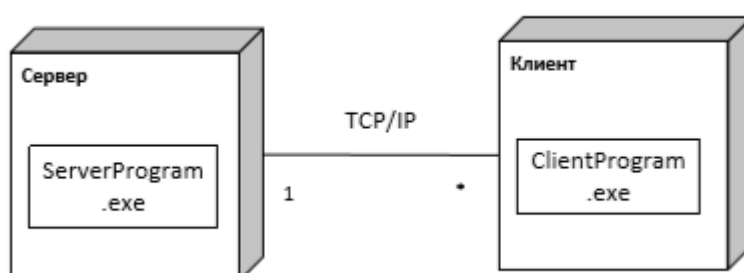
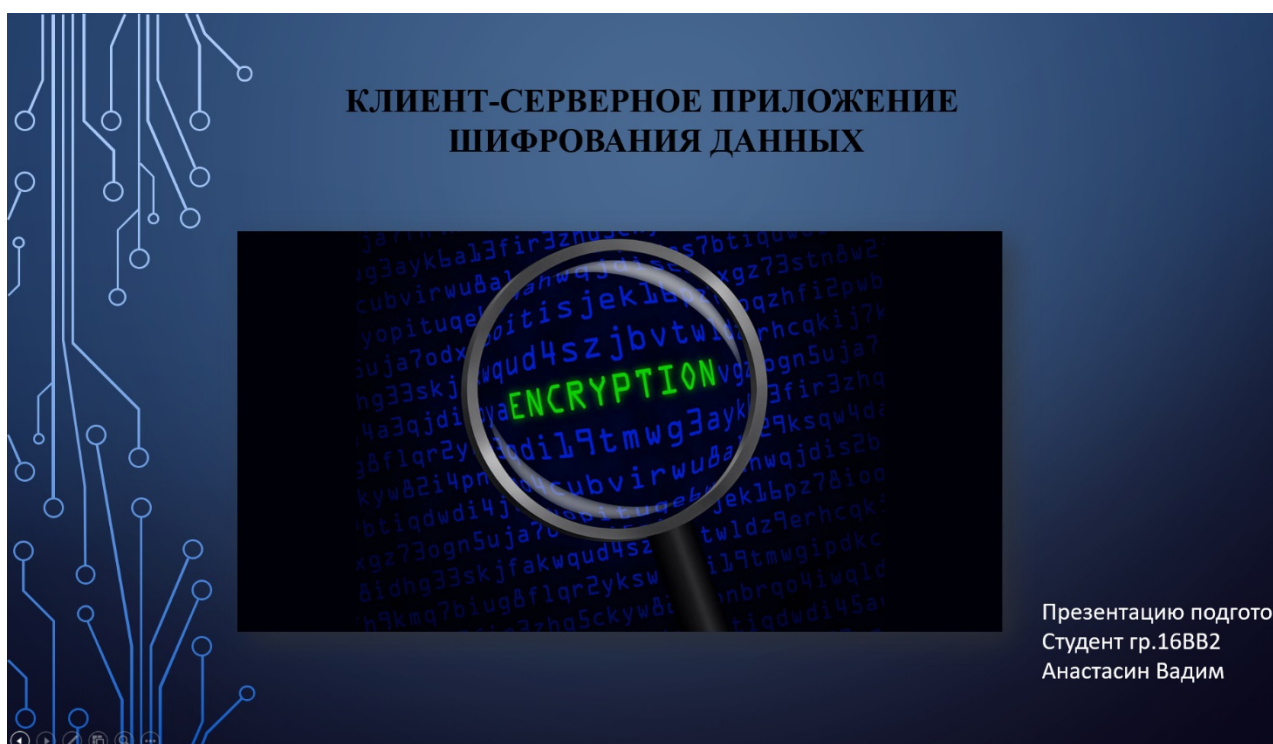


Рисунок 33 - UML-диаграмма развертывания

Приложение Е

Презентация



ПОСТАНОВКА ЗАДАЧИ

Требуется разработать клиент-серверное приложение, позволяющее пользователю шифровать или расшифровать выбранный файл.

К приложению предъявляются следующие требования:

1. Клиент – серверное приложение должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему.
2. Клиентское приложение должно обеспечивать подключение к серверу.
3. Клиентское приложение не должно работать без подключения к серверной части.
4. Серверное приложение должно иметь понятный графический интерфейс.
5. В окне серверного приложения должна быть возможность выбора алгоритма шифрования, по которому необходимо зашифровать выбранный файл.
6. Клиентское приложение должно быть консольным.
7. Приложение должно выполнять шифрование файла согласно заданному алгоритму.
8. В приложение должны присутствовать 2 алгоритма шифрования данных.
9. Взаимодействие между клиентом и сервером должно выполняться с использованием протокола TCP/IP.
10. Взаимодействие пользователя с клиент-серверным приложением должно осуществляться с помощью клавиатуры и мыши.
11. Операционная система – MS Windows 10 64 bit.

ВВЕДЕНИЕ



Криптографический отдел «комната №40»



Шифровальная машина
«Энигма»

АКТУАЛЬНОСТЬ ИСПОЛЬЗОВАНИЯ ШИФРОВАНИЯ ДАННЫХ



АЛГОРИТМЫ ШИФРОВАНИЯ ДАННЫХ



Схема симметричного шифрования



Схема асимметричного шифрования

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

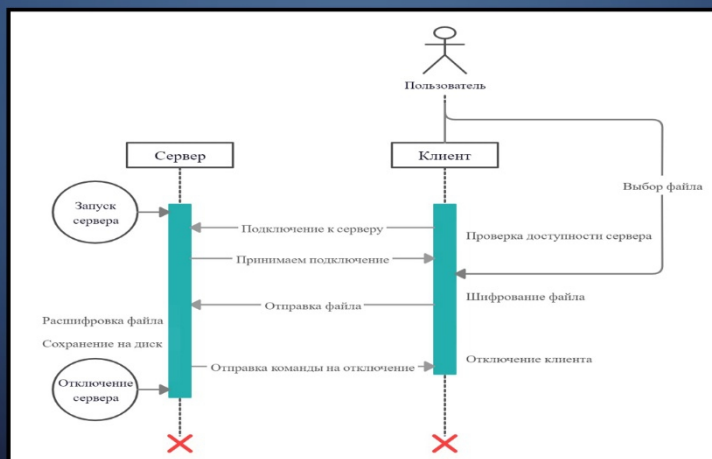


Диаграмма последовательности

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ



Диаграмма вариантов использования сервера

РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

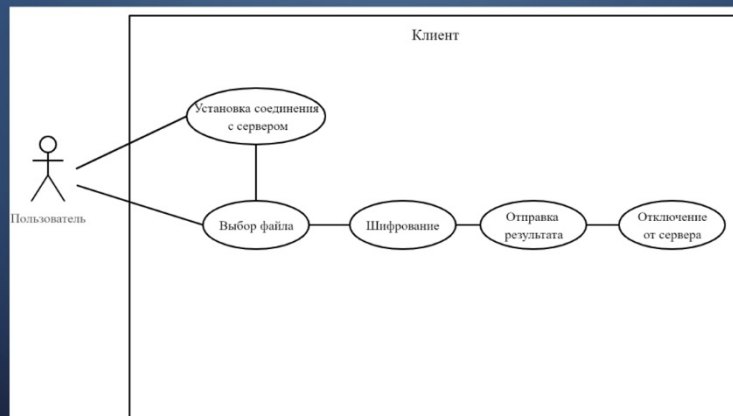


Диаграмма вариантов использования клиента

АЛГОРИТМ ШИФРОВАНИЯ XOR

		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

XOR обладает следующими свойствами:

$$a \text{ XOR } 0 = a$$

$$a \text{ XOR } a = 0$$

$$a \text{ XOR } b = b \text{ XOR } a$$

$$(a \text{ XOR } b) \text{ XOR } b = a$$

В языке JAVA (а также в C, C++, C#, Ruby, PHP, JavaScript) операция обозначается символом «^».

АЛГОРИТМ ШИФРОВАНИЯ RSA+AES

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

СПАСИБО ЗА ВНИМАНИЕ!