

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА—Российский технологический университет»**

**РТУ МИРЭА**

**Институт комплексной безопасности и специального приборостроения**

(наименование института, филиала)

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

(наименование кафедры)

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

по дисциплине «Технологии программирования»

(*наименование дисциплины*)

**Тема курсового проекта (работы)** «TCP/IP чат с RSA шифрованием»

**Студент группы Халиков А.А., БСБО-17-20 \_\_\_**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(*Ф.И.О., учебная группа*) (*подпись студента*)

**Руководитель курсового**

**проекта (работы) \_\_\_\_\_\_\_\_\_к.т.н., доц. Кашкин Е.В.\_\_ \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(*Ф.И.О., должность, звание, ученая степень*) (*подпись руководителя*)

**Рецензент**

**(при наличии)** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(*Ф.И.О., должность, звание, ученая степень*) (*подпись рецензента*)

**Курсовой проект (работа)**

**представлен(а)**

**к защите** «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Допущен(а) к защите** «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Москва 2021 г.**



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА—Российский технологический университет»**

**РТУ МИРЭА**

**Институт комплексной безопасности и специального приборостроения**

(наименование института, филиала)

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

(наименование кафедры)

Утверждаю

Заведующий кафедрой: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(*подпись*) *(Ф.И.О.)* (*Ф.И.О.*)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**ЗАДАНИЕ**

**на выполнение курсового проекта (работы)** по дисциплине

«Технологии программирования»

**Тема курсового проекта (работы)** «TCP/IP чат с RSA шифрованием»

**Студент**  Халиков Александр Александрович **Группа** БСБО-17-20\_

**Тема**  «TCP/IP чат с RSA шифрованием» 2

**Исходные данные****:** нет т

**Перечень вопросов, подлежащих обработке, и обязательного графического материала:**

|  |
| --- |
| -Титульный лист |
| -Содержание |
| -Введение |
| - 1. Глава 1. Анализ существующих программных решений TCP-чатов |
| - 2. Глава 2. Проектирование |
| - 3. Глава 3. Реализация решения |
| - Заключение |
| - Список использованной литературы |
| - Приложение 1 |
|  |

**Срок предоставление к защите курсового проекта (работы)** до«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Задание на курсовую работу выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(*подпись руководителя*) (*Ф.И.О. руководителя*)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Задание на курсовую работу получил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(*подпись обучающегося*) (*Ф.И.О. обучающегося)*

**Оглавление**

[Введение 4](#_Toc74314370)

[Постановка задачи 4](#_Toc74314371)

[Глава 1. Анализ существующих программных решений TCP-чатов 5](#_Toc74314372)

[1.1.1 MyChat 6](#_Toc74314373)

[1.1.2 RSend 6](#_Toc74314374)

[1.2 Основные методы шифрования сообщений 8](#_Toc74314375)

[1.2.1 Симметричное шифрование 9](#_Toc74314376)

[1.2.2. Ассиметричное шифрование 10](#_Toc74314377)

[1.2.3. RSA-шифрование 11](#_Toc74314378)

[1.3. Транспортный протокол TCP 12](#_Toc74314379)

[Глава 2. Проектирование 14](#_Toc74314380)

[2.1. Структура проекта 14](#_Toc74314381)

[2.2 Блок-схемы шифрования данных 15](#_Toc74314382)

[2.3. Форматы представления данных 18](#_Toc74314383)

[Глава 3. Реализация решения 19](#_Toc74314384)

[3.1 Описание работы чата 19](#_Toc74314385)

[Приложение 1. 27](#_Toc74314386)

[Заключение 29](#_Toc74314387)

# Введение

Выбранная тема для исследования является актуальной, по причине того, что в наше время в современном обществе фактор общения между людьми является основным. Целью данной курсовой работы станет создание приложения на основе клиент-серверной архитектуры, в котором пользователи смогут обмениваться сообщениями. Также в данном проекте будет реализовано RSA-шифрование сообщений, которое будет написано с нуля (без использования библиотек), для шифрования символа будет выбираться его номер из соответствующей таблицы кодировки. Чат должен в полной мере и эффективно решать проблему обмена сообщениями между пользователями, предоставляя им для этого удобный интерфейс.

Постановка задачи

Необходимо разработать программу, которая будет представлять собой клиент-серверное приложение с возможностью обмена сообщениями в зашифрованном виде (путем RSA шифрования).

Клиент будет реализован посредствам платформы для построения клиентских приложений WPF, соответственно для разметки самого интерфейса будет использован язык разметки XAML. Сервер же будет представлять собой консольное приложение на .NET Framework. Клиенты видят обычный текст, но сам сервер принимает уже зашифрованное сообщение, так что на стороне сервера как-то расшифровать сообщения не получится.

В итоге приложение должно работать следующим образом: для начала будет необходимо запустить сервер, для этого нужно будет ввести IP адрес и порт через двоеточие (всю необходимую информацию можно найти в настройках сети), но также сервер может запускаться и на стандартных настройках, которые сам определит. После успешного запуска сервера, запускается два или более клиентов и пользователям будет необходимо обменяться ключами, которые генерируются автоматически при создании нового клиента, также пользователь должен будет ввести адрес хоста и порт (адрес запущенного сервера), и свой логин (логины у всех клиентов должны быть разные, если два пользователя введут одинаковые логины, будет выдана соответствующая ошибка). Затем каждый из клиентов нажимает кнопку “Подключиться” и после этого можно начать обмен сообщениями. По завершении переписки клиенты должны нажать кнопку “Disconnect”, чтобы отключиться от чата и выйти из программы.

# **Глава 1. Анализ существующих программных решений TCP-чатов**

В рамках данной главы будут рассмотрены общие принципы работы чатов, основанных на клиент-серверной архитектуре, также будет представлен их сравнительный анализ. Далее будут рассмотрены основные методы шифрования сообщений, приведены примеры симметричного и ассиметричного шифрования. Особое внимание будет уделено RSA-шифрованию, так как именно оно будет использовано в этом проекте.

Аналогичными проектов очень много, реализованы они на разных языках программирования и с использованием разных технологий, так как многие начинающие программисты разрабатывают свои чаты для понимая работы передачи данных по сети с помощью одного из самых популярных протоколов TCP/IP. Хотя не во многих проектах реализовано ассиметричное шифрование RSA.Однако и в наши дни существуют чаты, созданные целыми компаниями и имеющие свою аудиторию, работают они также по схеме клиент-сервер, это позволяет использовать их в сетях со сложной конфигурацией, а также управлять клиентскими приложениями (например, Mychat, RSend). Конечно, такие приложения не пользуются особой популярностью, так как существуют более продвинутые и удобные сервисы для мгновенного обмена сообщениями – мессенджеры (Telegram, WhatsApp).

### MyChat

MyChat — клиент-серверное программное обеспечение для передачи текстовых сообщений с гарантированной доставкой. Последняя версия данного чата вышла в начале 2021 года. Незарегистрированная версия поддерживает до 20 одновременных подключений пользователей к серверу чата. В возможности этого чата входит:

1. Использование SSL для шифрования передаваемых сообщений
2. Достаточно широкие возможности администрирования сервера и управления клиентами.
3. Авторизация пользователей на сервере и централизованное хранение личных данных.
4. Отправка [офлайн](https://ru.wikipedia.org/wiki/%D0%9E%D1%84%D0%BB%D0%B0%D0%B9%D0%BD) сообщений для пользователей, которые отключены от сервера.
5. Файловое хранилище на сервере, разделенное на персональные и общее хранилища
6. Встроенный FTP сервер, используемый для передачи файлов и обновлений.
7. Программа работает на компьютерах под управлением ОС Windows, Linux, Mac, а также Android клиент для подключения с мобильных платформ
8. Веб-чат для общения пользователей без установки приложения, используя браузер.

И т.д.

### RSend

Rsend­– это программа, предназначенная для того, чтобы в сетях в режиме онлайн между объемным пользователей WAN и LAN обмениваться мгновенными сообщениями.

Rsend — является клиент-серверной программой, основанной на базе технологии .Net Remoting, которая в свою очередь использует протокол TCP. В подсети за маршрутизатором, данная программа работает успешно, потребляя при этом достаточно мало системных ресурсов и минимум сетевого трафика. Также Rsend имеет функцию отправлять массовые сообщения или же индивидуальные письма. Для того, чтобы отправить массовые рассылки, приложение может строить списки по разным критериям, такие как: имя компьютера, имя пользователя, личная группа или домен. Из-за того, что используется асинхронный многопоточный режим для передачи данных, сокращается время передачи данных, которое варьируется от пары секунд до одной (зависит это от платформы сервера, состояния удаленного сервера, количества пользователей)

В Rsend для удобного пользования реализованы дополнительные функции:

а) Для того, чтобы построить гранулированные списки клиентов, получающих сообщения, используются внутренний фильтр и поиск (встроенный в приложении)

б) Быстрый ответ пользователю на входящие сообщения

в) Создание запроса, который подтверждает о том, то сообщение доставлено

г) Программная функция «горячих» клавиш

Архитектура клиент-сервера реализуется с помощью технологии .Net Remoting и Visual C#, чаще всего .Net Remoting использует стандартную схему: сторона клиента с сервером по ТСР (или НТТР) каналу устанавливает соединение, далее периодически обращается к функции получения/отправки данных нужных ему. Этот подход разрешает много требований, поставленных в задаче, но есть исключения, которые не смогут быть реализованы: быстрая доставка и минимальный трафик интернета.

Клиент —> Сервер

Для того, чтобы реализовать эти два требования, стандартный сценарий был изменен: сторона клиента с сервером по ТСР каналу устанавливает соединение и далее переходит в режим ожидания. Когда же в сервер поступают новые сообщения от пользователя, сервер, по заданным спискам получателей, подсоединяется обратно к тем клиентам и вызывает определенную функцию на удаленном пользователе(сервер и клиент меняются местами)

Таблица 1. Сравнение трех чатов

|  |  |  |  |
| --- | --- | --- | --- |
| Название  Критерии  сравнения | RSend | MyChat | TCP-чат |
| Архитектура | Клиент-Сервер  (централизованный) | Клиент-Сервер  (централизованный) | Клиент-Сервер  (децентрализованный) |
| Шифрование | нет | SSL | RSA |
| Язык написания | C# | С++ | С# |
| Поиск и фильтры | Есть | Есть | Нет |

Исходя из Таблицы 1, можно сделать вывод, что главным отличием TCP-чата является отсутствие централизованного сервера – приложение будет децентрализованным, то есть любой человек сможет сам создать сервер. А после отключения от сервера вся переписка будет стерта с сервера – сделано это для безопасности (но будет реализована функция, с помощью которой весь чат можно будет перенести в .txt или .json файл и сохранить на свой компьютер).

**1.2 Основные методы шифрования сообщений**

С каждым днем мы все больше слышим о конфиденциальности, защите данных и шифровании. В данном разделе будут главные методы шифрования, которые используются в чатах/мессенджерах. В кибербезопасности существует ряд вещей, которые могут беспокоить пользователей, когда затрагивается тема тех или иных данных. К ним особенно относятся доступность информации, конфиденциальность и целостность.

Главная задача конфиденциальности в том, что данные не должны быть получены и прочитаны неавторизованными клиентами.

Целостность информации –этотермин, суть которого состоит в том, что информация сто процентов должна оставаться нетронутой, а внесение изменений невозможно.

Доступность информации – это возможность получение доступа к данным, когда это необходимо.

Основные способы шифрования:

1. Симметричное
2. Асимметричное
3. Хеширование
4. Цифровая подпись
   * 1. **Симметричное шифрование**

Для перехода к этому способу шифрования, нужно понимать, что означает слово «шифрование». Шифрование — это видоизменение информации в стремлении скрытия от незарегистрированных пользователей, но и в то же время, доступ открыт для зарегистрированных пользователей.

Для того, чтобы правильно расшифровывать и зашифровывать данные, нужны такие вещи, как: определенный ключ для дешифровки и имеющиеся данные. В нашем случае симметричного шифрования, ключ для шифровки и дешифровки данных — одинаковый.

Работа этого ключа происходит следующим образом:

Внутри системы шифрования имеется некоторая математическая последовательность. В цифровой «вход» отправляется заданный пароль и отправленный текст, фото и пр., потом информация шифруется и отправляется пользователю. При доставке сообщения, срабатывает снова та же последовательность, только в обратном порядке и осуществляется дешифровка, с помощью того самого пароля.

Говоря простым языком, зная пароль, безопасность этого способа практически нулевая. Поэтому пароль должен складываться из разных регистров, используя буквы, чтобы максимально усложнить его. Вопреки некоторым ограничениям, симметричное шифрование достаточно распространено, ввиду своей простоты и быстродействия.

Примеры некоторых симметричных шифров:

1. XTEA - наиболее простой в реализации алгоритм
2. AES – американский стандарт шифрования
3. DES – стандарт шифрования данных в США до AES
   * 1. **Ассиметричное шифрование**

Симметричное шифрование обладает некоторыми недостатками, которые состоят в следующем: например, необходимо через Интернет отослать те или иные данные. Когда для шифрования и расшифровки всех данных необходим один и тот же ключ, тогда следует, что для начала нужно отправить этот ключ. Это говорит о том, что отправить ключ придётся с использованием небезопасного соединения. Следовательно, таким образом ключ может быть перехвачен и в дальнейшем использован злоумышленниками. Именно для того, чтобы предотвратить подобные исходы, изобрели асимметричное шифрование.

В отличие от симметричного шифрования, для использования асимметричного, нужно сгенерировать именно два математически-связных ключа. Первый является приватным ключом, доступ к которому получаете только отправляющий. Второй же — это открытый ключ, то есть общедоступный и его видят все.

Один из примеров использования ассиметричной шифровки, это отправка сообщений со стороны сервера и со стороны пользователя. У одного и другого имеется два ключа шифрования: публичный и приватный. Ранее мы узнали, что ключи математически связаны, то есть сообщение, которое было зашифровано с помощью приватного ключа, можно будет расшифровать только с использованием публичного смежного ключа. Поэтому для начала общения, нужно будет обменяться этими ключами.

Чтобы понять, что открытый ключ какого-нибудь сервера относится именно к этому серверу, необходимо использовать инфраструктуру открытых ключей (PKI), это один из самых популярных методов решения проблемы (он и используется в Интернете). В ситуации с веб-сайтами существует так называемый Центр сертификации, у которого имеется каталог большинства сайтов, точнее всех на которые вдавались сертификаты и открытые ключи. Когда пользователь подключается к веб-сайту его открытый ключ сначала проверяется центром сертификации.

Основные алгоритмы ассиметричного шифрования:

1. Rivest—Shamir—Adleman (RSA) (наиболее распространенный)
2. Ellipticcurvecryptosystem (ECC)
3. Diffie—Hellman (DH)
4. ElGamal
   * 1. **RSA-шифрование**

RSA (Rivest-Shamir-Adleman) является практически первой описанной криптосистемой с публичным ключом, она распространена в целях безопасности передачи клиентских данных. В этой системе ключом шифрования является публичный ключ, который различен с ключом расшифровки(он приватный). В RSA асимметричность основана на практике сложной факторизации умножения двух крупных простых чисел(проблема факторинга). RSA является аббревиатурой из двух фамилий: Рона Ривеста, Ади Шамира и Леонарда Адлемана — они первые смогли написать этот алгоритм еще в 1978 году. Конкуренцию составил математик из Англии, которого звали Клиффорд Кокс. Этот человек работал на Англии в разведке под начальством связи правительства. Он создал похожую систему в 1973 году, но это было засекречено до 1997 года.

Тот, кто использует шифрование RSA создает и отправляет публичный ключ на базе простых больших двух чисел. Эти простые числа засекречены, потому что тот, кто знает эти числа, могут запросто расшифровать сообщения. Любой пользователь может знать публичный ключ, но, если он не знает вторую составляющую метода шифрования RSA(простые числа), расшифровать ничего не получится. В RSA существует проблема нарушения шифрования, и этот вопрос остается открытым и по сей день.

Шифрование методом RSA не часто используют для прямого шифрования данных пользователя, потому что это достаточно медленная процедура. Метод шифрования RSA сначала передает симметричные ключи (они выполняют большое количество операций шифровки и дешифровки на более высокой скорости) и зашифрованные ключи.

* 1. **Транспортный протокол TCP**

В наше время значительное количество трафика в сети посредствам

протокола TCP (Transmission Control Protocol). Данный протокол гарантирует доставку данных, так как имеет механизм контроля переданных данных, ипоэтому используется во многих современных интернет-приложениях, таких как электронная почта, WorldWideWeb (WWW).

Алгоритм работы TCP следующий:

1. Создается трехэтапное рукопожатие двух узлов. На этом этапе происходит согласование между узлами двух чисел — для каждого из узлов формируется начальные номера первого пакета.
2. Узлы, при каждой отправке пакетов, один за одним номеруют их, это значит, что всем сетевым пакетам присваивается только один номер из последовательных.
3. Для каждого из пакетов, в дополнении к номеру, рассчитывается итоговая сумма. При получении пакета снова идет высчитывание контрольной суммы для принятых данных. Эти суммы должны совпадать, потому что это определяет поврежден файл или нет.
4. Из-за того, что все пакеты пронумерованы, при получении можно проверить целостность данных. Если произошла потеря пакетов, отправляется повторный запрос.
5. Также, отправка повторного запроса происходит, когда не совпала контрольная сумма.

Таблица 2. Заголовок TCP-пакета

|  |  |  |
| --- | --- | --- |
| Порт источника | | Порт приемника |
| Порядковый номер | | |
| Порядковый номер (подтверждение) | | |
| Размер заголовка | Флаги | Размер приемного окна |
| Контрольная сумма | | Дополнительные параметры |
| Дополнительные параметры (необязательно) | | |
| Данные (необязательно) | | |

32 бита

Исходя из данных Таблицы 2 можно сделать вывод, о том, что если использовать 2 таких механизма, как контрольная сумма и последовательная нумерация каждого из двух и более пакетов получится достигнуть безопасности передачи данных и возможности организовать их в правильную последовательность независимо от того, в каком порядке они доставлены. Всё это возможно с помощью заголовков TCP пакета.

**Глава 2. Проектирование**

В данной главе будет рассмотрено как именно работают алгоритмы шифрования данных, будет представлена структура проекта и форматы представления данных.

**2.1. Структура проекта**

Программное решение будет представлять из себя клиент-серверное приложение, в котором:

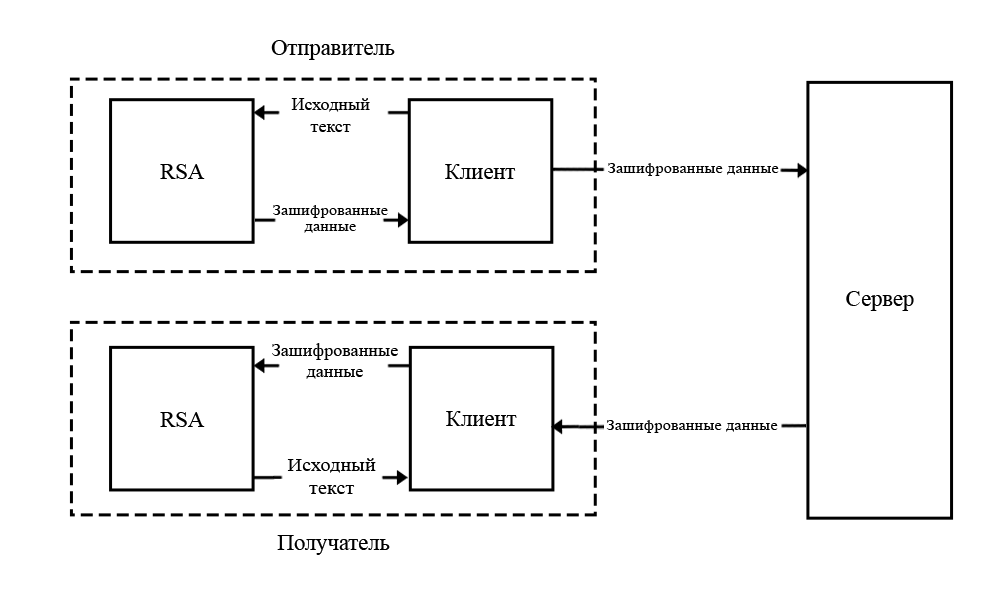
1. Есть библиотека для шифрования и расшифровки данных – RSA
2. Клиент –WPF-приложение, на вход которому будут поступать данные об открытом ключе второго клиента, уникальный nickname в пределах одной сессии, IP-адрес хоста, на котором
3. Сервер – консольное приложение, в которое необходимо ввести данные об IP-адресе, на котором он будет запущен и порт, который он будет прослушивать. 

Рисунок 1 – Модель работы программы

Стоит учитывать некоторые особые случаи: во-первых, если попытаться подключиться к сессии, в которой уже имеется пользователь

с таким логином, то соединение оборвётся. Во-вторых, если клиенты обменяются некорректными ключами, то соединение разорвётся. В-третьих, изменить исходные данные для подключения можно лишь после отключения от сессии.

* 1. **Блок-схемы шифрования данных**

Так как RSA шифрование базируется на факторизации больших простых чисел, данный алгоритм подразумевает следующие этапы:

1. Генерация ключей (Создание открытой {e, n} и закрытой {d, e} пары)
2. Передача ключей
3. Шифрование
4. Расшифровка

После инициализации ключей обоих участников, необходимо обменяться парами публичных ключей. Далее, чтобы второй участник смог расшифровать сообщение от первого, первый должен шифровать на открытом ключе второго. Аналогичная ситуация при расшифровке для первого участника. Таким образом, RSA алгоритм относится к шифрованию асимметричным способом (рис. 2, 3, 4).

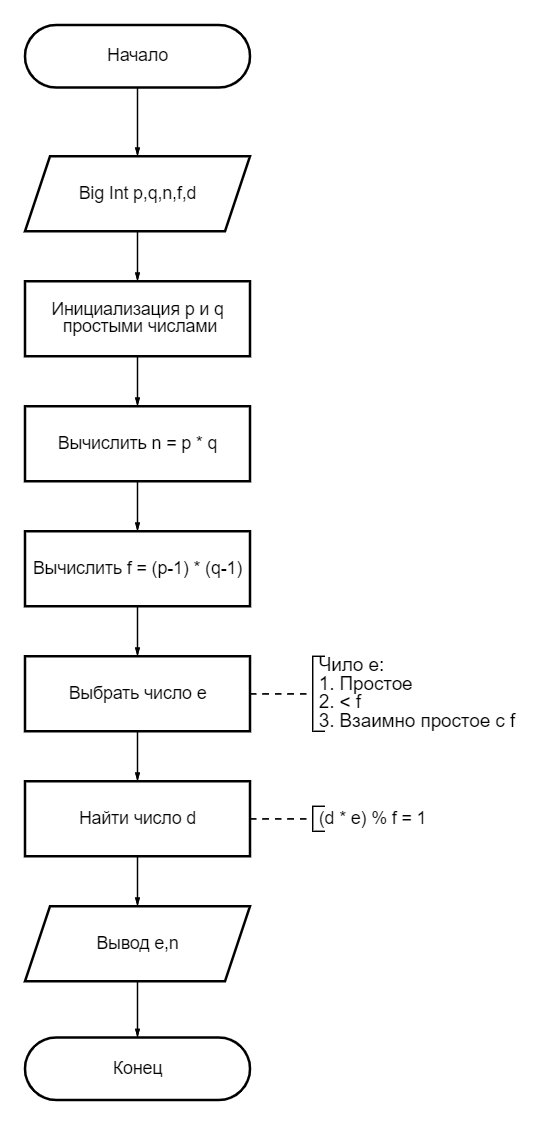


Рисунок 2 – Инициализация ключей

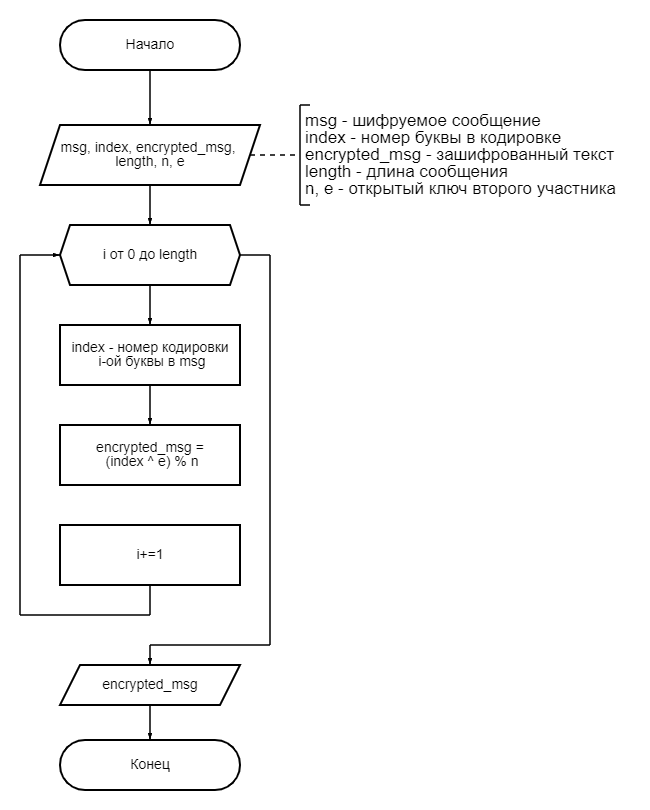


Рисунок 3 – Алгоритм шифрования текста

Для того, чтобы получить зашифрованный текст, необходимо знать номер каждой буквы в кодировке UNICODE, далее в цикле for присваивать каждой букве свой код и зашифровывать ее с помощью формулы, показанной на рис. 3 (encrypted\_msg=…), т.е. первый участник шифрует сообщение на открытых ключах своего собеседника. Далее записывать получившееся число в массив с набором зашифрованных символов. Для того, чтобы расшифровать сообщение собеседникам необходимо обменяться ключами.

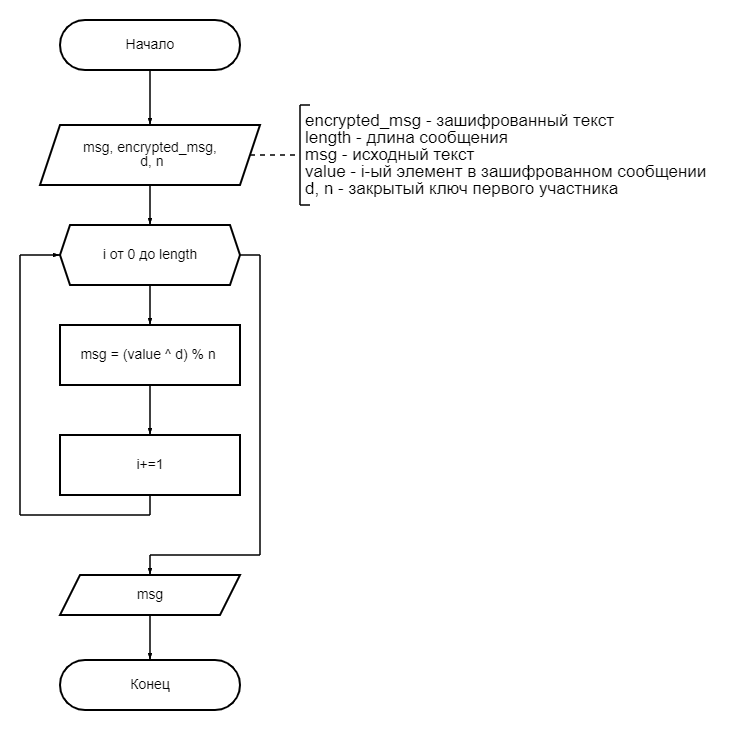


Рисунок 4 – Алгоритм расшифровки данных

**2.3. Форматы представления данных**

В программе при форматировании и передаче данных между клиентами используются следующие классы (рис. 5):

1. Client\_Base – абстрактный класс, содержащий id и логин(имя) клиента
2. Client\_Connected – класс на основе абстрактного класса Client\_Base, который содержит строку с хостом и портом, т.е. с информацией об источнике подключения
3. Client\_Server – класс на основе абстрактного класса Client\_Base, который содержит ссылку на подключённого клиента
4. Message – зашифрованное сообщение
5. Masbox – сообщение для вывода в UI
6. Enc\_Obj – передаваемое между клиентами зашифрованное сообщение

**Глава 3. Реализация решения**

В рамках данной главы разработано работающее решение TCP чата с ассиметричным шифрованием RSA. Оно направлена на решение задач генерации ключей пользователей, их проверки и подключения между собой, c целью обмена зашифрованными сообщениями.

**3.1 Описание работы чата**

Программная реализация разрабатываемой системы состоит их двух

частей:

1. Клиентской части системы;

2. Серверной части системы.

Клиентская часть системы, представляющая из себя десктопное приложение, разработана с использованием технологии WPF (Windows Presentation Foundation), являющийся частью экосистемы платформы .NET и представляющая собой подсистему для построения графических интерфейсов.

Серверная часть системы представляет из себя консольное приложение с возможностью запуска сервера на системных настройках и отображения действий пользователей.

Так как система децентрализована, т.е. не имеет одного центрального сервера, перед началом работы одному из собеседников необходимо запустить файл EncryptedChat.Server.exe, если пользователь запустит его через cmd, то сможет задать собственные настройки, но по умолчанию сервер берет настройки системы.

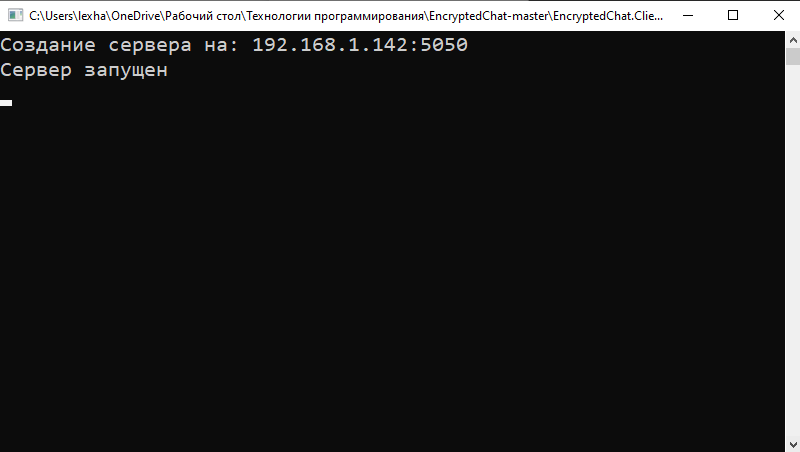


Рисунок 5. – Результат работы программы после запуска

При запуске происходит инициализация метода Main, если пользователь задал настройки сам, то парсится IP адрес и его порт, и если все верно, то происходит создание экземпляра класса и в его конструктор передаются значения введенных значений: IP адреса и порта.

static void Main(string[] args)

{

if (args.Length == 2 &&

IPAddress.TryParse(args[0], out IPAddress host) &&

int.TryParse(args[1], out int port))

{

new Server(host, port).Start();

}

else

{

new Server(GetLocalIPAddress(), 5050).Start();

}

}

public static IPAddress GetLocalIPAddress()

{

var host = Dns.GetHostEntry(Dns.GetHostName());

foreach (var ip in host.AddressList)

if (ip.AddressFamily == AddressFamily.InterNetwork)

return ip;

throw new Exception("No network adapters with an IPv4 address in the system!");

}

}

Листинг 1. – Функция запуска сервера

static TcpListener \_listener;

static List<ServerClient> \_clients;

internal Server(IPAddress host, int port)

{

\_listener = new TcpListener(host, 5050);

\_clients = new List<ServerClient>();

Console.WriteLine($"Создание сервера на: {host}:{port}");

}

Листинг 2. – конструктор класса Server

Далее сервер в бесконечном цикле ждет подключения новых пользователей, и, если кто-то подключается – под него выделяется свой поток и происходит инициализация необходимых модулей для работы с TCP соединением. Если имя не повторяется, с теми которые уже есть на сервере и все остальные данные введены верно – выводится сообщение об успешном подключении.

while (true)

{

var client = \_listener.AcceptTcpClient();

Task.Factory.StartNew(() =>

{

var stream = client.GetStream();

var bf = new BinaryFormatter();

var serverClient = new ServerClient();

var concreateClient = bf.Deserialize(stream) as ConnectedClient;

if (\_clients.Exists(c => c.Login == concreateClient.Login))

{

// Если клиент с таким именем уже есть

}

else

{

serverClient = new ServerClient(client, concreateClient.ID, concreateClient.Login,concreateClient.ContactPhoto );

\_clients.Add(serverClient);

WriteSignalAboutConnection(concreateClient);

SendToAllClients(concreateClient);

}

}

Листинг 3. – Прием клиентов на сервер

Пока на сервере есть подключенные клиенты, выполняется десериализация потока сообщений и отправка их клиентам.

while (client.Client.Connected)

{

try

{

var encObject = bf.Deserialize(stream) as EncryptedObject;

WriteTextToConsole(encObject);

SendToAllClients(encObject);

}

catch (Exception e)

{

if (client.Client.Poll(0, SelectMode.SelectRead))

{

var buff = new byte[1];

if (client.Client.Connected == false)

{

client.Client.Disconnect(true);

}

}

WriteExceptionToConsole(e);

\_clients.RemoveAll(x => !x.Client.Connected);

Thread.CurrentThread.Abort();

}

}

\_clients.Remove(serverClient);

}, TaskCreationOptions.LongRunning);

Листинг 4. – Работа с клиентом

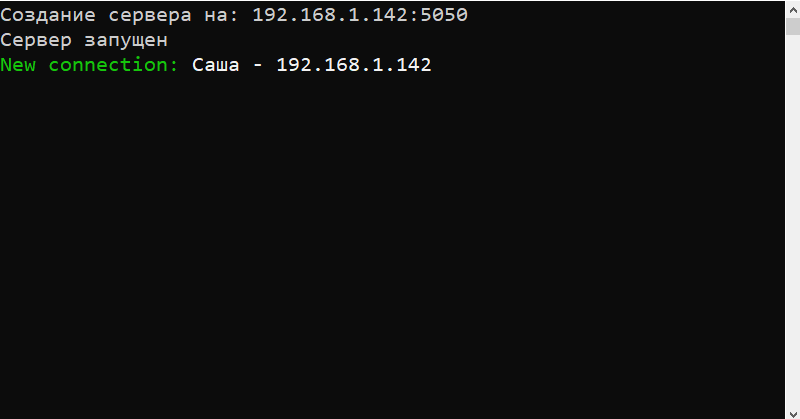


Рисунок 6. – Вывод консоли при подключении нового клиента

При запуске клиента пользователь видит интерфейс, в котором ему необходимо заполнить поля с ключами собеседника, так как присутствует RSA шифрование, IP адресом и портом сервера, и логином. Его собеседнику необходимо сделать тоже самое, ключи генерируются при запуске программы.

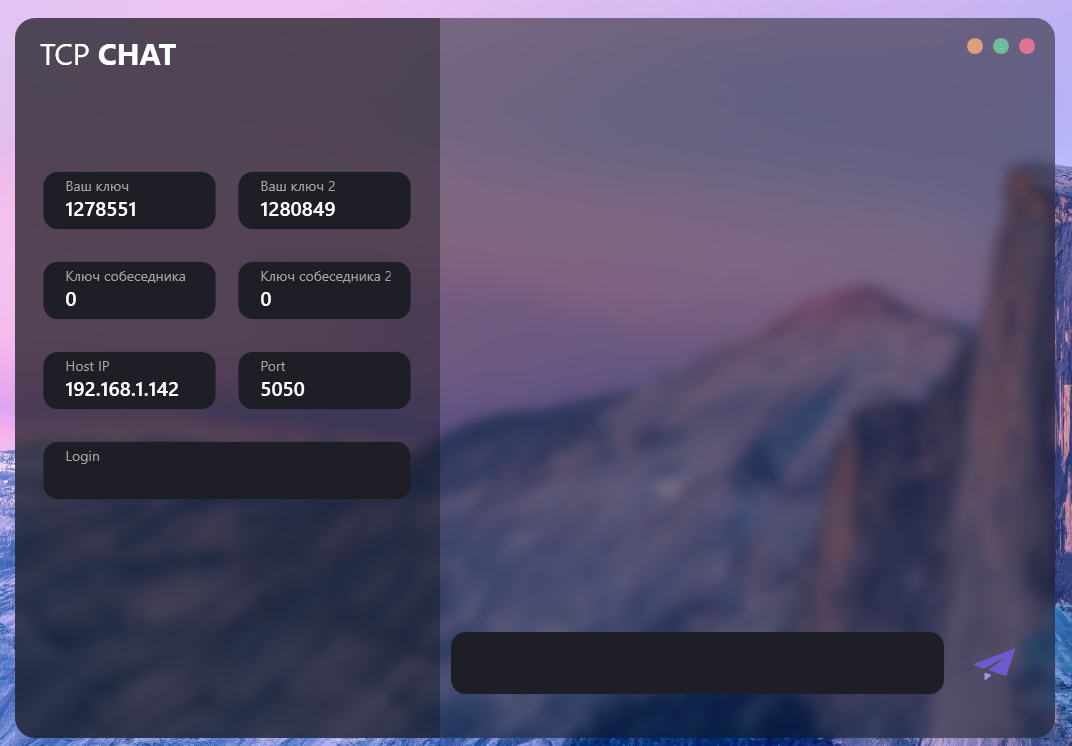


Рисунок 7. – Интерфейс программы при запуске

private RSA \_rsa = new RSA();

private TcpClient \_tcpClient;

private NetworkStream \_stream;

private ConnectedClient \_client;

public int LocalE => (int)\_rsa.e;

public int LocalN => (int)\_rsa.n;

public int RemoteE { get => remoteE; set { remoteE = value; OnPropertyChanged();}}

public int RemoteN { get => remoteN; set { remoteN = value; OnPropertyChanged();}}

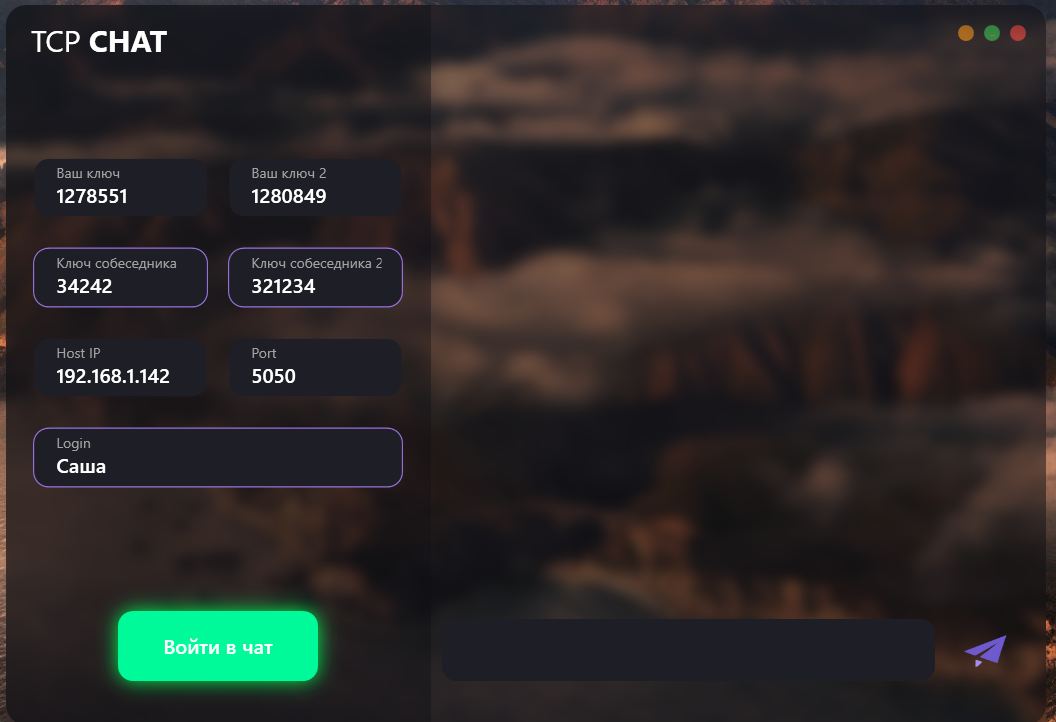
public string Host { get; set; }

public int Port { get; set; }

public string Login { get; set; }

Листинг 5. – Главный класс клиента

После успешного ввода данных появляется кнопка “Войти в чат”. По нажатии на эту кнопку выполняется метод Connect(). Если код выполняется без ошибок и клиент подключается – всем клиентам приходит сообщение о новом подключении.



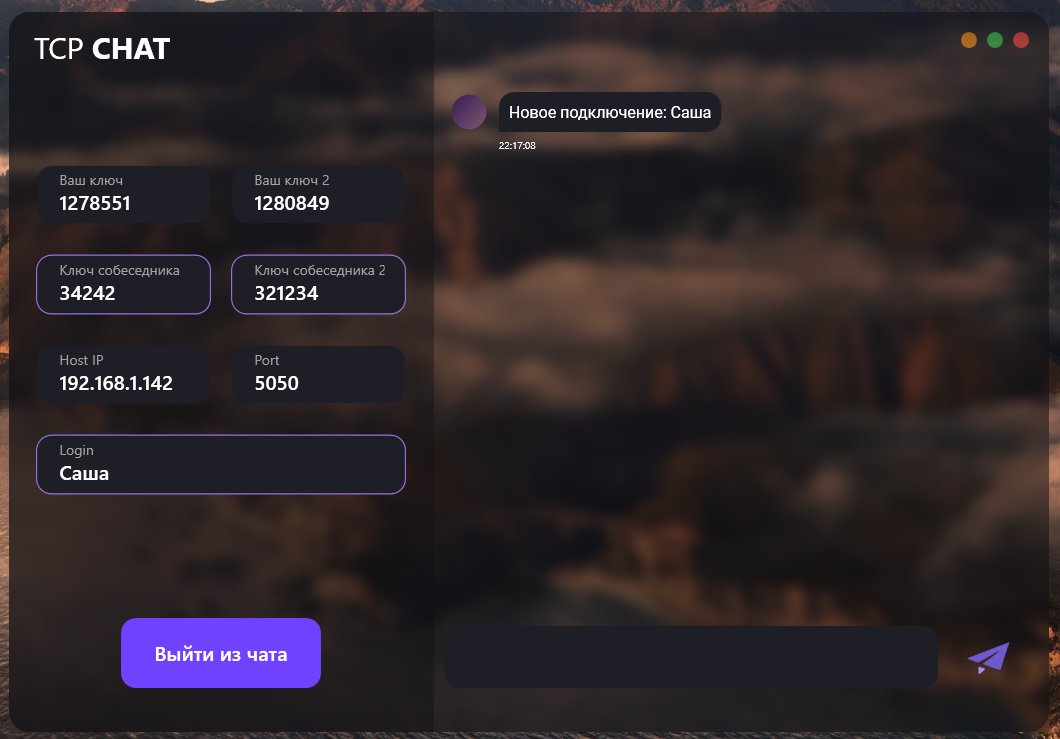


Рисунок 8-9. – Войти в чат и новое подключение

private void Connect()

{

\_tcpClient?.Close();

\_tcpClient = new TcpClient();

try

{

\_tcpClient.Connect(Host, Port);

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка подключения к серверу", MessageBoxButton.OK, MessageBoxImage.Information);

return;

}

\_stream = \_tcpClient.GetStream();

\_client = new ConnectedClient(GetLocalIPAddress().ToString(), Login, ContactPhoto);

Листинг 6. – Метод подключения к серверу

После успешного подключения пользователь может начать обмен сообщениями. Для это в поле ввода сообщения необходимо что-нибудь написать и нажать на кнопку отправки сообщения. По нажатии на эту кнопку вызывается метод SendMessage(). И сообщение шифруется (на стороне клиента), а затем сериализуетя и отправляется на сервер. На сервере отображается только зашифрованный текст. Метод шифрования и дешифрования представлен и остальные методы для работы с RSA представлены в приложении 1.

private void SendMessage()

{

try

{

var data = \_rsa.Encrypt(Text, RemoteE, RemoteN);

Dispatcher.CurrentDispatcher.Invoke(() =>

{

Messages.Add(new MessageItem() { Login = \_client.Login, Content = Text, SendTime = DateTime.Now, ContactPhoto=\_client.ContactPhoto });

});

var message = new EncryptedObject(new Message(data, DateTime.Now), \_client);

new BinaryFormatter().Serialize(\_stream, message); //отправляет на сервер

Text = string.Empty;

OnPropertyChanged(nameof(Text));

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка отправки сообщения", MessageBoxButton.OK, MessageBoxImage.Information); //обработчик ошибок отправки сообщения

return;

}

}

Листинг 7. – Метод отправки сообщения

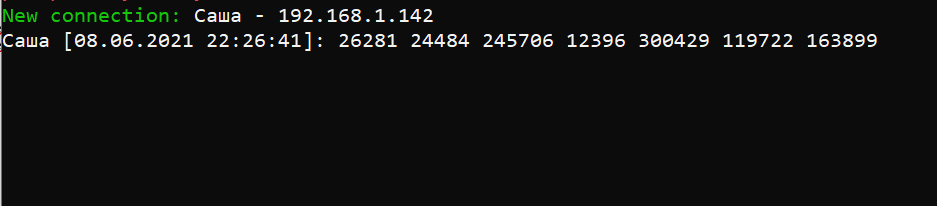


Рисунок 10. – Отображение отправленного сообщения на сервере

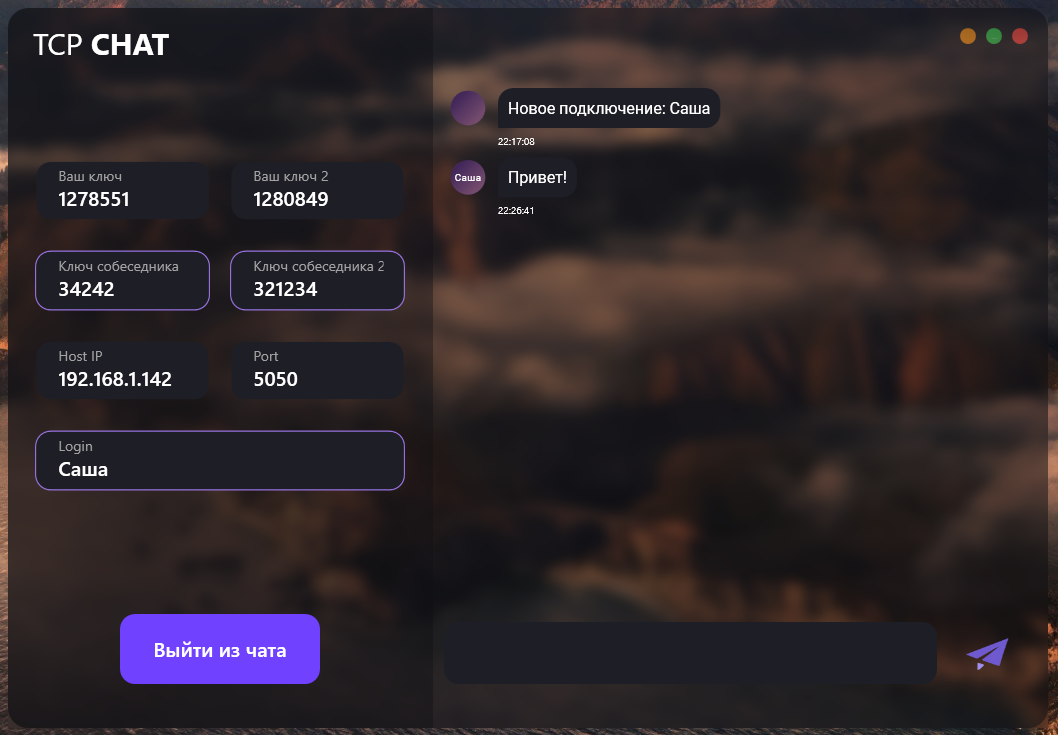


Рисунок 11. – Отображение отправленного сообщения в клиенте

По завершении диалога необходимо нажать кнопку “Выйти из чата”, после этого запустится метод “Disconnect”

private void Disconnect()

{

\_tcpClient?.Close(); //закрытие клиента

\_canConnect = true;

CanDisconnect = false;

OnPropertyChanged();

}

Листинг 8. – Метод отключение от сервера

# Приложение 1.

public class RSA

{

private bool Ready\_check;

private long local\_D;

public long p;

public long q;

public long n;

public long e;

public RSA()

{

Initialize();

}

public void Initialize()

{

p = PrimeNumberGenerator.Generate();

q = PrimeNumberGenerator.Generate();

n = p \* q;

var f = (p - 1) \* (q - 1);

e = GetPublicPartKey(f);

local\_D = GetPrivatePartKey(f, e);

Ready\_check = true;

}

#region Проверка готовности ключей и шифрование

public string[] Encrypt(string text, long publicE, long publicN)

{

if (!Ready\_check)

throw new ArgumentException("Method Initialize not called");

return Encode(text, publicE, publicN);

}

#endregion

#region Проверка наличия установленных ключей и расшифровка

public string Decrypt(string[] encrypted\_msg)

{

if (!Ready\_check)

throw new ArgumentException("Method Initialize not called");

return Decode(encrypted\_msg, local\_D, n);

}

#endregion

#region Шифрование методом RSA

private string[] Encode(string msg, long e, long n)

{

var encrypted\_msg = new List<string>();

BigInteger num;

foreach (var ch in msg)

{

int index = ch;

num = BigInteger.ModPow(index, e, n);

encrypted\_msg.Add(num.ToString());

}

return encrypted\_msg.ToArray();

}

#endregion

#region Дешифрирование методом RSA

private string Decode(string[] encrypted\_msg, long d, long n)

{

var strBuilder = new StringBuilder();

BigInteger num;

foreach (var item in encrypted\_msg)

{

var val = new BigInteger(Convert.ToInt64(item));

num = BigInteger.ModPow(val, d, n);

strBuilder.Append((char)num);

}

return strBuilder.ToString();

}

#endregion

#region Получить свой приватный ключ

private long GetPrivatePartKey(long f, long e)

{

long d = e + 1;

while (true)

{

if ((d \* e) % f == 1)

break;

d++;

}

return d;

}

#endregion

#region Получить свой публичный ключ для обмена

private long GetPublicPartKey(long f)

{

long e = f - 1;

while (true)

{

if (PrimeNumberGenerator.IsPrime(e) &&

e < f &&

BigInteger.GreatestCommonDivisor(new BigInteger(e), new BigInteger(f)) == BigInteger.One)

break;

e--;

}

return e;

}

#endregion

}

Листинг 8. – Класс RSA и работа его методов

# Заключение

В процессе создания приложения на основе клиент-серверной архитектуры – TCP/IP чата с ассиметричным RSA шифрованием, которое написано с нуля (без использования библиотек), была добавлена функция генерации ключей и их проверки. Чат в полной мере и эффективно решает проблему обмена сообщениями между пользователями, предоставляя им для этого удобный интерфейс.

**Список литературы**

1. Многопоточность C# [электронный ресурс]. URL: <https://metanit.com/sharp/tutorial/11.1.php> Дата обращения 05.04.21;  
2. Протокол TCP в C# и .NET [электронный ресурс]. URL: [https://metanit.com/sharp/net/4.1.php Дата обращения 10.05.21](https://metanit.com/sharp/net/4.1.php%20Дата%20обращения%2010.05.21);

3. Руководство по WPF [электронный ресурс] URL: <https://metanit.com/sharp/wpf/> Дата обращения 01.04.21;  
4. RSA: от простых чисел до электронной подписи [электронный ресурс]. URL: <https://habr.com/ru/post/534014> Дата обращения 10.04.21;  
5. Таненбаум, Э. Компьютерные сети / Э. Таненбаум. - СПб.: Питер, 2019. - 960 c.  
6. Криптография и главные способы шифрования информации [электронный ресурс]. URL: <https://proglib.io/p/methods-of-encryption/> Дата обращения 18.04.21;

7. MyChat [электронный ресурс]. URL: <https://nsoft-s.com/> Дата обращения 10.04.21;

8. RSend [электронный ресурс]. URL: <http://rsend.ru/> Дата обращения 10.04.21;

9. Симметричное шифрование [электронный ресурс]. URL: <https://encyclopedia.kaspersky.ru/glossary/symmetric-encryption/> Дата обращения 25.04.21;

10. Иллюстрация работы RSA на примере [электронный ресурс]. URL:<http://www.michurin.net/computer-science/rsa.html> Дата обращения 13.04.21;

11. Класс StringBuilder [электронный ресурс]. URL: <https://metanit.com/sharp/tutorial/7.3.php> Дата обращения 13.04.21;