Московский Авиационный институт

(Национальный исследовательский университет)

Институт №8
«Компьютерные науки и прикладная математика»

Кафедра 813

«Компьютерная математика»

Курсовая работа по дисциплине
«Основы криптографии»

Тема: «Разработка криптографического приложения на основе алгоритмов Benalo и RC6»

Студент: Тришин Д. А.

Группа: М8О-311Б-19

Преподаватель: Романенков А. М.

Оценка:

Дата:

Москва, 2022

Оглавление

Введение

В курсовой работе разработано клиент-серверное приложение “Чат”, позволяющее отправлять зашифрованные сообщения и файлы пользователям чата с последующей расшифровкой. Файлы, отправляемые пользователями, хранятся на сервере с возможностью последующего скачивания и расшифровки, а сообщения сразу доходят до всех клиентов чата. Для шифрования информации, передаваемой между клиентами, используется ключ симметричного шифрования алгоритма RC6, генерируемый отдельным приложением.

Теоретическая часть

Алгоритм Benalo

Описание

Криптосистема Бенало – модификация криптосистемы Гольдвассер-Микали. Основное их отличие состоит в том, что криптосистема позволяет зашифровывать блок данных единовременно, в то время как в схеме Гольдвассера и Микали каждый бит данных шифруется отдельно.

Разработана Джошем Бенало в 1988 году. Получила применение в системах электронного голосования.

Система является частично гомоморфной. Как и во многих криптосистемах с открытым ключом, эта система работает в группе $(Z/n\mathbb{Z})^*$, где n - произведение двух простых чисел. Криптосистема Benalo гомоморфна относительно операции сложения.

Стойкость криптосистемы Benalo основана на труднорешаемой задаче о вычетах высокой степени. Зная размер блока r, модуль n и шифротекст $E(M)$, где разложение на множители числа n неизвестно, определить открытый текст вычислительно сложно.

Генерация открытого и закрытого ключей

Генерация открытого и закрытого ключей шифрования производится следующим образом.

Для начала, выбирается блок размера r и два больших различных простых числа p и q, удовлетворяющие следующим условиям: r и $(p - 1) / r$ – взаимно простые числа; r и $q - 1$ – взаимно простые числа. После вычисляется $n = p \times q$ и $\varphi = (p - 1)(q - 1)$. Затем выбирается такое $y \in \mathbb{Z}_n^*$, что $y^{\frac{\varphi}{r}} \neq 1 \, mod \, n$. Замечание: если r составное, то вышеуказанные условия не являются достаточными для обеспечения правильной расшифровки, то есть для того, чтобы всегда выполнялось $D\big(E(m)\big) = m$. Чтобы избежать подобного, предлагается выполнять следующую проверку. Пусть $r = p_1 p_2 p_3 \dots p_k$. Тогда y выбирается таком образом, чтобы для каждого p_i выполнялось $y^{\frac{\varphi}{r}} \neq 1 \, mod \, n$. Наконец, $x = y^{\frac{\varphi}{r}} \, mod \, n$.

Таким образом, открытым ключом являются (y, r, n), а закрытым ключом $- (\varphi, x)$.

Шифрование и дешифрование сообщения

Пусть дано сообщение m, тогда шифрование этого сообщения $m \in \mathbb{Z}_{\varphi}^{*}$ выполняется следующим образом. Выбирается такое произвольное число $u \in \mathbb{Z}_{n}^{*}$, что $E_r(m) = y^m u^r \, mod \, n$. В результате преобразований получается зашифрованное сообщение.

Расшифрование зашифрованного сообщения происходит по следующему алгоритму. Для начала заметим, что для любых $m \in \mathbb{Z}_r$ и $u \in \mathbb{Z}_r$ выполняется $a = (c)^{\frac{\varphi}{r}} \equiv (y^m u^r)^{\frac{\varphi}{r}} \equiv (y^m)^{\frac{\varphi}{r}} (u^r)^{\frac{\varphi}{r}} \equiv \left(y^{\frac{\varphi}{r}}\right)^m (u)^{\varphi} \equiv (x)^m (u)^0 \equiv x^m \, mod \, n.$ Таким образом, чтобы вычислить m, зная a, проводится операция дискретного логарифмирования из a по основанию x. Если число r небольшое, возможно нахождение m через исчерпывающий перебор, то есть проверкой выполнения равенства $x^i \equiv a \, mod \, n$ для всех $0 \ldots (r-1)$. При больших значениях r m можно найти с помощью алгоритма Гельфонда-Шенкса (алгоритм больших и малых шагов), получив временную сложность расшифрования $O(\sqrt{r})$.

Расшифрование шифротекста $c \in \mathbb{Z}_{n}^{*}$ происходит следующим образом. Во-первых, вычисляется $a = c^{\frac{\varphi}{r}} \, mod \, n.$ Во-вторых, подбирается $m = log_x(a)$, то есть такое m, что $x^m \equiv a \, mod \, n.$

Алгоритм RC6

Описание

RC6 — симметричный блочный криптографический алгоритм, производный от алгоритма RC5. Был создан Роном Ривестом, Мэттом Робшау и Рэем Сиднеем для удовлетворения требований конкурса Advanced Encryption Standard (AES). Является собственническим (проприетарным) алгоритмом, и запатентован RSA Security.

Вариант шифра RC6, заявленный на конкурс AES, поддерживает блоки длиной 128 бит и ключи длиной 128, 192 и 256 бит, но сам алгоритм может быть сконфигурирован для поддержки более широкого диапазона длин как блоков, так и ключей (от 0 до 2040 бит). RC6 очень похож на RC5 по своей структуре и также довольно прост в реализации.

Является финалистом AES, однако одна из примитивных операций — операция умножения, медленно выполняемая на некотором оборудовании, затрудняет реализацию шифра на ряде аппаратных платформ и, что оказалось сюрпризом для авторов, на системах с архитектурой Intel IA-64 также реализована довольно плохо. В данном случае алгоритм теряет одно из своих ключевых преимуществ — высокую скорость выполнения, что стало причиной для критики и одной из преград для избрания в качестве нового стандарта.

Так же, как и RC5, RC6 — полностью параметризированная семья алгоритмов шифрования. Для спецификации алгоритма с конкретными параметрами, принято обозначение RC6-w/r/b, где w — длина машинного слова в битах, r — число раундов, b — длина ключа в битах. Возможные значения 0...255 бит.

Для того чтобы соответствовать требованиям AES, блочный шифр должен обращаться с 128-битовыми блоками. Так как RC5 — исключительно быстрый блочный шифр, расширение его, чтобы работать с 128-битовыми блоками, привело бы к использованию двух 64-битовых рабочих регистров. Но архитектура и языки программирования ещё не поддерживают 64-битные операции, поэтому пришлось изменить проект так, чтобы использовать четыре 32-битных регистров вместо двух 64-битных.

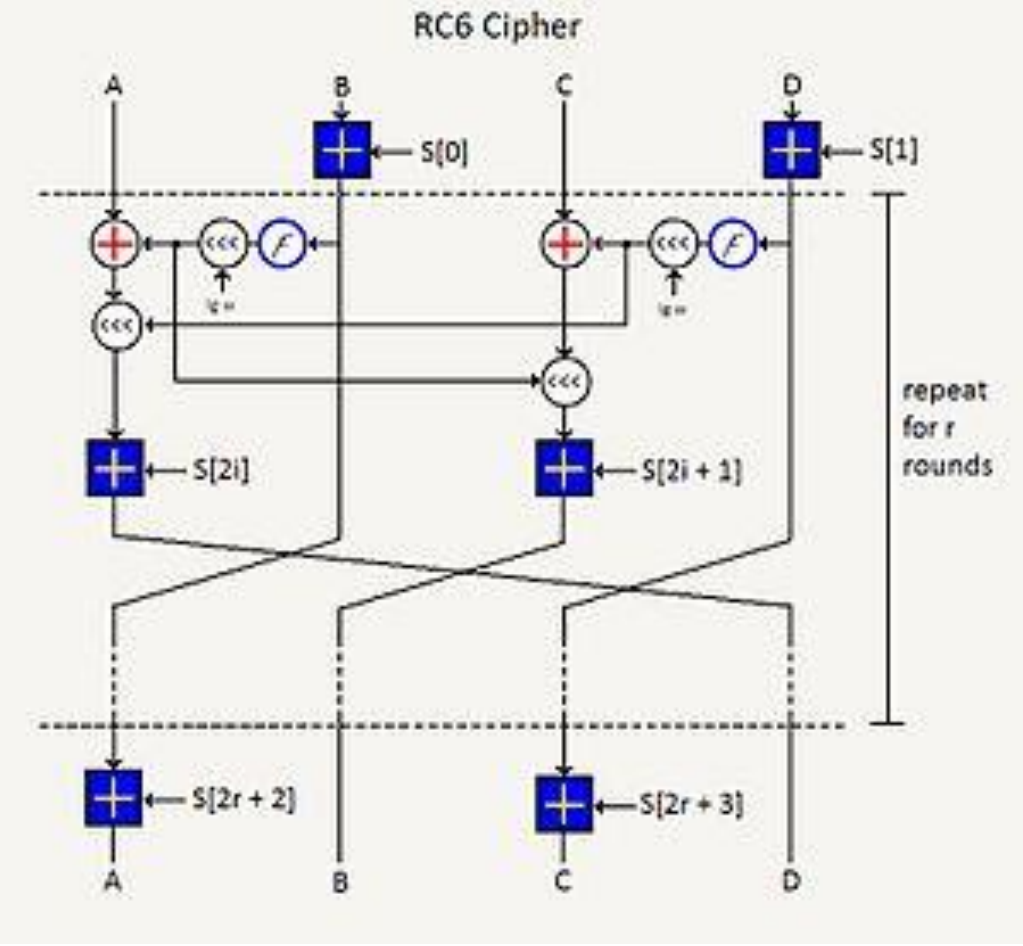

Рисунок 1. Сеть Фейстеля алгоритма RC6

Алгоритм генерации ключа

Так же, как и в RC5, в RC6 генерируются две псевдослучайные величины, используя две математические константы: экспонента (*e*) и золотое сечение (*f*).

$$Q_\omega \leftarrow Odd((f-1)*2^\omega)$$

$$P_\omega \leftarrow Odd(e-2)*2^\omega),$$

где Odd() — это округление до ближайшего нечетного целого. При *w* = 32 бита (в шестнадцатеричном виде):

$$Q_{32} = 9E3779B9_{16} = 2654435769_{10}$$

$$P_{32} = B7E15163_{16} = 3084996963_{10}.$$

Процедура расширения ключа для RC6-w/r/b представлена на рисунке 2. Входными параметрами являются *b*-байтный ключ, заданный пользователем, предварительно преобразованный в массив из *c* слов *L[0, …, c - 1]*, и *r* – количество раундов, а выходным параметром является *w*-байтная таблица ключей *S[0, …, 2r + 3]*.

```
S[0]=Pw
for i=1 to 2r+3 do
        S[i]=S[i-1]+Qw

A=B=i=j=0

v=3*max{c,2r+4}
for s=1 to v do
    {
        A=S[i]=(S[i]+A+B)<<<3
        B=L[j]=(L[j]+A+B)<<<(A+B)
        i=(i+1) mod (2r+4)
        j=(j+1) mod c
    }
```

Рисунок 2. Процедура расширения ключа для RC6

Алгоритм шифрования

RC6 работает с четырьмя w-битными регистрами A, B, C и D, которые содержат входной исходный текст и выходной шифрованный текст в конце шифрования. Входными параметрами для процедуры шифрования являются количество раундов *r* и *w*-разрядные ключи для каждого раунда *S[0, ... , 2r + 3]*. В результате выполнения алгоритма, приведенного на рисунке 3, получается зашифрованное сообщение в виде *w*-битных регистров A, B, C, D.

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
    t = (B(2B + 1)) <<< lg w
    u = (D(2D + 1)) <<< lg w
    A = ((A ⊕ t) <<< u) + S[2i]
    C = ((C ⊕ u) <<< t) + S[2i + 1]
            (A, B, C, D)  =  (B, C, D, A)

}
A = A + S[2r + 2]
C = C + S[2r + 3]
```

Рисунок 3. Процедура шифрования

Алгоритм дешифрования

Входными параметрами для процедуры дешифрования являются зашифрованный текст, предварительно сохраненный в A, B, C, D, количество раундов r и w-разрядные ключи для каждого раунда *S[0, … , 2r + 3]*. В результате выполнения алгоритма получается исходный текст, сохраненный в A, B, C, D. Алгоритм дешифрования приведен на рисунке 4.

```
C = C - S[2r + 3]
A = A - S[2r + 2]

for i = r downto 1 do
{
   (A, B, C, D) = (D, A, B, C)
    u = (D(2D + 1)) <<< lg w
    t = (B(2B + 1)) <<< lg w
    C = ((C - S[2i + 1]) >>> t) ⊕ u
    A = ((A - S[2i]) >>> u) ⊕ t
}
D = D - S[1]
B = B - S[0]
```

Рисунок 4. Процедура расшифровки

Вероятностные тесты простоты

Тест Ферма

Если n – простое число, то оно удовлетворяет условию $a^{n-1} \equiv 1\ (mod\ n)$ для любого a, которое не делится на n. Выполнение этого сравнения является необходимым, но не достаточным признаком простоты числа. То есть, если найдётся хотя бы одно a, для которого $a^{n-1} \neq 1\ (mod\ n)$, то число n – составное. В противном случае ничего сказать нельзя, хотя шансы на то, что число является простым, увеличиваются. Если для составного числа n выполняется сравнение $a^{n-1} \equiv 1\ (mod\ n)$, то число n называют псевдопростым по основанию a.

При проверке числа на простоту тестом Ферма выбирают несколько чисел a, и чем больше количество a, для которых $a^{n-1} \equiv 1\ (mod\ n)$, тем больше шансы, что число n – простое. Однако существуют составные числа, для которых сравнение $a^{n-1} \equiv 1\ (mod\ n)$ выполняется для всех a, взаимно простых с n – это числа Кармайкла. Всего чисел Кармайкла бесконечное количество, наименьшее равно 561. Тем не менее, тест Ферма является довольно эффективным для проверки числа на простоту.

Тест Соловея–Штрассена

Тест Соловея-Штрассена опирается на малую теорему Ферма и свойства символа Якоби: если n – нечетное составное число, то количество целых чисел a, взаимнопростых с n и меньших n, удовлетворяющих сравнению $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n)$, не превосходит $\frac{n}{2}$. Составные числа n удовлетворяющие этому сравнению называются псевдопростыми Эйлера-Якоби по основанию a.

Алгоритм Соловея-Штрассена параметризуется количеством раундов k. В каждом раунде случайным образом выбирается число $a < n$. Если НОД(a, n) > 1, то выносится решение, что n составное, в противном случае проверяется справедливость равенства $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n)$. Если оно не выполняется, то выносится решение, что n — составное. Если это сравнение выполняется, то a является свидетелем простоты числа n. Далее выбирается другое случайное a, и процедура повторяется. После нахождения k свидетелей простоты в k раундах выносится заключение, что n является простым числом с вероятностью $1 - 2^{-k}$.

Тест Миллера-Рабина

Как и тесты Ферма и Соловея-Штрассена, тест Миллера-Рабина опирается на проверку ряда равенств, которые выполняются для простых чисел. Если хотя бы одно такое равенство не выполняется, это доказывает, что число составное.

Для теста Миллера-Рабина используется следующее утверждение. Пусть n – простое число, и $n - 1 = \ 2^s d$, где d нечетно. Тогда для любого a из $\mathbb{Z}_n$ выполняется хотя бы одно из условий:

1. $a^d \equiv 1\ (mod\ n)$
2. Существует целое число $r < s$ такое, что $a^{2^r d} \equiv -1\ (mod\ n)$.

Практическая часть

Описание работы сервера

Сервер – консольное приложение, написанное на языке программирования C#. Сервер работает на локальном IP-адресе с определенным портом, который можно изменять в зависимости от того, занят ли этот порт системой.

При запуске серверного приложения создается серверный сокет, который в бесконечном цикле (до тех пор, пока администратор сервера не завершит серверное приложение) ожидает подключение клиентов. Если происходит подключение нового пользователя, сервер принимает подключение, заносит пользователя в список активных пользователей и отправляет этому пользователю список активных участников чата для отображения активных пользователей чата в интерфейсе.

Для обработки действие клиентов на сервере создан класс Connection, который отвечает за создание нового подключения и обработки действий, связанных с клиентом. При создании нового подключения на сервере запускается асинхронный бесконечный цикл обработки действий клиента, который подключился к серверу. Цикл обрабатывает такие операции, как: подключение нового клиента, прием и отправление сообщений и файлов, отключение клиента от сервера.

Как только пользователь подключился к серверу, он может участвовать в операциях отправления и получения информации. Для упрощения обмена информацией между сервером и клиентами были созданы классы PacketBuilder и PacketReader. Класс PacketBuiler служит для формирования сообщения и преобразования получившегося сообщения в массив байтов. Сообщение имеет следующий вид: на первом месте стоит код операции, на втором месте – какая-либо полезная информация или сообщение. Полученный массив байтов может быть отправлен либо серверу, либо

клиенту. Если сообщение доставлено по месту назначения, оно обрабатывается и в зависимости от кода операции выполняет действие. Для обработки полученных сообщений используется класс PacketReader, который считывает код операции и выполняет какое-либо действие (например, считывание имени пользователя при регистрации на сервере или приём сообщения от сервера).

Так как информация, передаваемая между сервером и клиентом, нуждается в защите, для шифрования и дешифрования информации используется алгоритм RC6. Как только сервер отправил сообщение или файл, оно шифруется ключом симметричного шифрования, которым владеют только участники текущей сессии чата, после чего отправляется на сервер. Сервер принимает зашифрованное сообщение и работает только с зашифрованными данными, что позволяет предотвратить кражу информации во время отправки данных и работы с ними. В свою очередь, клиент при получении зашифрованной информации может расшифровать ее и считать. Если ключ симметричного шифрования неверный, то получить данные текущей сессии чата будет невозможно.

Описание работы клиента

Клиентское приложение представляет собой чат для общения и передачи файлов между собеседниками. Для разработки пользовательского интерфейса был выбран язык C# и система создания графических приложений WPF. Стартовый интерфейс приведен на рисунке 5.

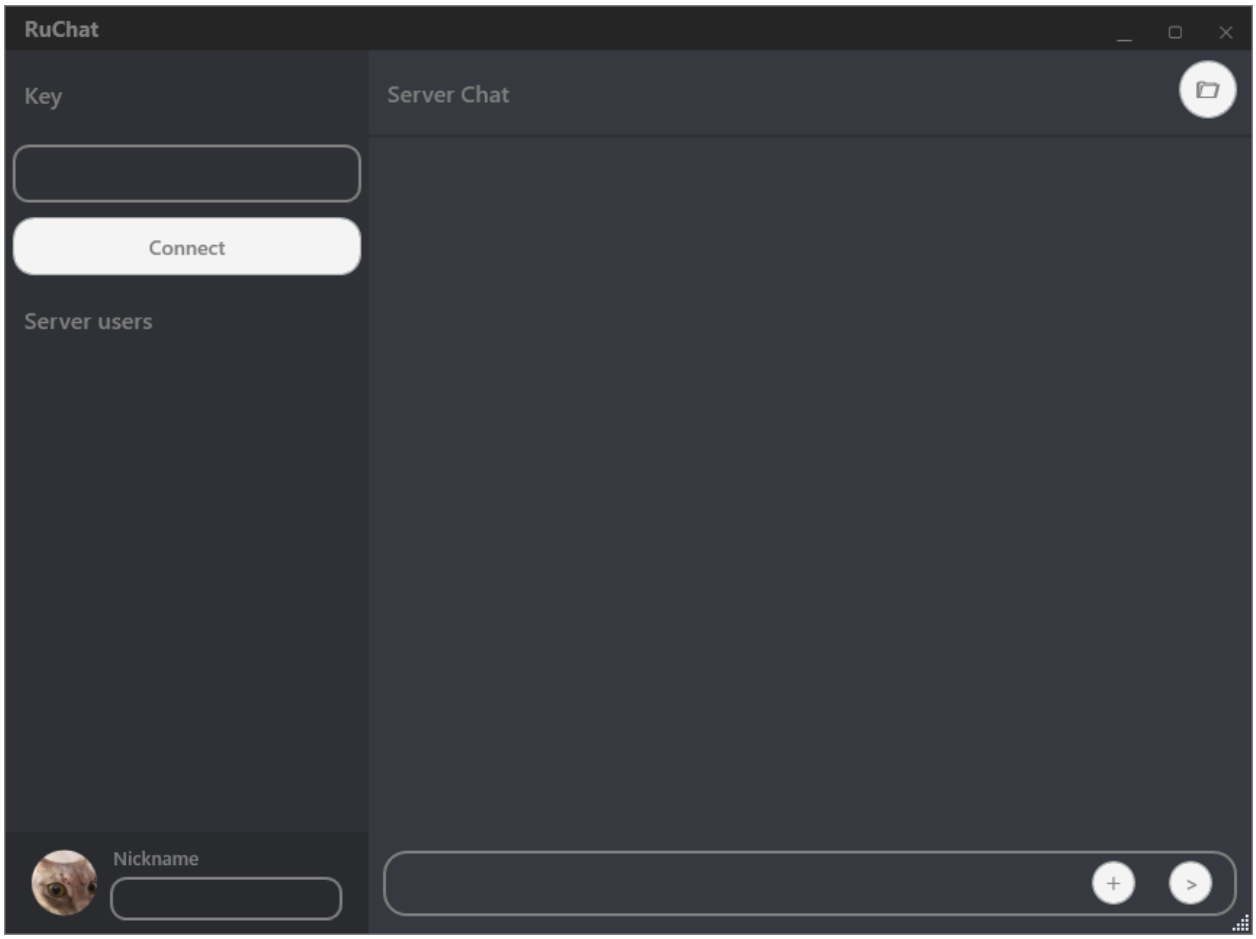

Рисунок 5. Стартовый интерфейс программы

Для авторизации в чате пользователю необходимо ввести свое имя (поле “Nickname”) и ключ симметричного алгоритма для шифрования и дешифрования переписки. Как только пользователь ввел свое имя и ключ шифрования, он может подключиться к чату с помощью кнопки “Connect”. Для создания ключа шифрования была разработана небольшая программа, генерирующая 128-битовый ключ алгоритма RC6. Для удобства пользователей сгенерированный ключ имеет числовое представление, а не

байтовое (в языке программирования C# — это тип данных BigInteger). Интерфейс программы для генерации ключей симметричного алгоритма продемонстрирован на рисунке 6.

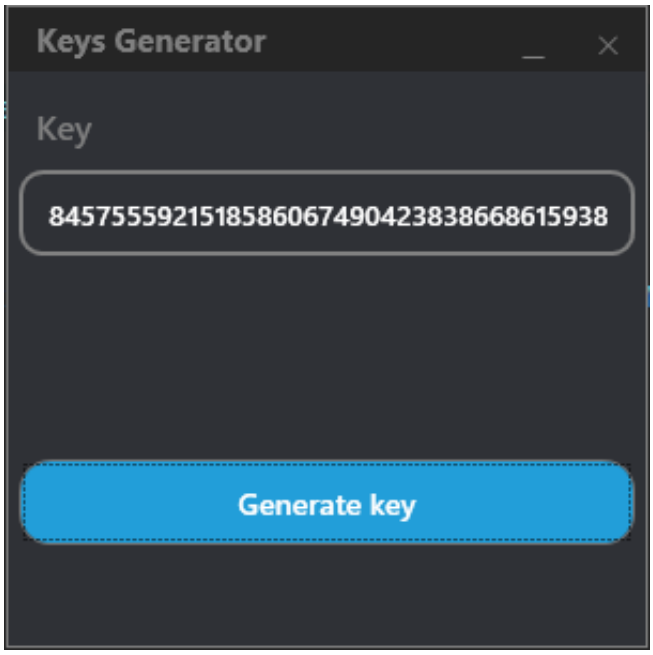

Рисунок 6. Приложение для генерации ключей шифрования

В случае, если клиенту не удалось подключиться к серверу, приложение уведомляет пользователя об ошибке. Если же подключение успешно совершено, пользователю становится доступен список активных пользователей сервера, а также возможность отправлять сообщения в чат, отправлять файлы на сервер и скачивать файлы с сервера. Пример успешной авторизации приведен на рисунке 7.

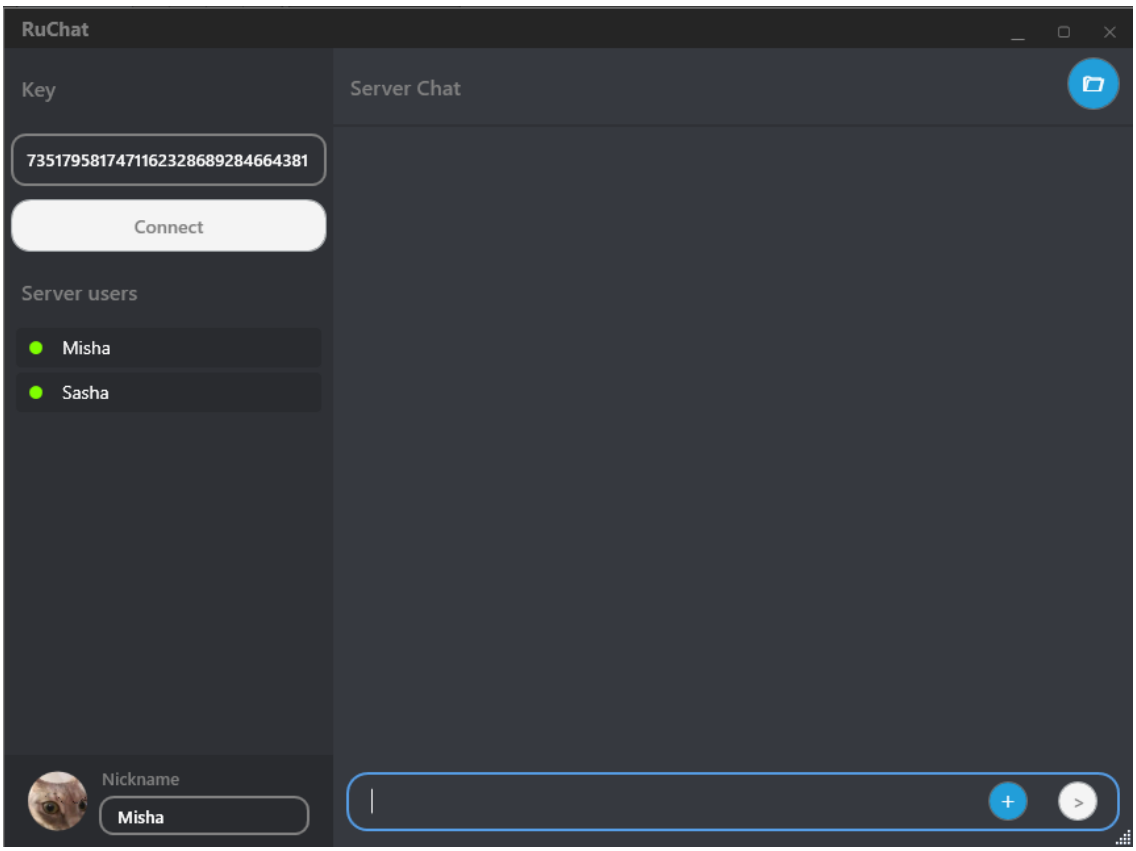

Рисунок 7. Успешная авторизация в чате

Пользователи могут отправлять сообщения в чат и общаться друг с другом. Каждое сообщение шифруется и дешифруется сгенерированным ключом шифрования, что обеспечивает приватность и безопасность общения. Пример общения приведен на рисунке 8.

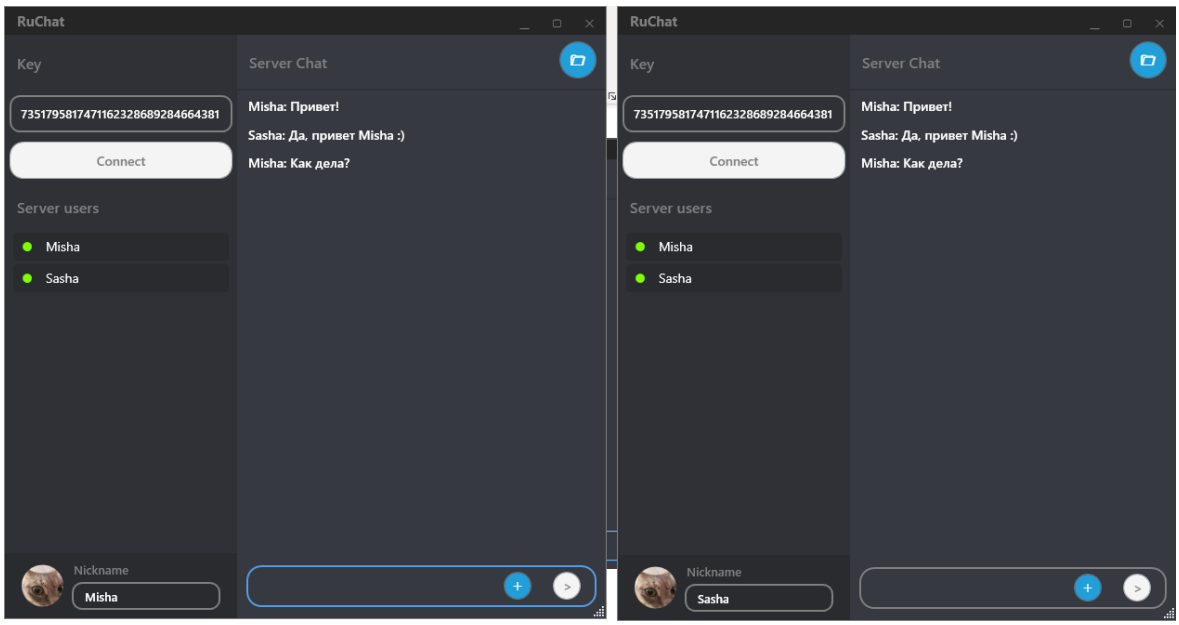

Рисунок 8. Общение пользователей в чате

Кроме обмена сообщений, чат предлагает пользователям и обмен файлами. Все файлы, которыми пользователи обмениваются в текущей сессии чата, хранятся на сервере. Для того, чтобы поделиться файлом с другими пользователями (другими словами, загрузить на сервер с возможностью последующего скачивания), необходимо нажать на кнопку “+” и выбрать файл, после чего нажать кнопку “Ок”. Файлы хранятся в зашифрованном виде на сервере только во время текущей сессии. В случае потери ключа шифрования или завершения работы сервера доступ к данным будет утерян.

Для скачивания файла с сервера необходимо нажать на кнопку со значком папки. Загруженные файлы можно найти в папке UserFiles основной директории клиентского приложения. Пример файлов, хранящихся на сервере и доступных для скачивания, приведен на рисунке 9.

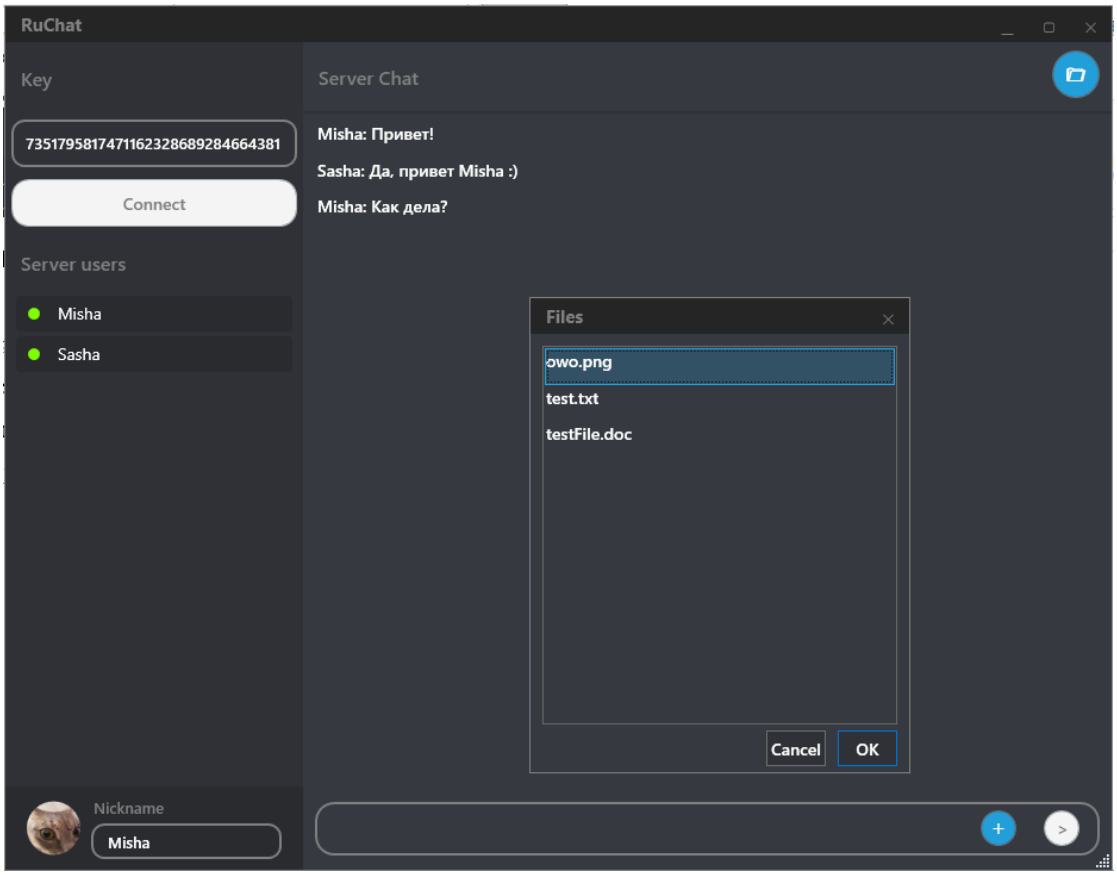

Рисунок 9. Файлы на сервере

Для выхода из чата достаточно нажать кнопку “X” в правом верхнем углу.

Вывод

В процессе работы были изучены и реализованы на языке программирования C# алгоритмы симметричного шифрования RC6 и ассиметричного шифрования Benalo. Для демонстрации работы этих алгоритмов было разработано клиент-серверное приложение “Чат”, состоящее из двух отдельных приложений: сервера и клиента чата.

Сервер, написанный на языке программирования C#, служит для асинхронной обработки действий клиентов и обменом данных, что обеспечивает быстродействие. Клиентское приложение на WPF позволяет пользователю генерировать ключи симметричного алгоритма шифрования и общаться между с другими клиентами, производить обмен данными и файлами.

Все данные, которыми обмениваются пользователи, защищены алгоритмом шифрования, что обеспечивает надежность программы, а также сохранность данных и приватность пользователей.

Список литературы

1. Н. Смарт Криптография Москва: Техносфера, 2005. — 528 с. ISBN 5-94836-043-1
2. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. М.: Триумф, 2003. 806 с.
3. Режим шифрования // Википедия. [2021]. Дата обновления: 14.10.2021. URL: https://ru.wikipedia.org/?curid=387918&oldid=117218082 (дата обращения: 15.05.2022).
4. RC6 // Википедия. [2022]. Дата обновления: 15.03.2022. URL: https://ru.wikipedia.org/?curid=923453&oldid=120673359 (дата обращения: 25.05.2022).
5. Криптосистема Бенало // Википедия. [2021]. Дата обновления: 09.12.2021.URL: https://ru.wikipedia.org/?curid=7243317&oldid=118506161 (дата обращения: 27.05.2022).
6. TcpListener Класс [Электронный ресурс]: документация по C#. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.sockets.tcplistener?view=net-6.0
7. TcpClient Класс [Электронный ресурс]: документация по C#. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.sockets.tcpclient?view=net-6.0
8. Тест Ферма // Википедия. [2022]. Дата обновления: 11.05.2022. URL: https://ru.wikipedia.org/?curid=3661995&oldid=122231269 (дата обращения: 18.05.2022).
9. Тест Соловея — Штрассена // Википедия. [2020]. Дата обновления: 02.01.2020. URL: https://ru.wikipedia.org/?curid=2103098&oldid=104322137 (дата обращения: 18.05.2022).

10. Тест Миллера — Рабина // Википедия. [2022]. Дата обновления: 23.04.2022. URL: https://ru.wikipedia.org/?curid=11550&oldid=121637892 (дата обращения: 18.05.2022)

Приложение

UserModel.cs

```
namespace Messanger.ClientWPF.MVVM.Model;

public class UserModel
{
    public string Username { get; init; }
    public string Uid { get; init; }
}
```

MainWindow.xaml

```
<Window x:Class="Messanger.ClientWPF.MVVM.View.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:viewModel="clr-
namespace:Messanger.ClientWPF.MVVM.ViewModel"
        xmlns:local="clr-namespace:Messanger.ClientWPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="640" Width="860"
        Icon="pack://application:,,,/Resources/Images/Icon.ico"
        Background="#2F3136"
        WindowStyle="None"
        AllowsTransparency="True"
        ResizeMode="CanResizeWithGrip"
        BorderThickness="1"
        BorderBrush="Gray">

    <Window.DataContext>
        <viewModel:MainWindowViewModel />
    </Window.DataContext>

    <Window.InputBindings>
        <KeyBinding Command="{Binding SendMessageCommand}"
Key="Enter"/>
    </Window.InputBindings>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition />
```

```
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="250" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>

<!-- Window Border-->
<Border Grid.Row="0"
        Grid.Column="0"
        Grid.ColumnSpan="2"
        Background="#252525"
        MouseDown="WindowBorderOnMouseDown">
    <Grid HorizontalAlignment="Stretch">
        <Label Content="RuChat"
               Foreground="Gray"
               FontWeight="Bold"
               FontSize="15"
               Margin="8 0 0 0"/>
        <StackPanel HorizontalAlignment="Right"
                    Orientation="Horizontal">
            <Button Width="35"
                    Height="35"
                    Content="___"
                    Foreground="Gray"
                    HorizontalAlignment="Right"
                    Background="Transparent"
                    BorderThickness="0"
                    Click="MinimizeButtonOnClick"/>
            <Button Width="35"
                    Height="35"
                    Content="▢"
                    Foreground="Gray"
                    HorizontalAlignment="Right"
                    Background="Transparent"
                    BorderThickness="0"
                    Click="MaximizeButtonOnClick"/>
            <Button Width="35"
                    Height="35"
                    Content="✕"
                    Foreground="Gray"
                    HorizontalAlignment="Right"
                    Background="Transparent"
                    BorderThickness="0"
                    Click="CloseButtonOnClick"/>
        </StackPanel>
    </Grid>
</Border>
```

```
        <!-- Left menu field with users information -->
        <Grid ShowGridLines="False"
              Grid.Row="1"
              Grid.Column="0"
              Background="#2F3136">
            <Grid.RowDefinitions>
                <RowDefinition Height="60" />
                <RowDefinition Height="50" />
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition />
                <RowDefinition Height="70" />
            </Grid.RowDefinitions>

            <Label Grid.Row="0"
                   Content="Key"
                   VerticalAlignment="Center"
                   FontWeight="Medium"
                   Foreground="Gray"
                   FontSize="16"
                   Margin="8 0 0 0"/>
            <TextBox Grid.Row="1"
                     Text="{Binding Key,
UpdateSourceTrigger=PropertyChanged}"
                     Foreground="White"
                     FontSize="13"
                     Height="40"
                     FontWeight="SemiBold"
                     Background="Transparent"
                     BorderThickness="2"
                     BorderBrush="Gray"
                     Margin="5 5 5 5"
                     Padding="10 0 10 0"
                     CaretBrush="White"
                     VerticalContentAlignment="Center">
                <TextBox.Resources>
                    <Style TargetType="{x:Type Border}">
                        <Setter Property="CornerRadius" Value="10" />
                    </Style>
                </TextBox.Resources>
            </TextBox>

            <Button Grid.Row="2"
                    Height="40"
                    DockPanel.Dock="Top"
                    Content="Connect"
```

```
                    Style="{StaticResource ButtonStyle}"
                    Margin="5 5 5 5"
                    Command="{Binding ConnectToServerCommand}">
                <Button.Resources>
                    <Style TargetType="{x:Type Border}">
                        <Setter Property="CornerRadius" Value="14" />
                    </Style>
                </Button.Resources>
            </Button>

            <Label Grid.Row="3"
                   Content="Server users"
                   VerticalAlignment="Center"
                   FontWeight="Medium"
                   Foreground="Gray"
                   FontSize="16"
                   Margin="8 0 0 0"/>

            <ListView Grid.Row="4"
                      Background="Transparent"
                      BorderThickness="0"
                      ItemsSource="{Binding Users}"
                      ItemContainerStyle="{StaticResource
ServerUserStyle}">

                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Username}"/>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

            <DockPanel Grid.Row="5">
                <StackPanel Orientation="Horizontal"
                            Background="#292B2F"
                            DockPanel.Dock="Bottom">
                    <Ellipse Width="45" Height="45"
                             Margin="18 0 0 0">
                        <Ellipse.Fill>
                            <ImageBrush
ImageSource="../../Resources/Images/Avatar.jpg" />
                        </Ellipse.Fill>
                    </Ellipse>

                    <StackPanel VerticalAlignment="Center"
                                Margin="6 0 0 5">
                        <Label Content="Nickname"
```

```
                                Foreground="Gray"
                                FontSize="13"
                                FontWeight="SemiBold"/>
                        <TextBox Text="{Binding Username,
UpdateSourceTrigger=PropertyChanged}"
                                 Foreground="White"
                                 Width="160"
                                 Height="30"
                                 FontSize="13"
                                 FontWeight="SemiBold"
                                 Background="Transparent"
                                 BorderThickness="2"
                                 BorderBrush="Gray"
                                 Margin="3 0 0 0"
                                 Padding="10 0 10 0"
                                 CaretBrush="White"
                                 VerticalContentAlignment="Center">
                            <TextBox.Resources>
                                <Style TargetType="{x:Type Border}">
                                    <Setter Property="CornerRadius"
Value="10" />
                                </Style>
                            </TextBox.Resources>
                        </TextBox>
                    </StackPanel>
                </StackPanel>
            </DockPanel>
        </Grid>

        <!-- Right field for messages -->
        <Grid Grid.Row="1" Grid.Column="1">
            <Grid.RowDefinitions>
                <RowDefinition Height="60" />
                <RowDefinition />
                <RowDefinition Height="60" />
            </Grid.RowDefinitions>

            <Grid Grid.Row="0" ShowGridLines="False"
                  Background="#36393F"
                  Margin="0 0 0 2">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition Width="60"/>
                </Grid.ColumnDefinitions>

                <Label Grid.Column="0"
                       Content="Server Chat"
```

```
                        VerticalAlignment="Center"
                        FontWeight="Medium"
                        Foreground="Gray"
                        FontSize="16"
                        Margin="8 0 0 0"/>

                <Button Grid.Column="1"
                        Height="40"
                        Width="40"
                        HorizontalAlignment="Right"
                        Content="📂"
                        Style="{StaticResource ButtonStyle}"
                        Margin=" 5 5 10 10"
                        Command="{Binding OpenServerFilesCommand}">
                    <Button.Resources>
                        <Style TargetType="{x:Type Border}">
                            <Setter Property="CornerRadius" Value="25"
/>
                            <Setter Property="Background"
Value="Aqua"></Setter>
                        </Style>
                    </Button.Resources>
                </Button>
            </Grid>

            <ListView Grid.Row="1"
                      Background="#36393F"

                      Style="{StaticResource ChatStyle}"
                      ItemsSource="{Binding Messages}">
                <ListView.Resources>
                    <Style TargetType="{x:Type ListViewItem}">
                        <Setter Property="Padding" Value="10 5 10 5
"/>
                        <Setter Property="Focusable" Value="False"/>
                    </Style>
                </ListView.Resources>
            </ListView>

            <Grid Grid.Row="2"
                  Background="#36393F">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition Width="50"/>
                    <ColumnDefinition Width="70" />
                </Grid.ColumnDefinitions>
```

```
                <TextBox Grid.Row="0" Grid.Column="0"
                         Grid.ColumnSpan="3"
                         Style="{StaticResource MessageBoxStyle}"
                         Text="{Binding Message,
UpdateSourceTrigger=PropertyChanged}">
                    <TextBox.Resources>
                        <Style TargetType="{x:Type Border}">
                            <Setter Property="CornerRadius" Value="15"
/>
                        </Style>
                    </TextBox.Resources>
                </TextBox>

                <Button Grid.Column="1"
                        Width="30"
                        Height="30"
                        Content="＋"
                        VerticalContentAlignment="Center"
                        Margin=" 5 5 5 15"
                        Style="{StaticResource ButtonStyle}"
                        Command="{Binding OpenClientFilesCommand}">
                    <Button.Resources>
                        <Style TargetType="{x:Type Border}">
                            <Setter Property="CornerRadius" Value="25"
/>
                        </Style>
                    </Button.Resources>
                </Button>

                <Button Grid.Column="2"
                        Width="30"
                        Height="30"
                        DockPanel.Dock="Top"
                        Content=">"
                        Style="{StaticResource ButtonStyle}"
                        Margin=" 5 5 20 15"
                        Padding="3 0 0 0"
                        Command="{Binding SendMessageCommand}">
                    <Button.Resources>
                        <Style TargetType="{x:Type Border}">
                            <Setter Property="CornerRadius" Value="25"
/>
                        </Style>

                    </Button.Resources>

                </Button>
```

```
            </Grid>
        </Grid>

    </Grid>
</Window>
```

MainWindowViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.IO;
using System.IO.Enumeration;
using System.Linq;
using System.Text;
using System.Windows;
using Messanger.ClientWPF.MVVM.Model;
using Messanger.ClientWPF.Net;
using Messanger.ClientWPF.Themes;
using Microsoft.Win32;
using MVVM.Core.Command;
using MVVM.Core.ViewModel;

namespace Messanger.ClientWPF.MVVM.ViewModel
{
    public sealed class MainWindowViewModel : ViewModelBase
    {
        private const string UserFiles =
"\\Messanger.ClientWPF\\UserFiles\\";
        private readonly Client _client;

        public string Username { get; set; }
        public string Message { get; set; }
        public string Key { get; set; }

        public ObservableCollection<UserModel> Users { get; }
        public ObservableCollection<string> Messages { get; }

        public RelayCommand ConnectToServerCommand { get; }
        public RelayCommand SendMessageCommand { get; set; }
        public RelayCommand OpenServerFilesCommand { get; }
        public RelayCommand OpenClientFilesCommand { get; set; }

        public MainWindowViewModel()
        {
            Username = string.Empty;
            Message = string.Empty;
            Key = string.Empty;
            Users = new ObservableCollection<UserModel>();
```

```
            Messages = new ObservableCollection<string>();

            _client = new Client();
            _client.ConnectedEvent += UserConnected;
            _client.MessageReceivedEvent += MessageReceived;
            _client.UserDisconnectedEvent += UserDisconnected;
            _client.FileReceivedEvent += FileReceived;

            ConnectToServerCommand = new RelayCommand(
                _ =>
                {
                    try
                    {
                        _client.ConnectToServer(Username, Key);
                    }
                    catch (Exception)
                    {
                        MessageBox.Show($"Failed to connect to the
server");
                    }
                },
                _ => !string.IsNullOrEmpty(Username) &&
!string.IsNullOrEmpty(Key) && !_client.IsConnectedToServer());
            SendMessageCommand = new RelayCommand(
                _ =>
                {   try
                    {
                        _client.SendMessageToServer(Message);
                        Message = "";
                        RaisePropertyChanged(nameof(Message));
                    }
                    catch (Exception)
                    {
                        MessageBox.Show($"Failed to send message");
                    }
                },
                _ => !string.IsNullOrEmpty(Message) &&
_client.IsConnectedToServer());
            OpenServerFilesCommand = new RelayCommand(
                _ => OpenServerFileDialog(),
                _ => _client.IsConnectedToServer());
            OpenClientFilesCommand = new RelayCommand(
                _ => OpenClientFileDialog(),
                _ => _client.IsConnectedToServer());
        }

        private void FileReceived()
```

```
        {
            var filename =
Encoding.Default.GetString(_client.PacketReader.ReadMessage());
            var message =
_client.Algorithm.Decrypt(_client.PacketReader.ReadMessage());
            var directory = new
DirectoryInfo(Directory.GetCurrentDirectory());
            while (directory != null &&
!directory.GetFiles("*.sln").Any())
            {
                directory = directory.Parent;
            }

            var fullPath = directory + UserFiles + filename;
            File.WriteAllBytesAsync(fullPath, message);
;        }

        private void UserConnected()
        {
            var user = new UserModel
            {
                Username =
Encoding.Default.GetString(_client.PacketReader.ReadMessage()),
                Uid =
Encoding.Default.GetString(_client.PacketReader.ReadMessage())
            };

            if (Users.All(x => x.Uid != user.Uid))
            {
                Application.Current.Dispatcher.Invoke(() =>
Users.Add(user));
            }
        }

        private void MessageReceived()
        {
            var username =
Encoding.Default.GetString(_client.PacketReader.ReadMessage());
            var separator =
Encoding.Default.GetString(_client.PacketReader.ReadMessage());
            var message =
_client.Algorithm.Decrypt(_client.PacketReader.ReadMessage());
            var buildMessage = username + separator +
Encoding.Default.GetString(message);
            Application.Current.Dispatcher.Invoke(() =>
Messages.Add(buildMessage));
        }
```

```
        private void UserDisconnected()
        {
            var uid =
Encoding.Default.GetString(_client.PacketReader.ReadMessage());
            var user = Users.FirstOrDefault(x => x.Uid == uid);
            Application.Current.Dispatcher.Invoke(() => user != null
&& Users.Remove(user));
        }

        private void OpenClientFileDialog()
        {
            var openFileDialog = new OpenFileDialog();
            if (openFileDialog.ShowDialog() == true)
            {
                var fullPath = openFileDialog.FileName;
                _client.SendFileToServer(fullPath);
            }
        }

        private void OpenServerFileDialog()
        {
            var openFileDialog = new ServerFilesDialog();
            if (openFileDialog.ShowDialog() == true)
            {
                var fullPath = openFileDialog.FileName;
                var index = fullPath.LastIndexOf('\\');
                var filename = fullPath.Substring(index + 1,
fullPath.Length - index - 1);
                _client.GetFileFromServer(filename);
            }
        }
    }
}
```

Client.cs

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
using Messanger.Crypto.RC6.Classes;
using Messanger.Server.Net.IO;

namespace Messanger.ClientWPF.Net
{
```

```
public sealed class Client
{
    private readonly EncryptionMode _mode;
    private readonly byte[] _initVector;
    private readonly string _param;
    internal CipherContext Algorithm;

    private readonly TcpClient _client;
    public PacketReader PacketReader;

    public event Action ConnectedEvent;
    public event Action MessageReceivedEvent;
    public event Action UserDisconnectedEvent;
    public event Action FileReceivedEvent;

    public Client()
    {
        _mode = EncryptionMode.ECB;
        _initVector = GenerateInitVector();
        _param = "";
        _client = new TcpClient();
    }

    private byte[] GenerateInitVector()
    {
        var random = new Random();
        var initVector = new byte[16];
        for (var i = 0; i < 16; ++i)
        {
            initVector[i] = (byte)random.Next(0, 255);
        }
        return initVector;
    }

    public void ConnectToServer(string username, string key)
    {
        if (_client.Connected)
            return;

        var byteKey = BigInteger.Parse(key).ToByteArray();
        if (byteKey.Length * 8 != 128)
            throw new ArgumentException("Incorrect session key");

        _client.ConnectAsync("127.0.0.1", 7891);
        PacketReader = new PacketReader(_client.GetStream());

        if (!string.IsNullOrEmpty(username))
```

```
            {
                var connectPacket = new PacketBuilder();
                connectPacket.WriteOpCode(0);

connectPacket.WriteMessage(Encoding.Default.GetBytes(username));
                _client.Client.Send(connectPacket.GetPacketBytes());
            }

            Algorithm = new CipherContext(_mode, _initVector, _param)
            {
                Encrypter = new
RC6(BigInteger.Parse(key).ToByteArray(), 128)
            };

            ReadPackets();
        }

        public bool IsConnectedToServer()
        {
            return _client.Connected;
        }

        private void ReadPackets()
        {
            Task.Run(() =>
            {
                while (true)
                {
                    var opcode = PacketReader.ReadByte();
                    switch (opcode)
                    {
                        case 1:
                            ConnectedEvent?.Invoke();
                            break;
                        case 5:
                            MessageReceivedEvent?.Invoke();
                            break;
                        case 10:
                            break;
                        case 15:
                            UserDisconnectedEvent?.Invoke();
                            break;
                        case 20:
                            FileReceivedEvent?.Invoke();
                            break;
                    }
                }
```

```
            });
        }

        public void SendMessageToServer(string message)
        {
            var messagePacket = new PacketBuilder();
            messagePacket.WriteOpCode(5);

messagePacket.WriteMessage(Algorithm.Encrypt(Encoding.Default.GetBytes
(message)));
            _client.Client.Send(messagePacket.GetPacketBytes());
        }

        public void SendFileToServer(string fullPath)
        {
            var messagePacket = new PacketBuilder();
            messagePacket.WriteOpCode(10);

            var index = fullPath.LastIndexOf('\\');
            var filename = fullPath.Substring(index + 1,
fullPath.Length - index - 1);

messagePacket.WriteMessage(Encoding.Default.GetBytes(filename));

            var text = File.ReadAllBytes(fullPath);
            messagePacket.WriteMessage(Algorithm.Encrypt(text));
            _client.Client.Send(messagePacket.GetPacketBytes());
        }

        public void GetFileFromServer(string filename)
        {
            var messagePacket = new PacketBuilder();
            messagePacket.WriteOpCode(20);

messagePacket.WriteMessage(Encoding.Default.GetBytes(filename));
            _client.Client.Send(messagePacket.GetPacketBytes());
        }
    }
}
```

ButtonStyle.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Style TargetType="Button" x:Key="ButtonStyle">
        <Setter Property="Foreground" Value="White" />
```

```
        <Setter Property="FontSize" Value="14" />
        <Setter Property="FontWeight" Value="SemiBold"></Setter>
        <Style.Triggers>
            <Trigger Property="IsEnabled" Value="False">
                <Setter Property="Background" Value="#0C8DD0" />
                <Setter Property="Foreground" Value="Gray" />
            </Trigger>
            <Trigger Property="IsEnabled" Value="True">
                <Setter Property="Background" Value="#229ED9" />
                <Setter Property="Foreground" Value="White" />
            </Trigger>
        </Style.Triggers>
    </Style>
</ResourceDictionary>
```

ChatStyle.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Style TargetType="ListView" x:Key="ChatStyle">
        <Setter Property="Background" Value="Transparent" />
        <Setter Property="BorderThickness" Value="0" />
        <Setter Property="Foreground" Value="White" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="FontWeight" Value="SemiBold" />
    </Style>
</ResourceDictionary>
```

MessageBoxStyle.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Style TargetType="TextBox" x:Key="MessageBoxStyle">
        <Setter Property="Height" Value="45" />
        <Setter Property="Background" Value="Transparent" />
        <Setter Property="Foreground" Value="White" />
        <Setter Property="VerticalContentAlignment" Value="Center" />
        <Setter Property="FontWeight" Value="SemiBold" />
        <Setter Property="FontSize" Value="14" />
        <Setter Property="BorderBrush" Value="Gray" />
        <Setter Property="BorderThickness" Value="2" />
        <Setter Property="CaretBrush" Value="White" />
        <Setter Property="Padding" Value="15 0 105 0" />
        <Setter Property="Margin" Value="10 0 10 10" />
```

```
    </Style>
</ResourceDictionary>
```

ScrollbarStyle.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<!--Scrollbar Thumbs-->
    <Style x:Key="ScrollThumbs" TargetType="{x:Type Thumb}">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Thumb}">
                    <Grid x:Name="Grid">
                        <Rectangle HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Width="Auto" Height="Auto"
Fill="Transparent" />
                        <Border x:Name="Rectangle1" CornerRadius="5"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Width="Auto"
Height="Auto" Background="{TemplateBinding Background}" />
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger Property="Tag" Value="Horizontal">
                            <Setter TargetName="Rectangle1"
Property="Width" Value="Auto" />
                            <Setter TargetName="Rectangle1"
Property="Height" Value="7" />
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
    <!--ScrollBars-->
    <Style x:Key="{x:Type ScrollBar}" TargetType="{x:Type ScrollBar}">
        <Setter Property="Stylus.IsFlicksEnabled" Value="false" />
        <Setter Property="Foreground" Value="#8C8C8C" />
        <Setter Property="Background" Value="Transparent" />
        <Setter Property="Width" Value="8" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type ScrollBar}">
                    <Grid x:Name="GridRoot" Width="8"
Background="{TemplateBinding Background}">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="0.00001*" />
                        </Grid.RowDefinitions>
```

```
                        <Track x:Name="PART_Track" Grid.Row="0"
IsDirectionReversed="true" Focusable="false">
                            <Track.Thumb>
                                <Thumb x:Name="Thumb"
Background="{TemplateBinding Foreground}" Style="{DynamicResource
ScrollThumbs}" />
                            </Track.Thumb>
                            <Track.IncreaseRepeatButton>
                                <RepeatButton x:Name="PageUp"
Command="ScrollBar.PageDownCommand" Opacity="0" Focusable="false" />
                            </Track.IncreaseRepeatButton>
                            <Track.DecreaseRepeatButton>
                                <RepeatButton x:Name="PageDown"
Command="ScrollBar.PageUpCommand" Opacity="0" Focusable="false" />
                            </Track.DecreaseRepeatButton>
                        </Track>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger SourceName="Thumb"
Property="IsMouseOver" Value="true">
                            <Setter Value="{DynamicResource
ButtonSelectBrush}" TargetName="Thumb" Property="Background" />
                        </Trigger>
                        <Trigger SourceName="Thumb"
Property="IsDragging" Value="true">
                            <Setter Value="{DynamicResource
DarkBrush}" TargetName="Thumb" Property="Background" />
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="false">
                            <Setter TargetName="Thumb"
Property="Visibility" Value="Collapsed" />
                        </Trigger>
                        <Trigger Property="Orientation"
Value="Horizontal">
                            <Setter TargetName="GridRoot"
Property="LayoutTransform">
                                <Setter.Value>
                                    <RotateTransform Angle="-90" />
                                </Setter.Value>
                            </Setter>
                            <Setter TargetName="PART_Track"
Property="LayoutTransform">
                                <Setter.Value>
                                    <RotateTransform Angle="-90" />
                                </Setter.Value>
                            </Setter>
                            <Setter Property="Width" Value="Auto" />
```

```
                            <Setter Property="Height" Value="8" />
                            <Setter TargetName="Thumb" Property="Tag"
Value="Horizontal" />
                            <Setter TargetName="PageDown"
Property="Command" Value="ScrollBar.PageLeftCommand" />
                            <Setter TargetName="PageUp"
Property="Command" Value="ScrollBar.PageRightCommand" />
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

ServerFilesDialog.xaml

```
<Window x:Class="Messanger.ClientWPF.Themes.ServerFilesDialog"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:Messanger.ClientWPF.Themes"
        mc:Ignorable="d"
        Title="MainWindow" Height="400" Width="320"
        Background="#36393F"
        WindowStyle="None"
        AllowsTransparency="True"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        BorderThickness="1"
        BorderBrush="Gray">
    <Grid ShowGridLines="False">
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition />
            <RowDefinition Height="40"/>
        </Grid.RowDefinitions>

        <!-- Window Border-->
        <Border Grid.Row="0"
                Grid.Column="0"
                Background="#252525">
            <Grid HorizontalAlignment="Stretch">
                <Label Content="Files"
                       Foreground="Gray"
```

```
                    FontWeight="Bold"
                    FontSize="15"
                    Margin="8 0 0 0"/>
            <StackPanel HorizontalAlignment="Right"
                        Orientation="Horizontal">
                <Button Width="35"
                        Height="35"
                        Content="✕"
                        Foreground="Gray"
                        HorizontalAlignment="Right"
                        Background="Transparent"
                        BorderThickness="0"
                        Command="{Binding CancelCommand,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
local:ServerFilesDialog}}}"/>
            </StackPanel>
        </Grid>
    </Border>

    <!-- Files list -->
    <ListView Grid.Row="1"
              x:Name="ServerFilesListView"
              Background="Transparent"
              BorderBrush="DimGray"
              BorderThickness="1"
              Foreground="White"
              FontWeight="SemiBold"
              FontSize="14"
              Margin="10 10 10 0">
    <ListView.Resources>
            <Style TargetType="{x:Type ListViewItem}">
                <Setter Property="Padding" Value="0 0 0 10 "/>
            </Style>
        </ListView.Resources>
    </ListView>

    <!-- Buttons -->
    <StackPanel Grid.Row="2"
                Orientation="Horizontal"
                HorizontalAlignment="Right">
        <Button Content="Cancel"
                Height="30"
                Width="50"
                Margin="0 0 10 0"
                Background="#2F3136"
                Foreground="White"
                FontWeight="SemiBold"
```

```
                    FontSize="14"
                    Command="{Binding CancelCommand,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
local:ServerFilesDialog}}}"/>
            <Button Content="OK"
                    Height="30"
                    Width="50"
                    Margin="0 0 10 0"
                    Background="#2F3136"
                    Foreground="White"
                    FontWeight="SemiBold"
                    FontSize="14"
                    IsDefault="True"
                    Command="{Binding OkCommand,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
local:ServerFilesDialog}}}"/>
        </StackPanel>
    </Grid>
</Window>
```

ServerFilesDialog.xaml.cs

```
using System.IO;
using System.Linq;
using System.Windows;
using MVVM.Core.Command;

namespace Messanger.ClientWPF.Themes;

public partial class ServerFilesDialog
{
    private const string ServerFiles = "\\Messanger.Server\\Files\\";

    public string FileName => (string)
ServerFilesListView.SelectedItem;
    public RelayCommand CancelCommand { get; }
    public RelayCommand OkCommand { get; }

    public ServerFilesDialog()
    {
        InitializeComponent();
        Loaded += WindowLoaded;

        CancelCommand = new RelayCommand(_ => CancelCommandMethod());
        OkCommand = new RelayCommand(_ => OkCommandMethod(), _ =>
!string.IsNullOrEmpty(FileName));
    }
```

```
    private void WindowLoaded(object sender, RoutedEventArgs e)
    {
        var directory = new
DirectoryInfo(Directory.GetCurrentDirectory());
        while (directory != null &&
!directory.GetFiles("*.sln").Any())
        {
            directory = directory.Parent;
        }
        var serverFilesFullPath = directory + ServerFiles;
        var fileEntries = Directory.GetFiles(serverFilesFullPath);
        foreach (var fileName in fileEntries)
        {
            var index = fileName.LastIndexOf('\\');
            ServerFilesListView.Items.Add(fileName.Substring(index +
1, fileName.Length - index - 1));
        }
    }

    private void OkCommandMethod()
    {
        DialogResult = true;
        Close();
    }

    private void CancelCommandMethod()
    {
        Close();
    }
}
```

ServerUserStyle.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Style TargetType="ListViewItem" x:Key="ServerUserStyle">
        <Setter Property="Background" Value="#292B2F" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ListViewItem">
                    <Border Background="{TemplateBinding Background}"
                            Height="30"
                            CornerRadius="4"
                            Margin="8 2 8 2 ">
                        <StackPanel Orientation="Horizontal"
                                    Margin="10 0 0 0">
```

```
                            <Border Width="10"
                                    Height="10"
                                    Background="Chartreuse"
                                    CornerRadius="25" />
                            <StackPanel>
                                <Label Content="{Binding Username}"
                                       Foreground="White"
                                       FontSize="14"
                                       Margin="10 0 0 0"/>
                            </StackPanel>
                        </StackPanel>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

Benaloh.cs

```
using System.Numerics;
using BenalohCryptosystem.Classes;
using Messanger.Crypto.Benaloh.Interfaces;
using Messanger.Crypto.Benaloh.Utils;

namespace Messanger.Crypto.Benaloh.Classes
{
    public sealed class Benaloh : ICrypto
    {
        private sealed class BenalohKeysGenerator : IKeysGenerator
        {
            private readonly PrimalityTestMode _mode;
            private readonly double _minProbability;
            private readonly ulong _length;

            public BenalohKeysGenerator(PrimalityTestMode mode, double
minProbability, ulong length)
            {
                _mode = mode;
                _minProbability = minProbability;
                _length = length;
            }

            public Keys GenerateKeys(BigInteger message)
            {
                var keys = new Keys();
                var p = GenerateRandomPrimeNumber();
                var q = GenerateRandomPrimeNumber();
```

```
                keys.n = BigInteger.Multiply(p, q);
                keys.phi = BigInteger.Multiply(p - 1, q - 1);
                keys.r = message;

                while (true)
                {
                    ++keys.r;
                    if ((p - 1) % keys.r == 0
                        && BigInteger.GreatestCommonDivisor(keys.r, (p
- 1) / keys.r) == 1
                        && BigInteger.GreatestCommonDivisor(keys.r, q
- 1) == 1)
                        break;
                }
                while (true)
                {
                    var pow = BigInteger.Divide(keys.phi, keys.r);
                    keys.y = BenalohUtils.GenerateRandomInteger(1,
keys.n);
                    keys.x = BigInteger.ModPow(keys.y, pow, keys.n);
                    if (keys.x != 1)
                        break;
                }

                return keys;
            }

            private BigInteger GenerateRandomPrimeNumber()
            {
                var random = new Random();
                var buffer = new byte[_length];
                while (true)
                {
                    random.NextBytes(buffer);
                    var primeNumber = new BigInteger(buffer);
                    if (primeNumber < 2)
                        continue;

                    switch (_mode)
                    {
                        case PrimalityTestMode.Fermat:
                        {
                            var test = new FermatPrimalityTest();
                            if (test.SimplicityTest(primeNumber,
_minProbability))
                                return primeNumber;
                            break;
```

```
                        }
                        case PrimalityTestMode.SolovayStrassen:
                        {
                            var test = new
SolovayStrassenPrimalityTest();
                            if (test.SimplicityTest(primeNumber,
_minProbability))
                                return primeNumber;
                            break;
                        }
                        case PrimalityTestMode.MillerRabin:
                        {
                            var test = new MillerRabinPrimalityTest();
                            if (test.SimplicityTest(primeNumber,
_minProbability))
                                return primeNumber;
                            break;
                        }
                        default:
                            throw new ArgumentOutOfRangeException();
                    }
                }
            }
        }

        private readonly BenalohKeysGenerator _keygen;
        private Keys _keys;

        public Benaloh(BigInteger message, PrimalityTestMode mode,
double minProbability, ulong length)
        {
            _keygen = new BenalohKeysGenerator(mode, minProbability,
length);
            _keys = _keygen.GenerateKeys(message);
        }

        public BigInteger Encrypt(BigInteger message)
        {
            // BigInteger u = BenalohUtils.GenerateRandomInteger(2,
_keys.n - 1);
            BigInteger u;
            while (true)
            {
                u = BenalohUtils.GenerateRandomInteger(2, _keys.n -
1);
                if (BigInteger.GreatestCommonDivisor(u, _keys.n) == 1)
                    break;
```

```
            }
            var left = BigInteger.ModPow(_keys.y, message, _keys.n);
            var right = BigInteger.ModPow(u, _keys.r, _keys.n);
            return BigInteger.Multiply(left, right) % _keys.n;
        }

        public BigInteger Decrypt(BigInteger message)
        {
            BigInteger md = 0;
            var a = BigInteger.ModPow(message, _keys.phi / _keys.r,
_keys.n);
            for (BigInteger i = 0;  i < _keys.r; i++)
            {
                var result = BigInteger.ModPow(_keys.x, i, _keys.n);
                if (result == a)
                    md = i;
            }
            return md;
        }

        public List<BigInteger> GetPublicKey()
        {
            return new List<BigInteger> { _keys.y, _keys.r, _keys.n };
        }

        public void GenerateKeys(BigInteger message)
        {
            _keys = _keygen.GenerateKeys(message);
        }
    }
}
```

FermatPrimalityTest.cs

```
using System.Numerics;
using Messanger.Crypto.Benaloh.Interfaces;
using Messanger.Crypto.Benaloh.Utils;

namespace Messanger.Crypto.Benaloh.Classes
{
    public sealed class FermatPrimalityTest : IPrimalityTest
    {
        public bool SimplicityTest(BigInteger n, double
minProbability)
        {
            if (minProbability is not (>= 0.5 and < 1))
                throw new
ArgumentOutOfRangeException(nameof(minProbability), "minProbability
```

```
must be [0.5, 1)");
            if (n == 1)
                return false;

            for (var i = 0; 1.0 - Math.Pow(2, -i) <= minProbability;
++i)
            {
                var a = BenalohUtils.GenerateRandomInteger(2, n - 1);

                if (BigInteger.ModPow(a, n - 1, n) != 1)
                    return false;
            }

            return true;
        }
    }
}
```

MillerRabinTest

```
using System;
using System.Numerics;
using Messanger.Crypto.Benaloh.Interfaces;
using Messanger.Crypto.Benaloh.Utils;

namespace BenalohCryptosystem.Classes
{
    public sealed class MillerRabinPrimalityTest : IPrimalityTest
    {
        public bool SimplicityTest(BigInteger n, double
minProbability)
        {
            if (minProbability is not (>= 0.7 and < 1))
                throw new
ArgumentOutOfRangeException(nameof(minProbability), "minProbability
must be [0.7, 1)");
            if (n == 1)
                return false;

            var d = n - 1;
            var degree = 0;
            while (d % 2 == 0)
            {
                d /= 2;
                degree += 1;
            }

            for (var i = 0; 1.0 - Math.Pow(4, -i) <= minProbability;
```

```
++i)
            {
                var a = BenalohUtils.GenerateRandomInteger(2, n - 1);
                var x = BigInteger.ModPow(a, d, n);
                if (x == 1 || x == n - 1)
                    continue;

                for (var r = 1; r < degree; ++r)
                {
                    x = BigInteger.ModPow(x, 2, n);
                    if (x == 1)
                        return false;
                    if (x == n - 1)
                        break;
                }

                if (x != n - 1)
                    return false;
            }

            return true;
        }
    }
}
```

SolovayStrassenPrimalityTest.cs

```
using System.Numerics;
using Messanger.Crypto.Benaloh.Interfaces;
using Messanger.Crypto.Benaloh.Utils;

namespace Messanger.Crypto.Benaloh.Classes
{
    public sealed class SolovayStrassenPrimalityTest : IPrimalityTest
    {
        public bool SimplicityTest(BigInteger n, double
minProbability)
        {
            if (minProbability is not (>= 0.5 and < 1))
                throw new
ArgumentOutOfRangeException(nameof(minProbability), "minProbability
must be [0.5, 1)");
            if (n == 1)
                return false;

            for (var i = 0; 1.0 - Math.Pow(2, -i) <= minProbability;
++i)
            {
```

```
                var a = BenalohUtils.GenerateRandomInteger(2, n - 1);

                if (BigInteger.GreatestCommonDivisor(a, n) > 1)
                {
                    return false;
                }

                if (BigInteger.ModPow(a, (n - 1) / 2, n) !=
BenalohUtils.Jacobi(a, n))
                {
                    return false;
                }
            }

            return true;
        }
    }
}
```

ICrypto.cs

```
using System.Numerics;

namespace Messanger.Crypto.Benaloh.Interfaces
{
    internal interface ICrypto
    {
        public BigInteger Encrypt(BigInteger message);

        public BigInteger Decrypt(BigInteger message);

        public void GenerateKeys(BigInteger message);
    }
}
```

IKeysGenerator.cs

```
using System.Numerics;

namespace Messanger.Crypto.Benaloh.Interfaces
{
    internal struct Keys
    {
        public BigInteger n;        // n = p * q
        public BigInteger y, r;     // public key
        public BigInteger phi, x;   // private key
    }

    internal interface IKeysGenerator
```

```
    {
        public Keys GenerateKeys(BigInteger message);
    }
}
```

IPrimalityTest.cs

```
using System.Numerics;

namespace Messanger.Crypto.Benaloh.Interfaces
{
    public enum PrimalityTestMode
    {
        Fermat,
        SolovayStrassen,
        MillerRabin
    }

    internal interface IPrimalityTest
    {
        public bool SimplicityTest(BigInteger n, double
minProbability);
    }
}
```

BenalohUtils.cs

```
using System.Numerics;
using System.Security.Cryptography;

namespace Messanger.Crypto.Benaloh.Utils
{
    internal static class BenalohUtils
    {
        public static BigInteger Legendre(BigInteger a, BigInteger p)
        {
            if (p < 2)
                throw new ArgumentOutOfRangeException(nameof(p), "P
must not be < 2");

            if (a == 0 || a == 1)
                return a;

            BigInteger result;
            if (a % 2 == 0)
            {
                result = Legendre(a / 2, p);
                if (((p * p - 1) & 8) != 0)
                    result = -result;
```

```
    }
    else
    {
        result = Legendre(p % a, a);
        if (((a - 1) * (p - 1) & 4) != 0)
            result = -result;
    }

    return result;
}

public static BigInteger Jacobi(BigInteger a, BigInteger b)
{
    if (BigInteger.GreatestCommonDivisor(a, b) != 1)
        return 0;

    var r = 1;
    if (a < 0)
    {
        a = -a;
        if (b % 4 == 3)
            r = -r;
    }
    do
    {
        var t = 0;
        while (a % 2 == 0)
        {
            a /= 2;
            ++t;
        }
        if (t % 2 == 1)
        {
            if (b % 8 == 3 || b % 8 == 5)
                r = -r;
        }

        if (a % 4 == b % 4 && a % 4 == 3)
            r = -r;

        var c = a;
        a = b % c;
        b = c;
    } while (a != 0);

    return r;
}
```

```
        public static BigInteger GenerateRandomInteger(BigInteger
left, BigInteger right)
        {
            var rndGenerator = RandomNumberGenerator.Create();
            var bytes = right.ToByteArray();

            BigInteger r;
            do
            {
                rndGenerator.GetBytes(bytes);
                r = new BigInteger(bytes);
            } while (!(r >= left && r < right));

            return r;
        }
    }
}
```

CipherContext.cs

```
using System.Numerics;
using Messanger.Crypto.RC6.Interfaces;
using Messanger.Crypto.RC6.Utils;

namespace Messanger.Crypto.RC6.Classes
{
    public enum EncryptionMode
    {
        ECB,
        CBC,
        CFB,
        OFB,
        CTR,
        RD,
        RDH
    }

    public sealed class CipherContext
    {
        public ICrypto Encrypter { get; set; }
        private byte[] _initializationVector;
        private readonly EncryptionMode _mode;
        private readonly string _param;

        public CipherContext(EncryptionMode mode, byte[] vector,
string param)
        {
```

```
            _mode = mode;
            _initializationVector = vector;
            _param = param;
        }

        private static byte[] Xor(byte[] leftBlock, byte[] rightBlock)
        {
            var resultBlock = new byte[leftBlock.Length];
            for (var i = 0; i < Math.Min(leftBlock.Length,
rightBlock.Length); ++i)
            {
                resultBlock[i] = (byte)(leftBlock[i] ^ rightBlock[i]);
            }
            return resultBlock;
        }

        private static byte[] PaddingPkcs7(byte[] block)
        {
            byte mod = (byte)(RC6Utils.BlockSize - block.Length %
RC6Utils.BlockSize);
            byte[] paddedData = new byte[block.Length + mod];
            Array.Copy(block, paddedData, block.Length);
            Array.Fill(paddedData, mod, block.Length, mod);
            return paddedData;
        }

        private static byte[] ListToByteArray(List<byte[]> blocksList)
        {
            var resultBlock = new byte[RC6Utils.BlockSize *
blocksList.Count];
            for (var i = 0; i < blocksList.Count; ++i)
            {
                Array.Copy(blocksList[i], 0, resultBlock,
                    i * RC6Utils.BlockSize, RC6Utils.BlockSize);
            }
            return resultBlock;
        }

        public byte[] Encrypt(byte[] block)
        {
            var paddedBlock = PaddingPkcs7(block);
            var encryptedBlocksList = new List<byte[]>();
            switch (_mode)
            {
                case EncryptionMode.ECB:
                {
                    var currBlock = new byte[RC6Utils.BlockSize];
```

```
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);

encryptedBlocksList.Add(Encrypter.Encrypt(currBlock));
                    }
                    break;
                }

                case EncryptionMode.CBC:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var currBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);

encryptedBlocksList.Add(Encrypter.Encrypt(Xor(currBlock, prevBlock)));
                        Array.Copy(encryptedBlocksList[i], prevBlock,
RC6Utils.BlockSize);
                    }
                    break;
                }

                case EncryptionMode.CFB:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var currBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);

encryptedBlocksList.Add(Xor(Encrypter.Encrypt(prevBlock), currBlock));
                        Array.Copy(encryptedBlocksList[i], prevBlock,
```

```
RC6Utils.BlockSize);
                    }
                    break;
                }

                case EncryptionMode.OFB:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var currBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);
                        var encryptedBlock =
Encrypter.Encrypt(prevBlock);
                        encryptedBlocksList.Add(Xor(encryptedBlock,
currBlock));
                        Array.Copy(encryptedBlock, prevBlock,
RC6Utils.BlockSize);
                    }
                    break;
                }

                case EncryptionMode.CTR:
                {
                    var copyIV = new
byte[_initializationVector.Length];
                    _initializationVector.CopyTo(copyIV, 0);
                    var counter = new BigInteger(copyIV);
                    var currBlock = new byte[RC6Utils.BlockSize];
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);

encryptedBlocksList.Add(Xor(Encrypter.Encrypt(copyIV), currBlock));
                        counter++;
                        copyIV = counter.ToByteArray();
                    }
                    break;
                }
```

```
                case EncryptionMode.RD:
                {
                    var currBlock = new byte[RC6Utils.BlockSize];
                    var copyIV = new
byte[_initializationVector.Length];
                    Array.Copy(_initializationVector, 0, copyIV,
                        0, RC6Utils.BlockSize);
                    var IV =  new BigInteger(copyIV);
                    var delta = new BigInteger(_initializationVector);

encryptedBlocksList.Add(Encrypter.Encrypt(copyIV));
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);

encryptedBlocksList.Add(Encrypter.Encrypt(Xor(copyIV, currBlock)));
                        IV += delta;
                        copyIV = IV.ToByteArray();
                    }
                    break;
                }

                case EncryptionMode.RDH:
                {
                    var currBlock = new byte[RC6Utils.BlockSize];
                    var copyIV = new
byte[_initializationVector.Length];
                    Array.Copy(_initializationVector, 0, copyIV,
                        0, RC6Utils.BlockSize);
                    var IV = new BigInteger(copyIV);
                    var delta = new BigInteger(_initializationVector);

encryptedBlocksList.Add(Encrypter.Encrypt(copyIV));
                    encryptedBlocksList.Add(Xor(copyIV,
PaddingPkcs7(BitConverter.GetBytes(_param.GetHashCode()))));
                    for (var i = 0; i < paddedBlock.Length /
RC6Utils.BlockSize; ++i)
                    {
                        IV += delta;
                        copyIV = IV.ToByteArray();
                        Array.Copy(paddedBlock, i *
RC6Utils.BlockSize, currBlock,
                            0, RC6Utils.BlockSize);
```

```
encryptedBlocksList.Add(Encrypter.Encrypt(Xor(copyIV, currBlock)));
                    }
                    break;
                }

                default:
                    throw new
ArgumentOutOfRangeException(nameof(_mode));
            }
            return ListToByteArray(encryptedBlocksList);
        }

        public byte[] Decrypt(byte[] block)
        {
            var decryptedBlocksList = new List<byte[]>();
            switch (_mode)
            {
                case EncryptionMode.ECB:
                {
                    var currBlock = new byte[RC6Utils.BlockSize];
                    for (var i = 0; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);

decryptedBlocksList.Add(Encrypter.Decrypt(currBlock));
                    }
                    break;
                }

                case EncryptionMode.CBC:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var currBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);
                        decryptedBlocksList.Add(Xor(prevBlock,
Encrypter.Decrypt(currBlock)));
```

```
                        Array.Copy(currBlock, prevBlock,
RC6Utils.BlockSize);
                    }
                    break;
                }

                case EncryptionMode.CFB:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var currBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);

decryptedBlocksList.Add(Xor(Encrypter.Encrypt(prevBlock), currBlock));
                        Array.Copy(currBlock, prevBlock,
RC6Utils.BlockSize);
                    }
                    break;
                }

                case EncryptionMode.OFB:
                {
                    var prevBlock = new byte[RC6Utils.BlockSize];
                    var curBlock = new byte[RC6Utils.BlockSize];
                    Array.Copy(_initializationVector, prevBlock,
prevBlock.Length);
                    for (var i = 0; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
curBlock,
                            0, RC6Utils.BlockSize);
                        var encryptedBlock =
Encrypter.Encrypt(prevBlock);
                        decryptedBlocksList.Add(Xor(encryptedBlock,
curBlock));
                        Array.Copy(encryptedBlock, prevBlock,
RC6Utils.BlockSize);
                    }
                    break;
                }
```

```
                case EncryptionMode.CTR:
                {
                    var counter = new
BigInteger(_initializationVector);
                    var currBlock = new byte[RC6Utils.BlockSize];
                    for (var i = 0; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);

decryptedBlocksList.Add(Xor(Encrypter.Encrypt(_initializationVector),
currBlock));
                        counter++;
                        _initializationVector = counter.ToByteArray();
                    }
                    break;
                }

                case EncryptionMode.RD:
                {
                    var currBlock = new byte[RC6Utils.BlockSize];
                    var copyIV = new
byte[_initializationVector.Length];
                    var delta = new BigInteger(_initializationVector);
                    Array.Copy(block, 0, currBlock,
                        0, RC6Utils.BlockSize);
                    copyIV = Encrypter.Decrypt(currBlock);
                    var IV = new BigInteger(copyIV);
                    for (var i = 1; i < block.Length /
RC6Utils.BlockSize; ++i)
                    {
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);

decryptedBlocksList.Add(Xor(Encrypter.Decrypt(currBlock), copyIV));
                        IV += delta;
                        copyIV = IV.ToByteArray();
                    }
                    break;
                }

                case EncryptionMode.RDH:
                {
```

```
                    var currBlock = new byte[RC6Utils.BlockSize];
                    var copyIV = new
byte[_initializationVector.Length];
                    var delta = new BigInteger(_initializationVector);
                    Array.Copy(block, 0, currBlock, 0,
RC6Utils.BlockSize);
                    copyIV = Encrypter.Decrypt(currBlock);
                    var IV = new BigInteger(copyIV);
                    Array.Copy(block, RC6Utils.BlockSize, currBlock,
                        0, RC6Utils.BlockSize);
                    if (!Xor(copyIV,
PaddingPkcs7(BitConverter.GetBytes(_param.GetHashCode()))).SequenceEqu
al(currBlock))
                        break;

                    for (int i = 2; i < block.Length /
RC6Utils.BlockSize; i++)
                    {
                        IV += delta;
                        copyIV = IV.ToByteArray();
                        Array.Copy(block, i * RC6Utils.BlockSize,
currBlock,
                            0, RC6Utils.BlockSize);

decryptedBlocksList.Add(Xor(Encrypter.Decrypt(currBlock), copyIV));
                    }
                    break;
                }

                default:
                    throw new
ArgumentOutOfRangeException(nameof(_mode), "");
            }
            var connectedBlocks =
ListToByteArray(decryptedBlocksList);
            var resultBlock = new byte[connectedBlocks.Length -
connectedBlocks[^1]];
            Array.Copy(connectedBlocks, resultBlock,
resultBlock.Length);
            return resultBlock;
        }
    }
}
```

RC6.cs

```
using Messanger.Crypto.RC6.Interfaces;
using Messanger.Crypto.RC6.Utils;

namespace Messanger.Crypto.RC6.Classes
{
    public sealed class RC6: ICrypto
    {
        private sealed class RC6KeysGenerator : IKeysGenerator
        {
            public uint[] GenerateRoundKeys(byte[] key, uint length)
            {
                var c = length switch
                {
                    128 => 4,
                    192 => 6,
                    256 => 8,
                    _ => throw new ArgumentException(null,
nameof(length))
                };

                int i, j;
                var L = new uint[c];
                for (i = 0; i < c; i++)
                {
                    L[i] = BitConverter.ToUInt32(key, i * 4);
                }

                var roundKey = new uint[2 * RC6Utils.R + 4];
                roundKey[0] = RC6Utils.P32;
                for (i = 1; i < 2 * RC6Utils.R + 4; i++)
                    roundKey[i] = roundKey[i - 1] + RC6Utils.Q32;

                i = j = 0;
                uint A = 0, B = 0;
                var V = 3 * Math.Max(c, 2 * RC6Utils.R + 4);
                for (var s = 1; s <= V; ++s)
                {
                    A = roundKey[i] = RC6Utils.LeftShift(roundKey[i] +
A + B, 3);
                    B = L[j] = RC6Utils.LeftShift(L[j] + A + B,
(int)(A + B));
                    i = (i + 1) % (2 * RC6Utils.R + 4);
                    j = (j + 1) % c;
                }

                return roundKey;
```

```
            }
        }

        private readonly uint[] _roundKeys;

        public RC6(byte[] key, uint length)
        {
            if (length != 128 && length != 192 && length != 256)
                throw new ArgumentException(null, nameof(length));

            var keygen = new RC6KeysGenerator();
            _roundKeys = keygen.GenerateRoundKeys(key, length);
        }

        private static byte[] ToArrayBytes(uint[] uints, int length)
        {
            var arrayBytes = new byte[length * 4];
            for (var i = 0; i < length; ++i)
            {
                var temp = BitConverter.GetBytes(uints[i]);
                temp.CopyTo(arrayBytes, i * 4);
            }
            return arrayBytes;
        }

        public byte[] Encrypt(byte[] block)
        {
            var i = block.Length;
            while (i % 16 != 0)
                i++;

            var text = new byte[i];
            block.CopyTo(text, 0);
            var cipherText = new byte[i];
            for (i = 0; i < text.Length; i += 16)
            {
                var A = BitConverter.ToUInt32(text, i);
                var B = BitConverter.ToUInt32(text, i + 4);
                var C = BitConverter.ToUInt32(text, i + 8);
                var D = BitConverter.ToUInt32(text, i + 12);

                B += _roundKeys[0];
                D += _roundKeys[1];
                for (var j = 1; j <= RC6Utils.R; ++j)
                {
                    var t = RC6Utils.LeftShift((B * (2 * B + 1)),
(int)(Math.Log(RC6Utils.W, 2)));
```

```
                    var u = RC6Utils.LeftShift((D * (2 * D + 1)),
(int)(Math.Log(RC6Utils.W, 2)));
                    A = (RC6Utils.LeftShift((A ^ t), (int)u)) +
_roundKeys[j * 2];
                    C = (RC6Utils.LeftShift((C ^ u), (int)t)) +
_roundKeys[j * 2 + 1];
                    var  temp = A;
                    A = B;
                    B = C;
                    C = D;
                    D = temp;
                }
                A += _roundKeys[2 * RC6Utils.R + 2];
                C += _roundKeys[2 * RC6Utils.R + 3];
                var returnBlock = ToArrayBytes(new [] {A, B, C, D},
4);
                returnBlock.CopyTo(cipherText, i);
            }
            return cipherText;
        }

        public byte[] Decrypt(byte[] block)
        {
            var plainText = new byte[block.Length];
            for (var i = 0; i < block.Length; i += 16)
            {
                var A = BitConverter.ToUInt32(block, i);
                var B = BitConverter.ToUInt32(block, i + 4);
                var C = BitConverter.ToUInt32(block, i + 8);
                var D = BitConverter.ToUInt32(block, i + 12);

                C -= _roundKeys[2 * RC6Utils.R + 3];
                A -= _roundKeys[2 * RC6Utils.R + 2];
                for (var j = RC6Utils.R; j >= 1; --j)
                {
                    var temp = D;
                    D = C;
                    C = B;
                    B = A;
                    A = temp;
                    var u = RC6Utils.LeftShift((D * (2 * D + 1)),
(int)Math.Log(RC6Utils.W, 2));
                    var t = RC6Utils.LeftShift((B * (2 * B + 1)),
(int)Math.Log(RC6Utils.W, 2));
                    C = RC6Utils.RightShift((C - _roundKeys[2 * j +
1]), (int)t) ^ u;
                    A = RC6Utils.RightShift((A - _roundKeys[2 * j]),
```

```
(int)u) ^ t;
                }
                D -= _roundKeys[1];
                B -= _roundKeys[0];
                var returnBlock = ToArrayBytes(new [] {A, B, C, D},
4);
                returnBlock.CopyTo(plainText, i);
            }
            return plainText;
        }
    }
}
```

ICrypto.cs

```
namespace Messanger.Crypto.RC6.Interfaces
{
    public interface ICrypto
    {
        public byte[] Encrypt(byte[] block);

        public byte[] Decrypt(byte[] block);
    }
}
```

IKeysGenerator.cs

```
namespace Messanger.Crypto.RC6.Interfaces
{
    internal interface IKeysGenerator
    {
        public uint[] GenerateRoundKeys(byte[] key, uint length);
    }
}
```

RC6Utils.cs

```
namespace Messanger.Crypto.RC6.Utils
{
    internal static class RC6Utils
    {
        public const uint P32 = 0xB7E15163;
        public const uint Q32 = 0x9E3779B9;
        public const int R = 20;
        public const int W = 32;
        public const int BlockSize = 16;

        public static uint RightShift(uint value, int shift)
        {
            return (value >> shift) | (value << (W - shift));
```

```
        }

        public static uint LeftShift(uint value, int shift)
        {
            return (value << shift) | (value >> (W - shift));
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="Messanger.KeysGeneratorWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:Messanger.KeysGeneratorWPF"
        xmlns:viewModel="clr-
namespace:Messanger.KeysGeneratorWPF.MVVM.ViewModel"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="300"
        Background="#2F3136"
        WindowStyle="None"
        AllowsTransparency="True"
        ResizeMode="NoResize"
        BorderThickness="1"
        BorderBrush="Gray">

    <Window.DataContext>
        <viewModel:MainWindowViewModel />
    </Window.DataContext>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>

        <!-- Window Border-->
        <Border Grid.Row="0"
                Grid.Column="0"
                Background="#252525"
                MouseDown="WindowBorderOnMouseDown">
            <Grid HorizontalAlignment="Stretch">
                <Label Content="Keys Generator"
```

```
                    Foreground="Gray"
                    FontWeight="Bold"
                    FontSize="15"
                    Margin="8 0 0 0"/>
            <StackPanel HorizontalAlignment="Right"
                        Orientation="Horizontal">
                <Button Width="35"
                        Height="35"
                        Content="___"
                        Foreground="Gray"
                        HorizontalAlignment="Right"
                        Background="Transparent"
                        BorderThickness="0"
                        Click="MinimizeButtonOnClick"/>
                <Button Width="35"
                        Height="35"
                        Content="✕"
                        Foreground="Gray"
                        HorizontalAlignment="Right"
                        Background="Transparent"
                        BorderThickness="0"
                        Click="CloseButtonOnClick"/>
            </StackPanel>
        </Grid>
    </Border>
    <StackPanel Grid.Row="1"
                Margin="0 10 0 0">
        <Label Content="Key"
               VerticalAlignment="Center"
               FontWeight="Medium"
               Foreground="Gray"
               FontSize="16"
               Margin="8 0 0 0"/>

        <TextBox Text="{Binding Key,
UpdateSourceTrigger=PropertyChanged}"
                 Foreground="White"
                 FontSize="13"
                 Height="40"
                 FontWeight="SemiBold"
                 Background="Transparent"
                 BorderThickness="2"
                 BorderBrush="Gray"
                 Margin="5 5 5 5"
                 Padding="10 0 10 0"
                 CaretBrush="White"
                 VerticalContentAlignment="Center">
```

```
                <TextBox.Resources>
                    <Style TargetType="{x:Type Border}">
                        <Setter Property="CornerRadius" Value="10" />
                    </Style>
                </TextBox.Resources>
            </TextBox>
        </StackPanel>

        <Button Grid.Row="2"
                Height="40"
                DockPanel.Dock="Top"
                Content="Generate key"
                Foreground="White"
                FontSize="14"
                FontWeight="SemiBold"
                Background="#229ED9"
                Margin="5 5 5 5"
                Command="{Binding GenerateButtonCommand,
UpdateSourceTrigger=PropertyChanged}">
            <Button.Resources>
                <Style TargetType="{x:Type Border}">
                    <Setter Property="CornerRadius" Value="14" />
                </Style>
            </Button.Resources>
        </Button>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;
using System.Windows.Input;

namespace Messanger.ClientWPF.MVVM.View
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void WindowBorderOnMouseDown(object sender,
MouseButtonEventArgs e)
        {
```

```
            DragMove();
        }

        private void MinimizeButtonOnClick(object sender,
RoutedEventArgs e)
        {
            if (Application.Current.MainWindow != null)
                Application.Current.MainWindow.WindowState =
WindowState.Minimized;
        }

        private void MaximizeButtonOnClick(object sender,
RoutedEventArgs e)
        {
            if (Application.Current.MainWindow.WindowState !=
WindowState.Maximized)
                Application.Current.MainWindow.WindowState =
WindowState.Maximized;
            else
                Application.Current.MainWindow.WindowState =
WindowState.Normal;
        }

        private void CloseButtonOnClick(object sender, RoutedEventArgs
e)
        {
            Close();
        }
    }
}
```

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

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

        private void WindowBorderOnMouseDown(object sender,
MouseButtonEventArgs e)
        {
            DragMove();
        }

        private void MinimizeButtonOnClick(object sender,
RoutedEventArgs e)
        {
            if (Application.Current.MainWindow != null)
                Application.Current.MainWindow.WindowState =
WindowState.Minimized;
        }

        private void CloseButtonOnClick(object sender, RoutedEventArgs
e)
        {
            Close();
        }
    }
}
```

MainWindowViewModel.cs

```
using System;
using System.Numerics;
using MVVM.Core.Command;
using MVVM.Core.ViewModel;

namespace Messanger.KeysGeneratorWPF.MVVM.ViewModel
{
    internal sealed class MainWindowViewModel : ViewModelBase
    {
        public string Key { get; set; }
        public RelayCommand GenerateButtonCommand { get; }
```

```
        public MainWindowViewModel()
        {
            GenerateButtonCommand = new RelayCommand(_ =>
            {
                Key = GenerateKey();
                RaisePropertiesChanged(nameof(Key));
            });
        }

        private string GenerateKey()
        {
            var byteKey = new byte[16];
            var random = new Random();
            for (var i = 0; i < 16; ++i)
            {
                byteKey[i] = (byte)random.Next(0, 255);
            }
            var key = new BigInteger(byteKey);
            return key.ToString();
        }
    }
}
```

RelayCommand.cs

```
using System;
using System.Windows.Input;

namespace MVVM.Core.Command
{
    public sealed class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        private readonly Predicate<object> _canExecute;

        public RelayCommand(
            Action<object> execute,
            Predicate<object> canExecute = null)
        {
            _canExecute = canExecute;
            _execute = execute ?? throw new
ArgumentNullException(nameof(execute));
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute?.Invoke(parameter) ?? true;
        }
```

```
        public void Execute(object parameter)
        {
            _execute(parameter);
        }

        public event EventHandler CanExecuteChanged
        {
            add =>
                CommandManager.RequerySuggested += value;

            remove =>
                CommandManager.RequerySuggested -= value;
        }
    }
}
```

ConverterBase.cs

```
using System;
using System.Globalization;
using System.Windows.Data;
using System.Windows.Markup;

namespace MVVM.Core.Converter
{
    public abstract class ConverterBase<TConverter> : MarkupExtension,
IValueConverter
        where TConverter: class, new()
    {
        private static TConverter _valueToProvide;

        public abstract object Convert(object value, Type targetType,
object parameter, CultureInfo culture);

        public virtual object ConvertBack(object value, Type
targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }

        public override object ProvideValue(IServiceProvider
serviceProvider)
        {
            return _valueToProvide ??= new TConverter();
        }
    }
}
```

MultiConverterBase.cs

```
using System;
using System.Globalization;
using System.Windows.Data;
using System.Windows.Markup;

namespace MVVM.Core.Converter
{
    public abstract class MultiConverterBase<TMultiConverter> :
MarkupExtension, IMultiValueConverter
        where TMultiConverter: class, new()
    {
        private static TMultiConverter _valueToProvide;

        public abstract object Convert(object[] values, Type
targetType, object parameter, CultureInfo culture);

        public virtual object[] ConvertBack(object value, Type[]
targetTypes, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }

        public override object ProvideValue(IServiceProvider
serviceProvider)
        {
            return _valueToProvide ??= new TMultiConverter();
        }
    }
}
```

ViewModelBase.cs

```
using System.ComponentModel;

namespace MVVM.Core.ViewModel
{
    public class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected void RaisePropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
```

```
        protected void RaisePropertiesChanged(params string[]
propertiesNames)
        {
            foreach (var propertyName in propertiesNames)
            {
                RaisePropertyChanged(propertyName);
            }
        }
    }
}
```

PacketBuilder.cs

```
namespace Messanger.Server.Net.IO
{
    public sealed class PacketBuilder
    {
        private readonly MemoryStream _ms;

        public PacketBuilder()
        {
            _ms = new MemoryStream();
        }

        public void WriteOpCode(byte opcode)
        {
            _ms.WriteByte(opcode);
        }

        public void WriteMessage(byte[] message)
        {
            var messageLength = message.Length;
            _ms.WriteAsync(BitConverter.GetBytes(messageLength));
            _ms.WriteAsync(message);
        }

        public byte[] GetPacketBytes()
        {
            return _ms.ToArray();
        }
    }
}
```

PacketReader.cs

```
using System.Net.Sockets;

namespace Messanger.Server.Net.IO
{
    public sealed class PacketReader : BinaryReader
    {
        private readonly NetworkStream _ns;

        public PacketReader(NetworkStream ns) : base(ns)
        {
            _ns = ns;
        }

        public byte[] ReadMessage()
        {
            var length = ReadInt32();
            var message = new byte[length];
            _ns.ReadAsync(message, 0, length);
            return message;
        }
    }
}
```

Connection.cs

```
using System.Diagnostics.CodeAnalysis;
using System.Net.Sockets;
using System.Text;
using Messanger.Server.Net.IO;

namespace Messanger.Server
{
    internal sealed class Connection
    {
        public string Username { get; }
        public Guid Uid { get; }

        public TcpClient ClientSocket { get;  }
        private readonly PacketReader _packetReader;

        public Connection(TcpClient client)
        {
            ClientSocket = client;
            Uid = Guid.NewGuid();
            _packetReader = new
PacketReader(ClientSocket.GetStream());
```

```
            _packetReader.ReadByte();
            Username =
Encoding.Default.GetString(_packetReader.ReadMessage());
            Task.Run(Process);
        }

        [SuppressMessage("ReSharper.DPA", "DPA0003: Excessive memory
allocations in LOH", MessageId = "type: System.Byte[]; size: 1820MB")]
        private void Process()
        {
            while (true)
            {
                try
                {
                    var opcode = _packetReader.ReadByte();
                    switch (opcode)
                    {
                        case 5:
                        {
                            var message = _packetReader.ReadMessage();

ServerProgram.BroadcastMessage(Encoding.Default.GetBytes(Username),
message);
                            break;
                        }

                        case 10:
                        {
                            var filename =
_packetReader.ReadMessage();
                            var text = _packetReader.ReadMessage();

                            var directory = new
DirectoryInfo(Directory.GetCurrentDirectory());
                            while (directory != null &&
!directory.GetFiles("*.sln").Any())
                            {
                                directory = directory.Parent;
                            }

                            var fullPath = directory +
ServerProgram.ServerFiles + Encoding.Default.GetString(filename);
                            File.WriteAllBytes(fullPath, text);
                            break;
                        }
                        case 20:
```

```
                        {
                            var filename =
_packetReader.ReadMessage();
                            ServerProgram.BroadcastFile(ClientSocket,
filename);
                            break;
                        }
                    }
                }
                catch (Exception)
                {
                    ServerProgram.BroadcastDisconnect(Uid.ToString());
                    ClientSocket.Close();
                    break;
                }
            }
        }
    }
}
```

ServerProgram.cs

```
using System.Net;
using System.Net.Sockets;
using System.Text;
using Messanger.Server.Net.IO;

namespace Messanger.Server
{
    internal static class ServerProgram
    {
        internal const string ServerFiles =
"\\Messanger.Server\\Files\\";
        private static TcpListener? _listener;
        private static List<Connection>? _users;

        private static void Main()
        {
            _users = new List<Connection>();
            _listener = new TcpListener(IPAddress.Parse("127.0.0.1"),
7891);
            _listener.Start();

            while (true)
            {
                var client = new
Connection(_listener.AcceptTcpClient());
                _users.Add(client);
```

```
                BroadcastConnection();
            }
        }

        internal static void BroadcastConnection()
        {
            if (_users == null)
                return;
            foreach (var user in _users)
            {
                foreach (var usr in _users)
                {
                    var broadcastPacket = new PacketBuilder();
                    broadcastPacket.WriteOpCode(1);

broadcastPacket.WriteMessage(Encoding.Default.GetBytes(usr.Username));

broadcastPacket.WriteMessage(Encoding.Default.GetBytes(usr.Uid.ToStrin
g()));

user.ClientSocket.Client.Send(broadcastPacket.GetPacketBytes());
                }
            }
        }

        public static void BroadcastMessage(byte[] username, byte[]
message)
        {
            if (_users == null)
            {
                return;
            }
            foreach (var user in _users)
            {
                var messagePacket = new PacketBuilder();
                messagePacket.WriteOpCode(5);
                messagePacket.WriteMessage(username);

messagePacket.WriteMessage(Encoding.Default.GetBytes(": "));
                messagePacket.WriteMessage(message);

user.ClientSocket.Client.Send(messagePacket.GetPacketBytes());
            }
        }

        internal static void BroadcastFile(TcpClient client, byte[]
filename)
```

```
        {
            var directory = new
DirectoryInfo(Directory.GetCurrentDirectory());
            while (directory != null &&
!directory.GetFiles("*.sln").Any())
            {
                directory = directory.Parent;
            }

            var fullPath = directory + ServerFiles +
Encoding.Default.GetString(filename);
            var text = File.ReadAllBytes(fullPath);

            var filePacket = new PacketBuilder();
            filePacket.WriteOpCode(20);
            filePacket.WriteMessage(filename);
            filePacket.WriteMessage(text);
            client.Client.Send(filePacket.GetPacketBytes());
        }

        internal static void BroadcastDisconnect(string uid)
        {
            var disconnectedUser = _users?.FirstOrDefault(
                x => x.Uid.ToString() == uid);
            if (disconnectedUser == null)
            {
                return;
            }

            _users?.Remove(disconnectedUser);
            if (_users == null)
                return;

            foreach (var user in _users)
            {
                var broadcastPacket = new PacketBuilder();
                broadcastPacket.WriteOpCode(15);

broadcastPacket.WriteMessage(Encoding.Default.GetBytes(uid));

user.ClientSocket.Client.Send(broadcastPacket.GetPacketBytes());
            }
        }
    }
}
```