МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра геоинформатики и информационной безопасности

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«РАЗРАБОТКА СИСТЕМЫ ВЫБОРОЧНОГО ШИФРОВАНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ»

по направлению подготовки 10.05.03 Информационная безопасность
автоматизированных систем
(уровень академического специалитета)
направленность (профиль) «Обеспечение информационной безопасности
распределенных информационных систем»

Студент_____А.Э. Коган
(подпись, дата)

Руководитель ВКР,
профессор, д.ф.-м.н._____В.В. Мясников
(подпись, дата)

Нормоконтролер_____Д.Б. Жмуров
(подпись, дата)

Самара 2019

МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Кафедра _геоинформатики и информационной безопасности_____

УТВЕРЖДАЮ
Заведующий кафедрой
_____ /__Сергеев В.В._____/
(подпись) *И.О.Фамилия*
«_23_»____января_____2019 г.

Задание на выпускную квалификационную работу (ВКР)

Обучающемуся__Когану Артуру Эдуардовичу_____
(ФИО, полностью)
группы__6511 – 100503 D_____
1.Тема ВКР: _Разработка системы выборочного шифрования пользовательских__ данных_____
_____
утверждена приказом по университету от «_23_»____января____2019 г. № 80-ст
2.Перечень вопросов, подлежащих разработке в ВКР:_____
__1) Анализ существующих сторонних решений по шифрованию_____ передаваемых данных._____
__2) Разработка архитектуры и пользовательского интерфейса приложения.__
__3) Реализация приложения выборочного шифрования пользовательских___ данных._____
__4) Проверка работоспособности и корректности работы разработанного___ приложения._____

3.Консультанты по разделам ВКР*(при наличии):*
раздел ВКР:_____
разрабатываемые вопросы: _____
_____
_____ /_____/
должность, степень *подпись* *И.О.Фамилия*

4.Дата выдачи задания: «_05_»____февраля____2019 г.
5.Срок представления на кафедру законченной ВКР: «____»_____20__г.

Руководитель ВКР
_Профессор, д.ф.-м.н._____ _____ /_Мясников В.В.____/
должность, степень *подпись* *И.О.Фамилия*

Задание принял к исполнению _____ /_Коган А.Э._____/
подпись обучающегося *И.О.Фамилия обучающегося*

РЕФЕРАТ

Пояснительная записка: 73 с., 26 рисунков, 14 источников, 1 приложение.

Графическая часть: 14 слайдов Microsoft PowerPoint.

ВЫБОРОЧНОЕ ШИФРОВАНИЕ, AES-128, ARGON2, WINAPI, BASE64, ШИФРОВАНИЕ ДАННЫХ, WINDOWS, БУФЕР ОБМЕНА.

Цель работы – разработка инструмента шифрования фрагмента текста, выбираемого пользователем в стороннем приложении.

Разрабатываемое приложение позволяет выборочно шифровать отправляемые собеседнику сообщения. Ключевым достоинством является совместимость с любой из программ передачи данных пользователя. Приложение позволяет осуществлять защищенный обмен сообщениями с любым количеством собеседников, что позволяет использовать программу в многопользовательском режиме.

Для удобства работы в приложении реализовано автоматическое сохранение и загрузка информации о собеседниках при завершении или запуске.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире передавать сообщения можно различными путями. Как с помощью мобильного телефона по СМС, так и по сети Internet. Если перехватывать данные мобильной сети без специальных устройств является трудной задачей, то для перехвата по сети Internet требуется компьютер или ноутбук с подключением к сети.

Для связи можно использовать как веб-приложения, запускаемые через браузер, так и специальные программы клиенты, например почтовый клиент или обычный мессенджер. С развитием информационных технологий совершенствуются и методы перехвата информации. Многие современные мессенджеры придерживаются стандартов защиты, но остаются пользователи, которые используют старые приложения, которые были разработаны во времена, когда не требовалось шифровать передаваемые данные и в них не реализована защита информации и работы по улучшению больше не ведутся. В таких случаях требуется дополнительная программа, которая обеспечит шифрование данных перед передачей по каналу связи.

Существует множество средств шифрования данных, но для их работы необходимо участие пользователя, а также установка нескольких программ и точная настройки используемых средств. Самым оптимальным решением для узкоспециализированных задач является написание своего инструмента, который будет удовлетворять требуемым условиям и режимам работы.

Так же о подобной проблеме задумаются и разработчики современных мессенджеров. В таких приложениях реализовано «прозрачное» шифрование. Пользователь не принимает никакого участия в процессе шифрования информации.

Разрабатываемое приложение должно быть универсальным, способным работать со всеми средствами обмена сообщениями. Это может быть использовано ради дополнительного шифрования для тех пользователей, которые не доверяют надежности отдельных сервисов.

В соответствии с вышеизложенными причинами были сформулированы и решены следующие задачи:

– Анализ существующих сторонних решений по шифрованию передаваемых данных.

– Разработка архитектуры и пользовательского интерфейса приложения.

– Реализация приложения выборочного шифрования пользовательских данных.

– Проверка работоспособности и корректности работы разработанного приложения.

В первой части данной работы будет произведён обзор существующих программ и решений в области шифрования передаваемых данных. Как доступных для скачивания и установки, так и тех, которым для корректной работы необходима предварительная настройка.

Во второй части работы описывается процесс выбора архитектуры и реализации готового программного обеспечения. Приводится описание каждого из модулей будущего приложения, а также алгоритмы работы приложения для каждой из задач.

В третьей части описываются алгоритмы и функции, которые были использованы при реализации будущего приложения.

В заключение, в четвёртой часть демонстрируется работоспособность и корректность дешифрования сообщений с помощью реализованного приложения.

1 Программные средства шифрования передаваемых данных

В современном мире для передачи данных в зашифрованном виде могут использоваться как готовые программные решения, так и разработанные пользователем для себя с помощью одного из языков программирования. Достоинства уже реализованных продуктов в том, что над их качеством трудилась команда разработчиков, но также там могут быть реализованы те функции, которые не нужны конкретному пользователю. В таком случае можно написать своё приложение, которое будет выполнять узкий круг требований или же реализовывать специфичный алгоритм обмена данными, но за качество и стойкость алгоритмов шифрования уже отвечает пользователь. Для обмена сообщениями по сети интернет могут быть использованы разные средства:

– E-Mail.
– Социальные сети.
– Мессенджеры.
– Клиенты и веб-версии различных чатов.

Шифрование передаваемых данных можно производить с помощью следующих программ и инструментов:

– Messenger 400.
– ProtonMail.
– Delta Chat.
– PGP.
– VPN.
– ZeroTier One.

Рассмотрим каждый из перечисленных вариантов подробнее.

1.1 Доступные программные решения

1.1.1 Messenger 400

Это основанная на базе электронного почтамта Messenger 400 [1] известная электронная почта X.400. Её производителем является фирма Infonet Software Solutions. Данная электронная почта предоставляет почтовые услуги, которые заключаются в обмене защищенными сообщениями между абонентами. Защита базирована на использовании механизма двусторонней аутентификации абонента: это означает, что и абонент на почтамте, и почтамта на абонентском пункте должны воспользоваться электронной цифровой подписью для того, чтобы сделать доступной использование данной технологией обмена сообщениями.

Необходимо отметить несколько особенностей реализации данного механизма. Такая защищенная электронная почта может стабильно функционировать в самых разных сетях (даже таких, как недорогие низкоскоростные линии обмена информации) и работать по таким протоколам связи, как TCP/IP, X.25, X.28, IPX/SPX. Хорошая производительность и низкая цена на услуги коммуникации в модуле обмена сообщениями достигается за счёт использования передовых средств маршрутизации. Также, главным достоинством рассматриваемой системы электронных писем относительно других сервисов является использование уведомлений о получении и просматривании писем, которые являются частью стандарта X.400.

Далее, рассмотрим, что входит в состав основанной на базе почтамта Messenger 400 электронной почты X.400. Основными модулями, включенными в неё, являются не только приложения, но программно-аппаратные средства.

Модули электронной почты X.400:

– Центр управления ключевой системой.

– Защищенный почтовый сервер.

– Средства криптографической защиты информации на абонентских пунктах.

При обмене сообщениями через электронный почтамт должны удовлетворятся требования к шифрованию и подписи электронных писем, возможности использования механизма двухсторонней аутентификации абонентов. Рабочие места, удовлетворяющие требованиям X.400 имеют возможность обмена сообщениями, которые являются подписанными и шифрованными. Но для успешности проведения таких операций, абоненты переписки должны быть авторизованными в системе защищенной почты X.400 с помощью алгоритма аутентификации, работающего в обе стороны.

Возможности оператора:

– Создание электронных сообщений.

– Рассылка электронных сообщений.

– Обработка электронных сообщений.

– Изменение конвертов.

– Операции над папками.

– Запись макросов.

– Обработка сетевого диска.

– Задачи сохранения защищенности.

– Обеспечение защиты конфиденциальной информации.

На одном рабочем месте может работать более одного пользователя за применения защиты профилей с помощью паролей. Абонент системы имеет доступ только к своим данным и не может получить доступ к чужим,

расположенным на одном и том же устройстве. Однако роль администратора дает право на получение любых данных из администрируемой системы.

Использование протокола LAPS скрывает недостатки использования связи через каналы с нестабильным соединением. Также, данный протокол поддерживает обмен информацией для взаимной аутентификации участников сеанса связи. Внутри корпоративной сети компьютеры обмениваются информацией по протоколу ELINK.

Но такая система отлично подходит для предприятий. Её невозможно использовать в повседневной жизни на компьютере или ноутбуке, так как для этого требуются дополнительные сервера и специализированное программное обеспечение, которое требует тонкой настройки.

1.1.2 ProtonMail

Особенностями безопасности ProtonMail [2] являются:

1) Открытый исходный код алгоритмов шифрования.

Использование библиотек с открытым исходным кодом является распространённым явлением. В таких библиотеках гарантированно нет встроенных механизмов перехвата или раскрытия зашифрованных данных, часто, такие программные вставки называют «закладками». Множество экспертов в области информационной безопасности проверяют библиотеки с открытым исходным кодом на предмет встроенных программных «закладок», чтобы доказать безопасность таких библиотек. Стандартами таких библиотек являются безопасные реализации AES, RSA и OpenPGP.

2) Отсутствие возможности доступа к пользовательским данным на сервере.

Использование клиентом личного уникального ключа шифрования гарантирует, что информация, шифрованная подобным ключом, будет доступна только ему. Даже администратор сервера не может прочесть

подобные данные. Применение специальных технических средств также не поможет при дешифровании чужого сообщения, алгоритмы были проверены специалистами со всего мира и гарантируют полную сохранность данных.

3) Сквозное шифрование.

При отправке сообщения пользователь шифрует его личным ключом. В таком виде сообщение попадает на сервера компании. Сообщения также остаются зашифрованными и при внутреннем обмене по защищенной корпоративной сети между серверами компании Proton Technologies. Естественно, при хранении на серверах все письма также остаются в зашифрованном виде. Возможность перехвата подобных сообщений при пересылке между серверами и устройствами пользователя является бессмысленной, так как данные всё равно зашифрованы. Таким образом, никто не сможет прочитать данные, предназначенные для конкретного получателя.

ProtonMail может использоваться на любых устройствах, но для работы с ним необходимо зарегистрироваться и получить адрес электронный почты. Соответственно, ProtonMail используется в целях общения по средством электронной почты.

1.1.3 Delta chat

Delta Chat [3] – децентрализованный сервис по обмену сообщениями, поддерживающий работу на таких операционных системах, как Linux, Android, Mac, iOS, использующий для работы стек протоколов E-Mail.

Основными отличительными чертами являются:

– Лицензия GPLv3, принципы разработки открытого исходного кода, применение стандартов при разработке.

– Применение алгоритма сквозного шифрования Autocrypt.

– Работа с любыми почтовым сервером, удобным пользователю.

– Канал обмена информацией строится между серверами почтовой федерации.

– Push-IMAP уведомления на протяжении всего жизненного цикла письма: отправка, доставка, получение и прочтение.

– Возможность общаться с человеком без установленного Delta Chat.

Для Delta Chat не нужно заводить новый адрес электронной почты, но он работает с использованием E-Mail. Для его работы необходим хотя бы один адрес электронной почты.

1.1.4 PGP

Следующая технология PGP [4] – это шифрование, которое сначала производит хеширование данных, затем сжимает их, шифрует симметричным ключом и в конце шифрует с открытым ключом. Причем для симметричного шифрования используется сеансовый ключ, создаваемый с помощью генератора псевдослучайных чисел, который является криптографически стойким. Таким образом, получатель зашифровывает сеансовый ключ своим открытым ключом, который связан с адресом электронной почты или пользовательским именем.

Сначала эта технология называлась «Сеть доверия». В своей первой версии она конкурировала с системой X.509, которая использовала в своей работе иерархический подход. Также X.509 базировалась на удостоверяющих центрах. Уже современные версии PGP используют оба способа шифрования данных.

Работает данная технология следующим образом. Сначала генерируются два ключа, открытый и закрытый. Для цифровой подписи и подтверждения цифровой подписи нужен открытый ключ, а для создания цифровой подписи и декодирования – закрытый. Причем при создании самих ключей необходимо задать срок действия ключей, типы, длины и их владельца. С использованием цифровой подписи PGP поддерживает аутентификацию пользователей, а также

с помощью него можно проверить целостность данных. Также данная технология сжимает данные перед шифрованием, чтобы минимизировать размер сообщений и файлов.

Но PGP неудобно дешифровать и остается проблема хранения долговременных ключей.

1.2 Другие средства защиты передаваемых данных

1.2.1 VPN

Самым простым способом является использование VPN-туннеля. Никто, кроме участников сети, не сможет прочитать информацию, передаваемую по VPN. Одним из примеров является OpenVPN [5], он имеет возможность создания TAP-интерфейса, который является виртуальным, то есть не существует физически, используемого для передачи Ethernet-фреймов, помимо IP-пакетов высоких уровней.

Но использование VPN служб не гарантирует безопасность, так как шифруются только данные передаваемые по VPN-туннелю между клиентами. В то время как многие приложения работают по клиент-серверной архитектуре и для корректной работы отправляют данные на сторонний сервер.

1.2.2 ZeroTier One

ZeroTier One [6] – это программа на основе открытого исходного кода, которая устанавливает одноранговые VPN-соединения между любыми средствами общения пользователя, будь то ноутбук, персональный компьютер, телефон, облачный ресурс и другие приложения. ZeroTier может быть определен как Social VPN или F2F (Friend to Friend), но может быть использован для виртуальной частной сети компаний. ZeroTier One может установить прямое соединение между участниками, даже если они находятся за NAT, в то время как традиционные VPN-соединения устанавливаются от клиентов к серверу, где сервер имеет публичный IP-адрес. Любые компьютеры

и устройства, подключенные к локальной сети, обычно подключаются к интернету через NAT и маршрутизатор. В домашней сети NAT и маршрутизатор, как правило, одно и то же устройство. ZeroTier One использует STUN и «hole punching», чтобы установить прямое VPN-соединение между участниками за NAT. Компании также имеют централизованный веб-портал с графическим интерфейсом для управления всеми ZeroTier One, сетевыми интерфейсами и корневыми узлами, установленными в интернете, используемыми для установления соединения аналогично ICE в WebRTC.

Но также, как и VPN, ZeroTier One использует VPN-туннель и данные посылаются на сторонние сервера.

1.2.3 Разработка собственного инструмента шифрования

Так же возможна реализация шифрования трафика, который проходит по канальному уровню, если использовать модель OSI. Большая часть программно-аппаратных средств, предназначенных для шифрования трафика, работает на сетевых уровнях не ниже третьего. Используемые модели и структура сетей третьего уровня, очень часто, чем-то похожа на структуру сети второго уровня. Поэтому, создание VPN с поддержкой криптографических средств будет трудной задачей, реализуемой различными путями, вторгающейся во всю маршрутизацию.

Использование средств прозрачного шифрования на втором уровне модели OSI не затрагивает сети маршрутизации и нет необходимости создавать туннели для передачи сетевых пакетов, что упрощает разработку. Необходимо будет лишь произвести обмен ключами шифрования между филиалами одного предприятия. Тем самым отпадет необходимость поддерживать сложные протоколы и процедуры по синхронизации общего ключа шифрования. В результате пропадут задержки использования статических ключей симметричного шифрования. При этом, размер трафика будет прежним.

1.3 Выводы и результаты

Средства шифрования требуют установки нескольких программ и точной настройки используемых средств. Также, часто программные решения нацелены на использование только одного канала связи, например, электронной почты, что делает невозможным использование их в связке с мессенджерами. Ни один из рассмотренных вариантов не является универсальным инструментом, для работы и с почтой, и с мессенджером необходимо запускать различные приложения, каждое из которых зависит от собственных дополнительных настроек. Оптимальным решением для узкоспециализированных задач является написание своего инструмента, который будет удовлетворять требуемым условиям и режимам работы.

Из всего вышесказанного делаем вывод, что мы не нашли требуемое нам приложение, которое являлось бы лёгким в настройке, универсальным в работе и удобным в использовании. В связи с этим будем реализовывать собственный инструмент для выборочного шифрования передаваемых данных.

2 Проектирование архитектуры и пользовательского интерфейса приложения

Для реализации шифрования передаваемых данных нам необходимо средство перехвата этих самых данных на уровне операционной системы. А также возможность получать зашифрованные данные от других участников общения и производить их дешифрование.

2.1 Требования к разрабатываемому приложению

Программа должна реализовывать все требования к системе выборочного шифрования и не требовать вмешательства и долгой и трудной настройки от конечного пользователя. Должна быть возможность шифровать любые сообщения, отправляемые через программы, установленные на рабочей машине конечного пользователя, и корректно дешифровать все полученные сообщения собеседника.

2.2 Структура приложения

Программа должна состоять из трёх основных модулей:

1) Шифратор/дешифратор и модуль по работе с ключами шифрования.
2) Модуль для преобразования данных.
3) Модуль по работе с операционной системой.

Шифратор/дешифратор отвечает за корректное шифрование отправляемых и дешифрование полученных пользовательских данных.

Модуль по работе с ключами шифрования отвечает за генерацию корректных ключей шифрования и поддержания актуального их состояния между двумя конкретными пользователями.

Модуль для преобразования данных необходимо для обработки бинарных данных для корректной работы с операционной системой и отображению их пользователю.

Модуль по работе с операционной системой необходимо, чтобы получать данные, введенные пользователем для дальнейшей их пересылки, а также отображения полученных от других пользователей сообщений.

2.2.1 Основные модули

Программа состоит из следующих модулей:

1) Шифратор – предназначен для шифрования данных заданным алгоритмом шифрования с использованием сгенерированного ключа шифрования.

2) Дешифратор – предназначен для дешифрования данных исходя из заданного алгоритма шифрования с использованием сгенерированного ключа дешифрования.

3) Модуль захвата отправляемых данных – необходимо для захвата введённых пользовательских данных, обработки их шифратором и возвращения программе для дальнейшей его работы по их пересылке.

4) Модуль отображения полученных данных – необходимо для перехвата полученных данных, их дешифрования и дальнейшего их отображения пользователю.

5) Модуль генерации ключей шифрования/дешифрования – необходимо для генерации корректных ключей, с помощью которых возможно произвести процедуру шифрования или дешифрования.

6) Модуль синхронизации ключей шифрования/дешифрования – необходимо для правильной работы программы и возвращения корректных результатов процедур шифрования/дешифрования даже после появления и устранения неисправностей в работе приложения или в режиме работы, когда свои данные шифрует только один из собеседников.

2.3 Выбор средств разработки и системных программных средств

Так как Windows является одной из популярных используемых операционных систем, то программа должна поддерживать данную операционную систему. Для реализации перехвата отправляемых и полученных данных будем использовать библиотеку WinAPI [7] и соответственно язык программирования C для работы с ней. Разработка программы будет происходить в среде разработки Visual Studio 2019 community edition, как в удобной и современной IDE для работы с языком программирования C.

2.4 Описание приложения

Работу приложения кратко можно описать следующим образом:

1) Обработка нажатия горячей клавиши.
2) Предобработка данных.
3) Шифрование/дешифрование.
4) Постобработка данных.

Также программа предусматривает сохранение «контактной книги» пользователя, чтобы не было необходимости вводить ключи собеседников при каждом запуске программы. Добавление и смена собеседника возможна как с помощью горячих клавиш, так и через меню главного окна программы.

На главном окне располагается информация о текущем собеседнике и место под текст, получаемый дешифровкой полученных от собеседника сообщений.

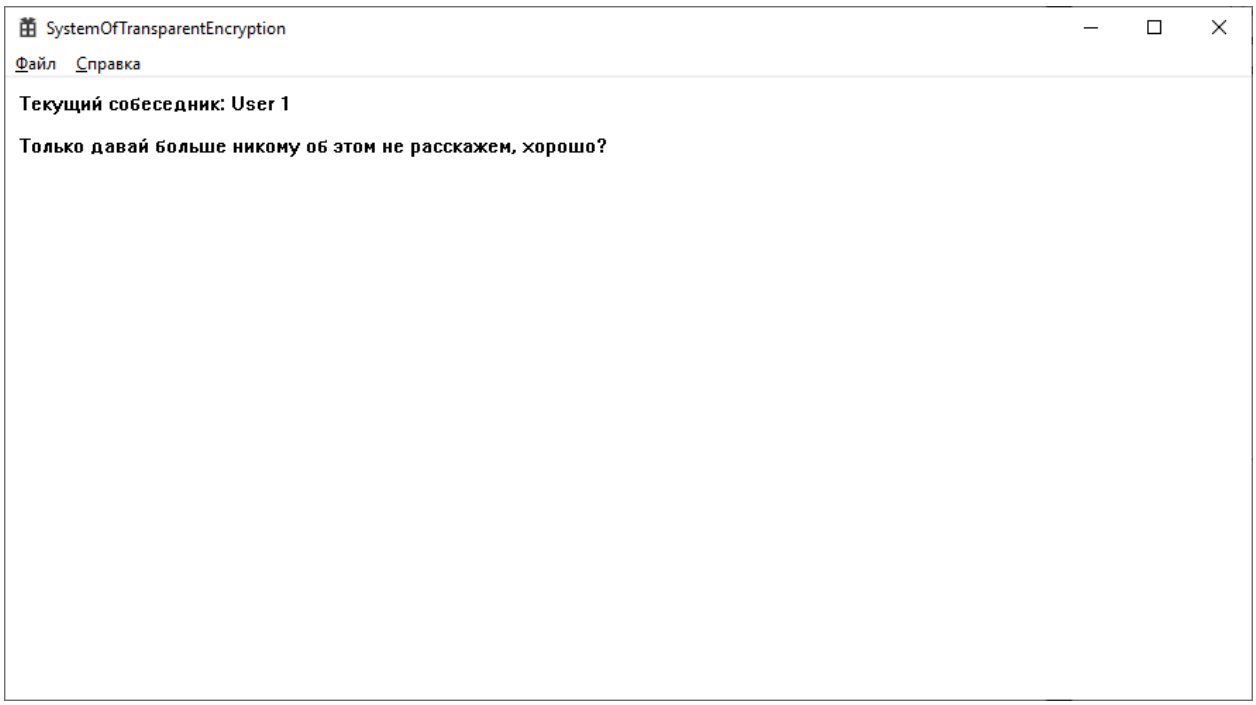

Рисунок 1 – Главное окно программы

Во время реализации программы были выделены следующие модули:

– Модуль взаимодействия с операционной системой.

– Модуль преобразования данных.

– Криптографический модуль.

– Модуль работы с собеседниками.

– Модуль графической подсистемы.

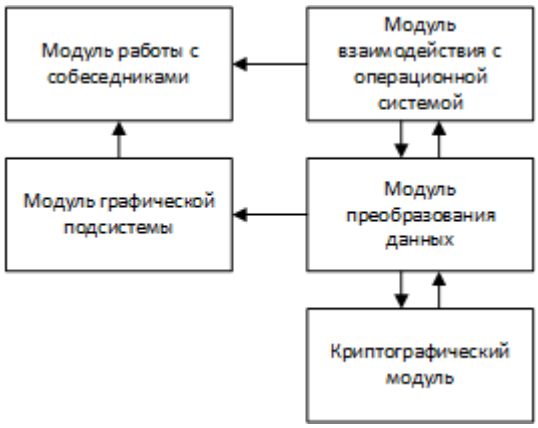

Рисунок 2 – Схема взаимодействия модулей программы

2.4.1 Описание модулей

Данный раздел описывает реализованные модули приложения. С исходным кодом [14] каждого из этих модулей можно ознакомиться в приложении А.

Модуль взаимодействия с операционной системой предназначен для передачи и получения сообщений от пользователя посредством буфера обмена. Также, в этом модуле заложена обработка горячих клавиш. Всё взаимодействие реализовано с использованием функций WinAPI. Данный модуль отвечает за создание класса главного окна, который будет обрабатывать все приходящие сообщения системы и реагировать на каждое из них по-своему, в зависимости от кого они пришли и кому предназначались. Также, он регистрирует в системе горячие клавиши, на которые должно реагировать приложение. Соответственно, класс окна обрабатывает и приходящие сообщения о нажатии только что зарегистрированных горячих клавиш. Основными являются действия шифрования и дешифрования выделенного текста, но их как-то необходимо получить от пользователя. В данном случае, приложение вызывает функцию этого модуля, которая отвечает за копирование текста в буфер обмена. После, для получения текста, мы вызываем функцию, которая получает текст из буфера обмена, она так же реализована в данном модуле. Аналогично, реализованы функции отвечающие за заполнение буфера обмена текстом и вставку текста из буфера обмена после текущего положения каретки ввода. При невозможности получить текст из стороннего приложения, пользователю предлагается самостоятельно вырезать текст в буфер обмена, отправить с помощью горячих клавиш приложению команду шифрования и вставить зашифрованное сообщение. Для дешифрования требуется поступить аналогичным образом, скопировать текст, оповестить приложение и прочитать дешифрованное сообщение в главном окне приложения.

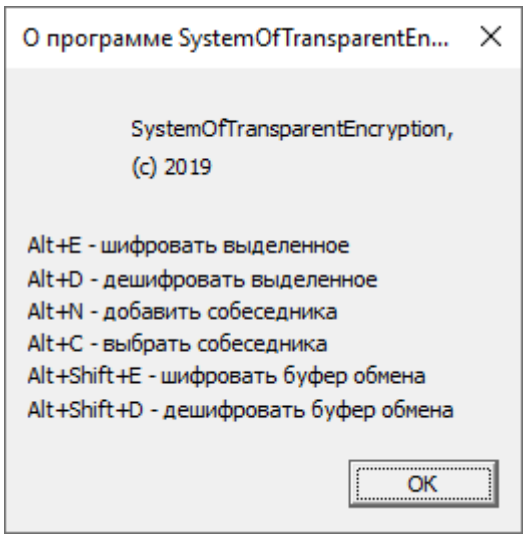

Рисунок 3 – Окно «О программе» со списком всех горячих клавиш

Модуль преобразования данных отвечает за Base64 [8] кодирование и декодирование сообщений. В этом модуле дополнительно реализованы функции работы со строками, такие как конкатенация, дополнение до требуемой длины и приведение к требуемому типу для отображения полученных сообщений на главном окне программы. Для корректной пересылки по любому каналу текстовой связи текст кодируется Base64, его описание и алгоритм работы описаны в разделе 3.3. Алгоритм шифрования AES требует, чтобы на вход подавались данные определенной длины. Для удовлетворения этим требованиям была реализована функция, которая добавляет в конец переданной строки дополнительные биты, так, чтобы результирующая длина совпадала с требуемой. Для генерации строк требуемого формата нам потребовалась функция конкатенации строк. Данная функция получает на вход 2 строки, выделяет память под результирующую строку и соединяет входные строки в одну. Для отображения текстовых данных на главном окне программы нам необходимо приведение строк к требуемым функциям WinAPI формату. В данном модуле находится функция, реализующая такое приведение типов.

Криптографический модуль использует как для шифрования и дешифрования сообщений, так и для дополнительных операций, в частности

генерация дополнительных данных требуемой длины и генерацию сессионных ключей на основе общего секрета. На вход алгоритму шифрования подается строка требуемого для конкретного алгоритма формата. Работу по расширению строки до требуемой длины выполняет модуль преобразования данных. Для усложнения взлома ключа статистическими методами используются дополнительные «одноразовые» сгенерированные случайные данные, называемые Nonce. Получение зашифрованной строки можно описать следующей формулой:

$$C = (N_{once} \,||\, E_{SK}(P)), \tag{1}$$

где C – зашифрованное сообщение;

P – открытый текст;

SK – сеансовый ключ, который генерируется по следующей формуле:

$$SK = Argon2(key, N_{once}), \tag{2}$$

где key – долговременный ключ собеседника.

Дешифрование зашифрованной строки происходит по следующей формуле:

$$P = D_{SK}(C). \tag{3}$$

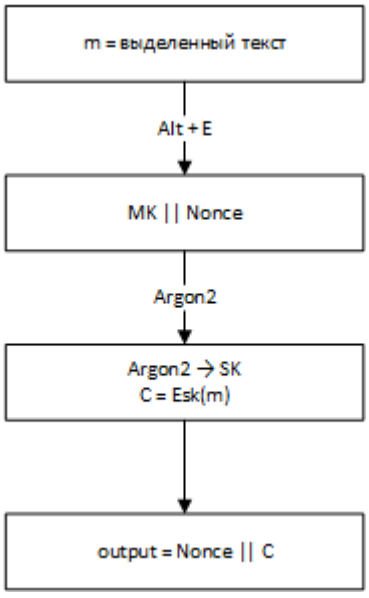

Рисунок 4 – Упрощенная схема процесса шифрования

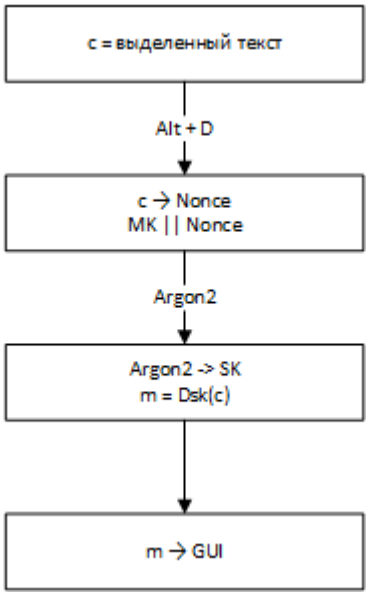

Рисунок 5 – Упрощенная схема процесса дешифрования

Модуль работы с собеседниками нужен для возможности пользователя общаться одновременно с несколькими собеседниками. Он содержит в себе также функции сохранения и загрузки файла ключей. Файл хранится в зашифрованном виде. Сохранение всех доступных собеседников в файл происходит в момент закрытия программы. Загрузка собеседников из файла происходит в момент запуска программы. Модуль отвечает за предоставления долговременных ключей шифрования собеседника, выбранного пользователем. Полученный ключ используется модулем шифрования и дешифрования для осуществления операций шифрования и дешифрования выбранных фрагментов отправляемого и получаемого текста. Модуль использует стандартные функции языка программирования C для чтения и записи файла. Файл содержит зашифрованные данные, описывающие каждого из собеседников адресной книги и их долговременные ключи шифрования, называемые «общим секретом». Для шифрования и дешифрования данных собеседников адресной книги используется пароль пользователя, который он вводит при запуске программы.

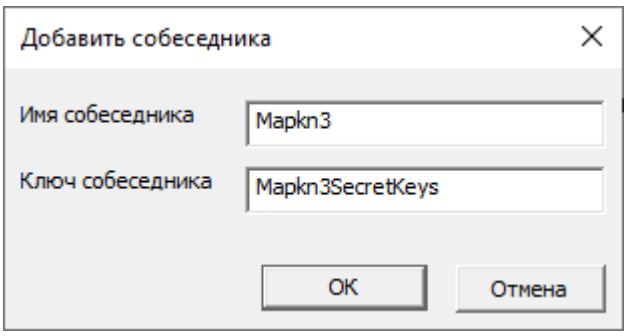

Рисунок 6 – Окно добавления нового собеседника

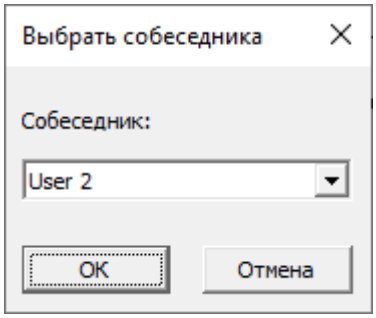

Рисунок 7 – Окно выбора текущего собеседника

Модуль графической подсистемы был добавлен для более удобного взаимодействия пользователя с программой. Он отвечает главное окно программы, где отображается вся текущая информация. Взаимодействие можно производить как через предоставленное меню, а и с помощью клавиш быстрого доступа вида Alt + <символ>. Является классическим окном операционной системы Windows. Содержит в себе главное меню приложения и окно справочной информации. В главной части окна отображается информация по текущему состоянию приложения, такая как выбранный собеседник и последнее дешифрованное сообщение. При смене собеседника через специальную форму, информация о собеседнике меняется. При дешифровании сообщения, полученный текст немедленно отображается в области окна, отведённой под текущее сообщение собеседника.

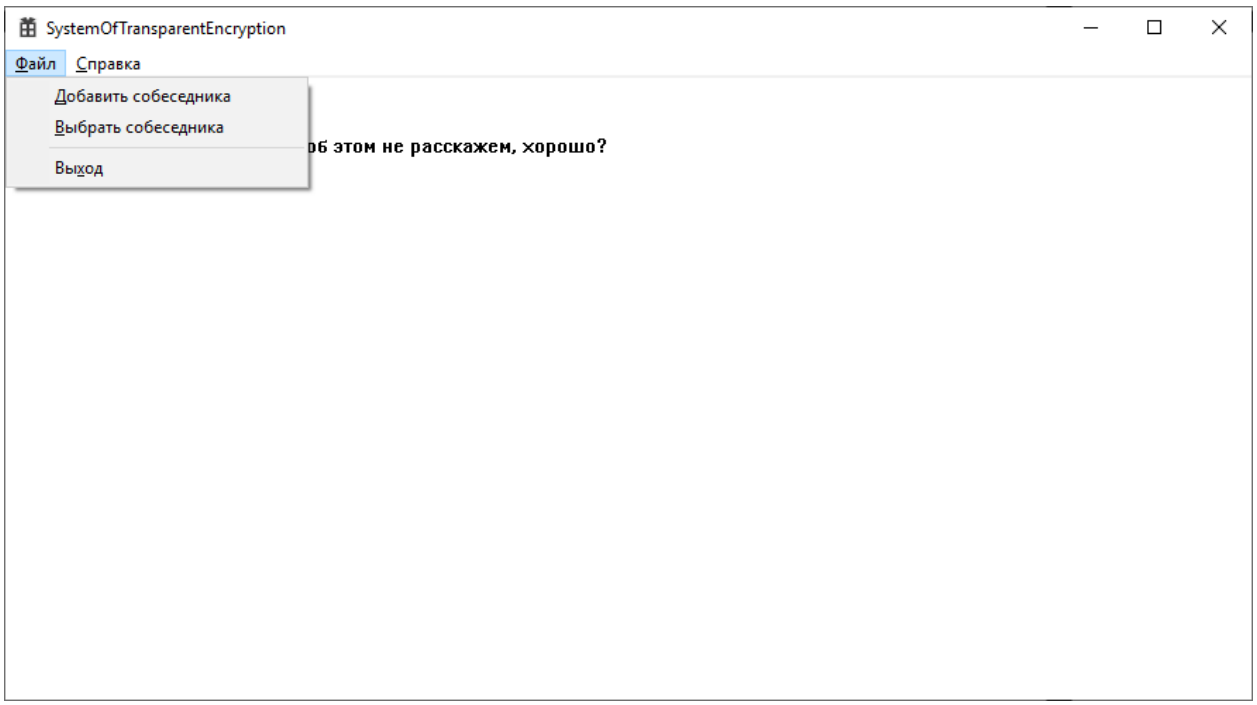

Рисунок 8 – Пункт меню «Файл»

2.4.2 Описание алгоритмов

Для упрощения работы пользователя с реализуемым приложением, мы используем ключи, называемые общим секретом, являющиеся долговременными ключами шифрования.

На основе общих секретов собеседников генерируются сеансовые ключи шифрования [9]. Для гарантии стойкости и возможности дешифровывать любые сообщения в любой момент времени также используется случайное «одноразовое» число, называемое Nonce, полученное с помощью генератора псевдослучайных чисел.

Описанное ранее число Nonce будет использовано в связке с общим секретом собеседников. Для корректности обработки Nonce преобразуется в строку, состоящую из символов алфавита Base64.

Рисунок 9 – Описание генерации шифротекста

Для генерации отправляемого шифротекста нам потребуется исходное сообщение и ключ, называемый общим секретом, собеседника. Сначала мы генерируем случайное «одноразовое» число, которое будет использовано для генерации сеансового ключа шифрования с помощью алгоритма Argon2 [10]. Новым сгенерированным сеансовым ключом с помощью алгоритма AES [11] шифруем исходное сообщение. Для гарантии того, что сообщение отобразится в любой программе и не будет явных потерь при передаче, кодируем полученный шифротекст с помощью Base64. Чтобы собеседник смог расшифровать сообщение, мы передаем Nonce вместе с кодированным шифротекстом. Так как мы не знаем, какой длины будет сообщение, мы гарантируем, что Nonce будет фиксированной, заранее известной длины, и дописываем его в начало передаваемого сообщения. Полученное сообщение возвращаем в программу, которую в данный момент использует пользователь.

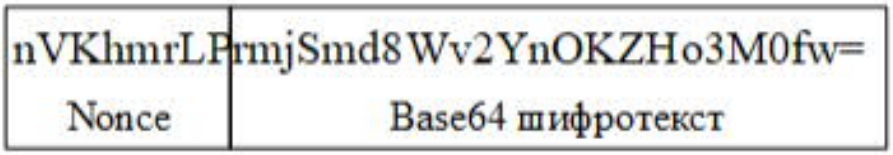

Рисунок 10 – Пример сгенерированного передаваемого сообщения

Рисунок 11 – Описание дешифрования полученного шифротекста

При получении от собеседника зашифрованного сообщения мы первым делом извлекаем из него случайное «одноразовое» число. После, алгоритмом Argon2 с помощью ключа собеседника, от которого получили сообщение, и извлечённого случайного «одноразового» числа мы генерируем сеансовый ключ шифрования. Далее, оставшуюся часть сообщения мы декодируем из Base64 в бинарные данные. Полученные бинарные данные с использованием

алгоритма шифрования AES с помощью полученного сеансового ключа мы дешифруем сообщение собеседника. Так у нас нет возможности вмешиваться в графический интерфейс других программ, установленных на устройстве пользователя, то мы отображаем полученное сообщение в специальном окне реализуемого приложения.

2.5 Выводы и результаты

Программа реализована в стиле системы выборочного шифрования и обработки пользовательских данных на уровнях получения сообщения из поля ввода и отображения полученного сообщения в специальном окне или передачи текста на отправку по каналу связи и приёма из канала связи ответа собеседника, в зависимости от реализации сетевого взаимодействия конкретной программы.

Для наименьшего вмешательства пользователя в работу программы было выбрано взаимодействие через горячие клавиши. Из-за отсутствия влияния на окна других программ, установленных на устройстве пользователя, в приложении реализован графический интерфейс, упрощающий не только выбор собеседника, но и работу с полученными сообщениями.

Продумано использование данного приложения с различными программами пользователя, что доказывает универсальность. Модульная архитектура приложения позволяет перенести программу на другую операционную систему с минимальными изменениями в программном коде. Переносимость достигается за счёт того, что основные модули реализованы платформонезависимыми и всё взаимодействие с операционной системой заложено в специально предназначенном для этого модуле.

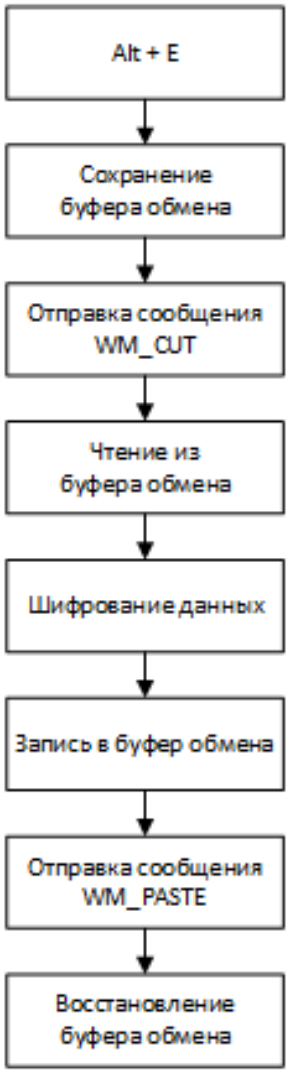

Рисунок 12 – Алгоритм работы программы при шифровании

Как видно по схеме, сначала происходит обработка нажатия горячей клавиши. После мы сохраняем текущее состояние буфера обмена, чтобы не вмешиваться в работу других процессов, и работа программы была как можно менее заметной. Далее нам нужно получить сообщение для шифрования. Мы будем это делать с помощью буфера обмена и поэтому имитируем нажатие горячей клавиши «вырезать». Вырезаем мы для того, чтобы записать зашифрованные данные на место выделенного сообщения. После этого, буфер обмена содержит необходимое нам сообщение. Получаем сообщение из буфера обмена и готовимся его шифровать. Для корректной работы AES-128 размер текста должен быть кратен 128 битам или 16 символам. Дополняем текст нулевыми битами и отправляем на шифрование. Для шифрования

генерируем некое случайное число, которое будет служить для некой синхронизации сеансовых ключей шифрования. Для генерации сеансового ключа мы используем общий секрет с собеседником и на основе его и некоего случайного числа с помощью Argon2 генерируем сеансовый ключ, которым и будет производить шифрование. После шифрования для корректного отображения в любой программе мы должны преобразовать полученные бинарные данные во что-то читаемое и гарантированно корректно отображаемое в любой программе. Для этого мы используем кодирование в Base64, что гарантирует нам отображение только символов английского алфавита, цифр и корректных знаков. После кодирования к полученной строке добавляем то самое синхронизирующее случайное число, сгенерированное ранее. Для возвращения строки пользователю мы так же используем буфер обмена и записываем в него полученную Base64 строку. Для возвращения пользователю имитируем нажатие горячей клавиши «Вставить». И, как говорилось ранее, возвращаем буфер обмена в исходное состояние.

Рисунок 13 – Алгоритм работы программы при дешифровании

Для дешифрования поступаем аналогичным образом и обрабатываем нажатие горячей клавиши. Затем также сохраняем текущее состояние буфера обмена для дальнейшего его восстановления. Получаем сообщение собеседника, но уже с помощью имитации горячей клавиши «копировать», так как вмешиваться в состояние окон «чужой» программы мы не можем, а соответственно не сможем заменить полученный текст сообщения. Теперь буер обмена содержит Base64 строку с зашифрованными данными собеседника. Получаем из строки синхронизирующее случайное число, с помощью которого вместе с общим секретом мы получим сеансовый ключ шифрования. После удаления синхронизирующего значения мы производим обработку оставшегося текста. Так как для дешифрования нам необходимы бинарные данные, а имеем мы только Base64, то производим декодирование Base64 для получения бинарного сообщения. Далее с помощью Argon2 из полученного случайного числа и общего секрета мы получаем сеансовый ключ. Дешифруем полученным ключом сообщение собеседника и отправляем его в главное окно программы. Для корректного отображения в главном окне, нам необходимо привести текст к требуемой кодировке. После корректного отображения мы восстанавливаем буфер обмена к исходному состоянию.

Работа с сессионными ключами шифрования будет происходить следующим образом:

1) Получаем текст шифруемого сообщения.

2) Генерируем из мастер-ключа и «одноразового» случайного числа значения на вход Argon2.

3) Получаем из Argon2 сессионный ключ.

4) Шифруем полученным сессионным ключом сообщение.

5) Отправляем пользователю строку, состоящую из «одноразового» случайного числа и шифротекста.

Работа с сессионными ключами дешифрования будет происходить следующим образом:

1) Получаем текст зашифрованного сообщения.

2) Получаем из строки «одноразовое» случайное число.

3) Генерируем из мастер-ключа и «одноразового» случайного числа значения на вход Argon2.

4) Получаем из Argon2 сессионный ключ.

5) Дешифруем полученным сессионным ключом сообщение.

6) Отображаем пользователю полученное сообщение.

Для гарантии того, что зашифрованные данные отобразятся у любого пользователя, после шифрования и перед дешифрованием происходит кодирование/декодирование Base64.

3 Используемые алгоритмы и функции

3.1 AES-128

В данном разделе будет рассмотрен важный алгоритм AES, который установлен стандартом шифрования, он хорошо изучен и распространен повсеместно. Его поддержка включена во многие современные процессоры Intel.

Rijndael – симметричный блочный шифр, который может обрабатывать данные блоками по 128 бит, используя ключи шифрования длиной 128, 192 и 256 бит. Rijndael спроектирован так, что позволяет использовать и другие длины блоков и ключей. Но данный шифр многие знают под названием AES.

Входом и выходом алгоритма AES являются последовательности из 128 бит. Эти последовательности рассматриваются как блоки. Количество бит в блоке будет называться длиной блока. Ключ шифрования для алгоритма AES – это последовательность из 128, 192 или 256 бит.

Базовым элементом, которым оперирует алгоритм AES, является байт – последовательность из восьми бит, обрабатываемых как единое целое. Описанные ранее последовательности бит, обрабатываются как массивы байт.

Внутри алгоритма AES выполняются операции над двумерным массивом байт, называемым матрицей состояния. Матрица состояния образована четырьмя строками, каждая из которых содержит Nb байт, где Nb – длина блока, делённая на 32. В матрице состояния обозначается символом s.

Данный алгоритм работает следующим образом. Входные данные копируются в матрицу состояния, после чего происходят операции шифрования и дешифрования, именно они изменяют матрицу состояния. В конце в выход копируется её последнее значение.

Четыре байта в каждом столбце матрицы состояния образуют 32-битные слова. Номер строки является для этих четырёх байт индексом в пределах каждого слова. Таким образом, матрица состояния может быть интерпретирована как одномерный массив 32-битных слов (столбцов).

В алгоритме AES все байты рассматриваются в качестве элементов конечного поля. Над этими элементами можно проводить операции умножения и сложения. Именно они будут описаны в следующих разделах.

3.1.1 Сложение

Для операции сложения необходимы представления элементов в виде многочленов. Тогда сложение двух элементов будет производиться как сложение коэффициентов при соответствующих степенях многочленов. Сложение выполняется с помощью операции XOR, то есть сложению по модулю 2. Следовательно, вычитание многочленов равнозначно их сложению. Альтернативно сложение элементов конечного поля может быть описано, как сложение по модулю 2 соответствующих бит в байте.

3.1.2 Умножение

При представлении байт в виде многочленов операция умножения в поле $GF(2^8)$ соответствует умножению многочленов по модулю, представляющему собой неприводимый многочлен степени 8. Многочлен является неприводимым, если его делителями являются только единица и он сам. Для алгоритма AES этим неприводимым многочленом является:

$$m(x) = x^8 + x^4 + x^3 + x + 1. \quad (4)$$

Приведение к модулю m(x) гарантирует, что результат будет двоичным многочленом со степенью меньше 8 и, следовательно, может быть представлен в виде байта. В отличие от сложения, на уровне байт нет простой операции, которая соответствовала бы умножению.

Произведение двоичного многочлена на многочлен x вычисляется путём приведения полученного простым произведением многочлена к модулю m(x), определённому выражением (3.1). Умножение на x может быть выполнено на уровне байт как сдвиг влево и последующий условный побитовый XOR с числом 00011011. Эта операция над байтами обозначается xtime(). Умножение на более высокие степени x может быть выполнено путём повторения

операции xtime(). Умножение на любую константу может быть выполнено путём сложения промежуточных результатов.

3.1.3 Описание алгоритма

Как было сказано выше, в алгоритме AES входной и выходной блоки, а также матрица состояний содержат 128 бит. Это показано равенством Nb = 4, где Nb отражает количество 32-битных слов в матрице состояния. В алгоритме AES длина ключа шифрования K равна 128, 192 или 256 бит. Длина ключа показана переменной Nk, равной 4, 6 или 8 и отражающей количество 32-битных слов в ключе шифрования. В процессе работы алгоритма AES выполняется определенное количество раундов, которое будет зависеть от длины ключа. Количество раундов показано переменной Nr, где Nr = 10, когда Nk = 4; Nr = 12, когда Nk = 6; и Nr = 14, когда Nk = 8.

В алгоритме AES в процедуре шифрования и процедуре дешифрования используется функция раунда, состоящая из четырёх различных байт-ориентированных преобразований. Этими преобразованиями являются:

– Замена байт с помощью таблицы (S-блока).

– Сдвиг строк матрицы состояния на различную величину.

– Перемешивание данных в пределах каждого столбца матрицы состояния.

– Сложение ключа раунда с матрицей состояния.

В начале процедуры шифрования вход копируется в матрицу состояния. Затем происходит операция сложения матрицы состояния и ключа раунда. Потом в зависимости от длины ключа матрица состояния видоизменяется с помощью функции раунда определенное количество раз (10, 12 или 14). Причём последний раунд немного отличается от предыдущих Nr - 1 раундов. Конечное значение матрицы состояния копируется в выход.

Функция раунда использует в качестве параметра массив подключей, который представляет собой одномерный массив 4-байтных слов. Эти слова вычисляются с помощью процедуры расширения ключа. Отдельные

преобразования – SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() – обрабатывают матрицу состояния.

Преобразование SubBytes() выполняет нелинейную замену байт матрицы состояния с помощью таблицы (S-блока). При этом каждый байт обрабатывается независимо от других.

Преобразование ShiftRows() выполняет циклический сдвиг байт в трёх последних строках матрицы состояния на различное число байт (смещений). Первая строка не сдвигается. В итоге байты сдвигаются на «младшие» позиции в строке, в то время как «самые младшие» байты перемещаются вокруг строки в её «вершину».

Преобразование MixColumns() обрабатывает матрицу состояния столбец за столбцом. Каждый столбец используется в качестве четырёхчленного многочлена. Столбцы рассматриваются как многочлены в поле $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на фиксированный многочлен a(x):

$$a(x) = 3x^3 + x^2 + x + 2. \quad (5)$$

Преобразование AddRoundKey() выполняет сложение ключа раунда и матрицы состояния с помощью простой побитовой операции XOR. Каждый ключ раунда состоит из Nb слов массива подключей. Каждое из этих Nb слов складывается со столбцами матрицы состояния.

Процедура расширения ключа используется в алгоритме AES для создания массива подключей из ключа шифрования K. Всего генерируется Nb(Nr + 1) слов. В начале алгоритма требуется Nb слов и затем в каждом из Nr раундов требуется Nb слов ключевых данных.

Функция SubWord() принимает на вход 4-байтное слово. Выходное слово формируется путём замены каждого из этих четырёх байт с помощью S-блока.

Функция RotWord() принимает на вход слово и выполняет циклическую перестановку.

Первые Nk слов расширенного ключа заполняются ключом шифрования. Каждое последующее слово, вычисляется путём выполнения операции XOR между предыдущим словом, и словом, находящимся на Nk позиций раньше.

При вычислении слов, находящихся на позициях кратных Nk, над предыдущим словом производятся дополнительные операции. Сначала в слове производится циклический сдвиг байт функцией RotWord(). Затем функция SubWord() изменяет все четыре байта слова с помощью таблицы. После чего выполняется операция XOR между выходом функции SubWord() и словом из массива констант.

Важно отметить, что процедура расширения ключа для 256-битных ключей шифрования (Nk = 8) немного отличается от процедуры для 128- и 192-битных ключей. Если Nk = 8 и i – 4 кратно Nk, то перед операцией XOR предыдущее слово обрабатывается функцией SubWord().

Преобразования, составляющие процедуру шифрования, могут быть инвертированы и применены в обратном порядке для получения прямой процедуры дешифрования алгоритма AES. Отдельные преобразования, использующиеся в процедуре дешифрования – InvShiftRows(), InvSubBytes(), InvMixColumns() и AddRoundKey() – обрабатывают матрицу состояния.

Преобразование InvShiftRows() является обратным к ShiftRows(). Байты в трёх последних строках матрицы состояния циклически сдвигаются на различное число байт (смещений). Первая строка (с номером r = 0) не сдвигается. Остальные строки сдвигаются аналогично ShiftRows().

Преобразование InvSubBytes() является обратным к преобразованию замены байт. Преобразование InvSubBytes() изменяет каждый байт матрицы состояния с помощью инвертированного S-блока. Для этого выполняется преобразование, обратное аффинному преобразованию, и затем находится обратный по умножению элемент в поле $GF(2^8)$.

Преобразование InvMixColumns() является обратным к MixColumns(). Преобразование InvMixColumns() обрабатывает матрицу состояния столбец за столбцом. Каждый столбец используется в качестве четырёхчленного

многочлена. Столбцы рассматриваются как многочлены в поле $GF(2^8)$ и умножаются по модулю $x^4 + 1$x на фиксированный многочлен $a^{-1}(x)$:

$$a^{-1}(x) = 11x^3 + 13x^2 + 9x + 14. \quad (6)$$

Преобразование AddRoundKey(), описанное ранее, является обратным само к себе, так как включает в себя только операцию XOR.

В прямой процедуре дешифрования порядок следования преобразований отличается от порядка, применённого в процедуре шифрования. При этом для шифрования и дешифрования используется одинаковый массив подключей. Однако некоторые свойства алгоритма AES позволяют использовать эквивалентную процедуру дешифрования, в которой последовательность преобразований совпадает с последовательностью в процедуре шифрования (при этом преобразования заменяются на обратные). Это достигается путём изменения массива подключей.

Двумя свойствами, благодаря которым возможна эквивалентная процедура дешифрования, являются:

– Преобразования SubBytes() и ShiftRows() коммутативны. То есть преобразование SubBytes(), следующее сразу после преобразования ShiftRows(), будет эквивалентно преобразованию ShiftRows(), следующему сразу после преобразования SubBytes(). То же верно и для обратных к ним преобразований – InvSubBytes() и InvShiftRows();

– Операции перемешивания столбцов – MixColumns() и InvMixColumns() – являются линейными по отношению к входному столбцу.

Эти свойства позволяют поменять местами преобразования InvSubBytes() и InvShiftRows(). Преобразования AddRoundKey() и InvMixColumns() тоже можно поменять местами, если при этом столбцы (слова) массива подключей дешифрования будут изменены преобразованием InvMixColumns().

Эквивалентная процедура дешифрования образуется путём обмена местами преобразований InvSubBytes() и InvShiftRows(), а также обмена

местами в “цикле раунда” преобразований AddRoundKey() и InvMixColumns(). При этом подключи дешифрования, используемые с 1 по Nr -1 раунд, должны быть изменены преобразованием InvMixColumns(). Но первые и последние Nb слов массива подключей дешифрования не должны быть им изменены.

С учётом этих изменений структура эквивалентной процедуры дешифрования является более эффективной.

3.2 Argon2

Алгоритм Argon2 был выбран по причине того, что он является победителем конкурса Password Hashing Competition, который был проведён для создания новой функции хеширования паролей, которая удовлетворяла бы ряду требований: число проходов по блокам памяти, общему объёму затрагиваемой памяти и устойчивость к криптоанализу. Таким образом, функция Argon2 способна контролировать затрачиваемое на хеширование время и делает атаки полного перебора бессмысленными, так как на вычисление каждого хэша должно затрачиваться не менее заданного промежутка времени.

В современном мире остро стоит вопрос замедления хеширования. Это было еще до введения моды на быстрые алгоритмы по нахождению исходного значения для конкретного хэша. Существуют разные способы замедления, такие как применение хеширования несколько раз подряд или использование соли, но специальные устройства упрощают перебор. С такими устройствами не справиться, даже если использовать алгоритм bcrypt [12].

Для решения возникшей проблемы было решено провести конкурс, победителем которого станет алгоритм, который сложно ускорить на специальных устройствах и GPU, но при этом разработчик должен иметь возможность произвести точную настройку по своим требованиям. Таким алгоритмом стал Argon2.

В алгоритме Argon2 можно настроить следующие параметры:

– Размер полученного хэша, в байтах.

– Число прогонов алгоритма.

– Секретный ключ.

– Степень параллелизма.

– Заполненность памяти при работе.

– Встроенная информация.

Так же, KDF [13] Argon2 существует в двух вариантах: Argon2i и Argon2d. Argon2i рассчитан на нагрузку памяти, в связи с чем является более медленным. Argon2d быстрее, но на нём возможно провести timing атаку, а также он устойчив против GPU перебора.

Argon2i используется при хэшировании паролей, Argon2d — для криптовалют, где timing атаки являются бесполезными.

Алгоритм лучше всего проявляет себя на архитектурах x86/x64, в связи с чем ускорение на ASIC/GPU и прочих устройствах будет являться довольно трудной задачей. Достоинством является огромное число обходов памяти, там формируется матрица хэшей довольно большого объёма, взаимозависимые между собой. В это же время происходит довольно сложная их обработка.

3.3 Base64

Алгоритм преобразования данных, который работает на уровне байт и является перекодировщиком из произвольного алфавита в алфавит Base64, содержащий 64 символа: A-Z, a-z, 0-9, /, +. Для выравнивания недостающих бит и корректного декодирования используется последовательность из одного или нескольких специальных символов, чаще всего, это знак равенства «=».

Достоинства:

– Перевод бинарных, «нечитаемых» данных в последовательность валидных символов.

– Избыточность данных, по сравнению с другими Base-кодировками составляет всего 133.(3)% от размера кодируемой информации.

Недостатки:

– Важность регистра символа при его декодировании.

Помимо прочего, так как мы получаем данные в кодировке Unicode, то каждый символ занимает 2 байта, но после кодирования в Base64 каждый символ занимает 1 байт, что в 2 раза сокращает объём требуемой памяти.

Приведём пример работы алгоритма:

1) Исходное слово: «Base64».

2) Берём код каждого символа: 66, 97, 115, 101, 54, 52.

3) Переводим десятичный код в бинарный: 01000010, 01100001, 01110011, 01100101, 00110110, 00110100.

4) Перераспределяем биты в группы по 6 бит: 010000, 100110, 000101, 110011, 011001, 010011, 011000, 110100.

5) Переводим бинарный код в Base64-индексы: 16, 38, 5, 51, 25, 19, 24, 52.

6) Находим символы кодировки Base64 по их индексам: «QmFzZTY0»

Из приведённого алгоритма становится ясно, что слово «Base64» в кодировке ASCII превращает в слово «QmFzZTY0» в кодировке Base64.

Соответственно, декодирование из Base64 происходит аналогичным образом, но в обратном порядке, то есть группы из 6 бит перераспределяются в группы по 8 бит, что соответствует размеру стандартных кодировок.

1) Исходное слово: «QmFzZTY0».

2) Берём Base64-индекс каждого символа: 16, 38, 5, 51, 25, 19, 24, 52.

3) Переводим десятичный код в бинарный группами по 6 бит: 010000, 100110, 000101, 110011, 011001, 010011, 011000, 110100.

4) Перераспределяем биты в группы по 8 бит: 01000010, 01100001, 01110011, 01100101, 00110110, 00110100.

5) Переводим бинарный код в код символа: 16, 38, 5, 51, 25, 19, 24, 52.

6) Берём символ по его коду и получаем: «Base64».

4 Испытания разработанного приложения

4.1 Испытание корректной работы приложения

Проверим, что приложение корректно шифрует текст заданным ключом и правильно дешифрует полученный текст.

Для добавления в приложение нового собеседника нужно сделать следующее:

1) Нажать Alt+N для вызова формы добавления нового собеседника.

2) Заполнить поля имени и ключа собеседника.

3) Нажать OK.

4) Проверить, что на главном окне приложения указан верный текущий собеседник.

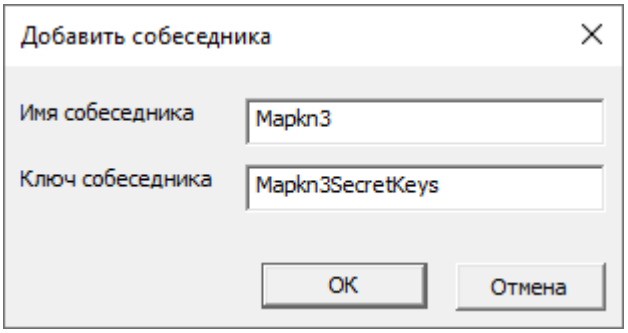

Рисунок 14 – Добавление тестового собеседника

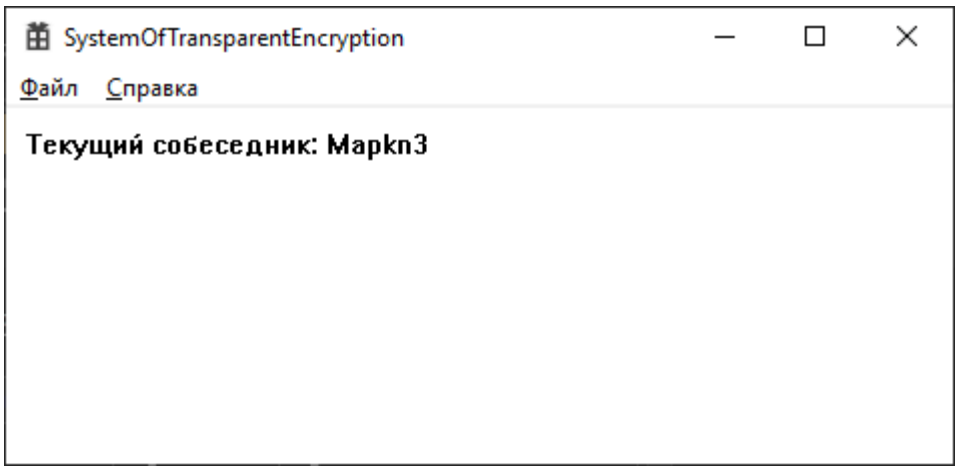

Рисунок 15 – Информация о текущем собеседнике на главном окне

Для проверки корректной работы шифрования и дешифрования выполним следующие шаги:

1) Открыть текстовый редактор.

2) Набрать любое проверочное сообщение.

3) Выделить сообщение.

4) Нажать Alt+E для шифрования сообщения.

5) Выделить полученный шифротекст.

6) Нажать Alt+D для дешифрования выделенного текста.

7) Проверить, что на главном окне приложения отображается верное сообщение.

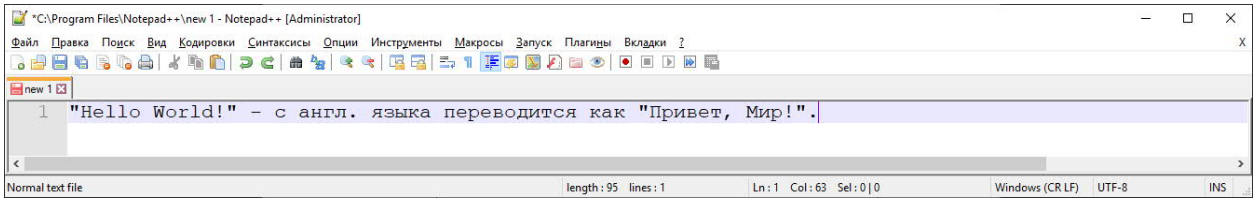

Рисунок 16 – Исходное сообщение

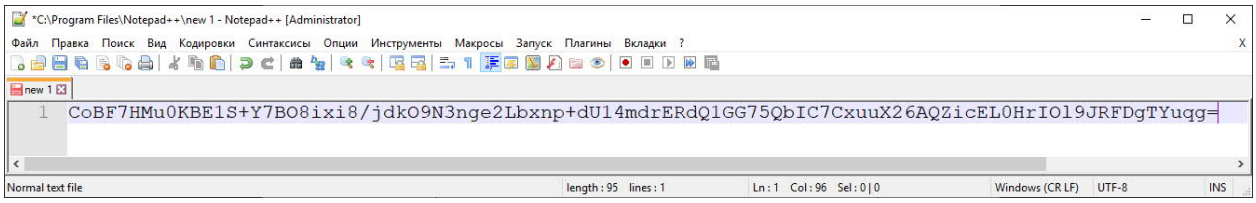

Рисунок 17 – Зашифрованное сообщение

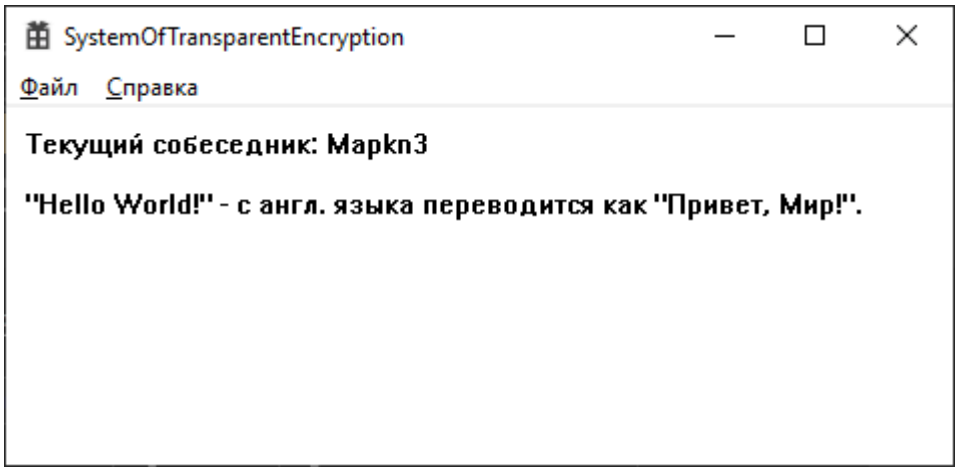

Рисунок 18 – Дешифрованное сообщение

4.2 Испытание дешифрования неверным ключом

Для проверки того, что нельзя дешифровать сообщение другим ключом, добавим в приложении ещё одного собеседника по алгоритму, описанному в предыдущем пункте.

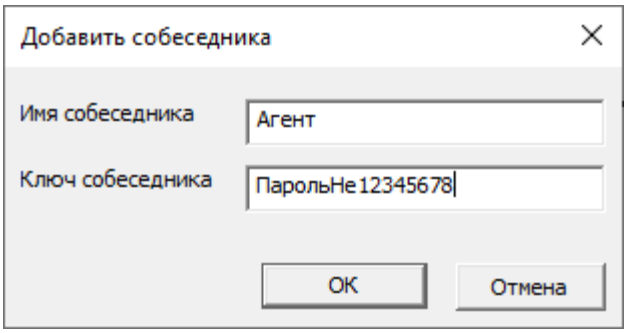

Рисунок 19 – Добавление ещё одного собеседника

Дешифруем сообщение, указанное на рисунке 17 с использованием ключа нового собеседника. Для этого выделяем сообщение и нажимаем Alt+D. На главном окне приложения мы видим несвязный набор символов, что свидетельствует о том, что не получилось дешифровать сообщение с помощью ключа, отличного от того, которым шифровали сообщение.

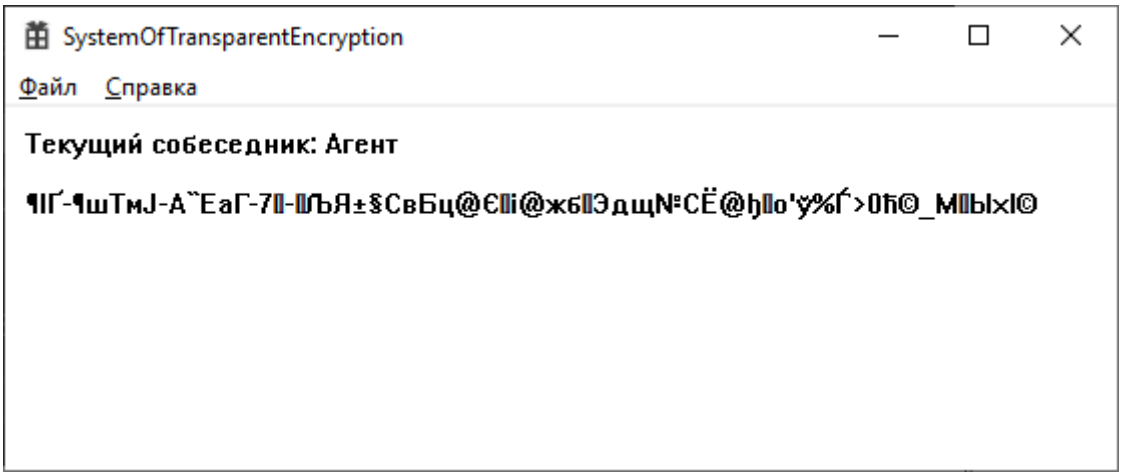

Рисунок 20 – Дешифрованное сообщение неверным ключом

Если приложение не может вырезать текст из сторонней программы, то можно использовать функцию шифрования буфера. По окончанию шифрования будет выведено справочное окно.

4.3 Испытание корректной работы при отправке сообщения через мессенджер

Для проверки корректности работы при передаче сообщений через мессенджеры будем использовать «Telegram», как один из распространенных и ежедневно используемых. Наш собеседник отправляет нам фразу: «Отправленное через «Telegram» сообщение корректно дешифровано», предварительно зашифровав его с помощью разработанного приложения. Результаты шифрования можно увидеть на следующем рисунке. Все предыдущие сообщения чата в Telegram скрыты.

Рисунок 21 – Полученное в Telegram зашифрованное сообщение

Дешифруем полученное сообщение с помощью следующих действий:

1) Выделить полученное сообщение.

2) Скопировать выделенный текст, нажав Ctrl+C.

3) Нажать Alt+Shift+D для дешифрования теста, находящегося в буфере обмена.

4) Прочитать полученное сообщение в главном окне приложения.

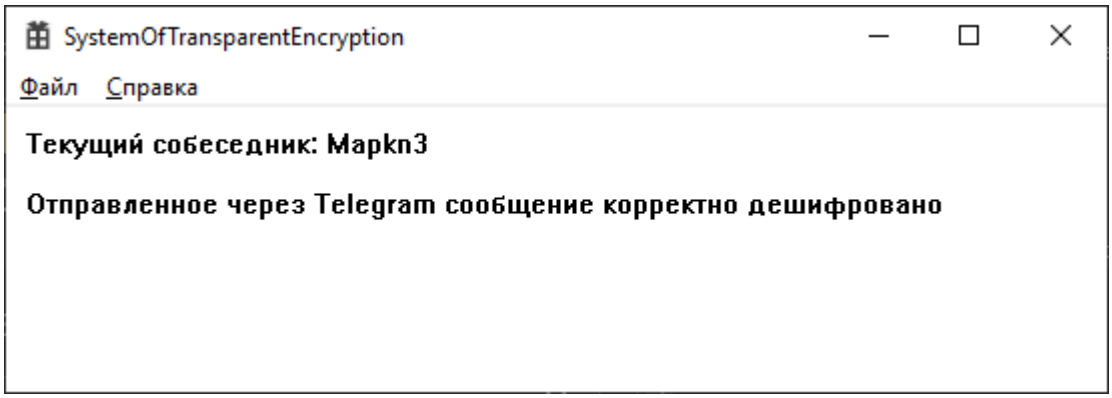

Рисунок 22 – Дешифрованное сообщение, полученное через Telegram

4.4 Испытание корректной работы при отправке сообщения через электронную почту в браузере

Для проверки работы при использовании переписки через электронную почту, попросим собеседника отправить на личный электронный адрес фразу: «Передаю привет по E-Mail», предварительно её зашифровав.

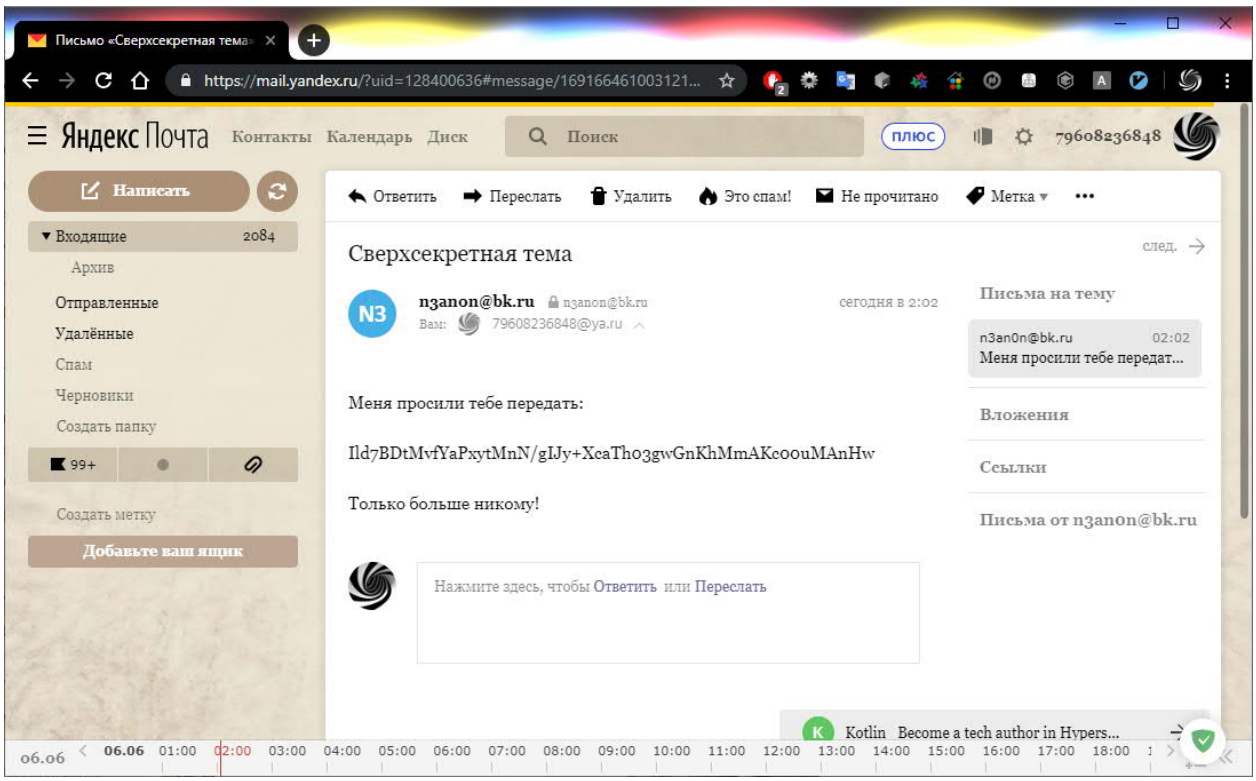

Рисунок 23 – Полученное по электронной почте зашифрованное сообщение

Рисунок 24 – Дешифрованное сообщение, полученное по электронной почте

На рисунке 23 показано, что необязательно всё сообщение должно быть зашифровано. Можно посылать смешанные сообщения, состоящие как из зашифрованного, так и из обычного текста. Также, мы можем посылать в одном сообщении несколько отдельных шифрованных фраз. Главное оповестить собеседника о том, как именно отличать разные сообщения друг от друга, например каждую шифрованную фразу начинать с новой строки. Если

же выделить весь текст целиком, то не получится дешифровать все фразы одновременно.

4.4 Испытание корректной работы при отправке сообщения через социальные сети

Проверим работу приложения, если сообщение будет переслано через сервис сообщений социальной сети VK. Данная социальная сеть является одной из самых популярных для общения в интернете. Контрольной фразой будет: «Представляешь, это работает даже в социальных сетях!».

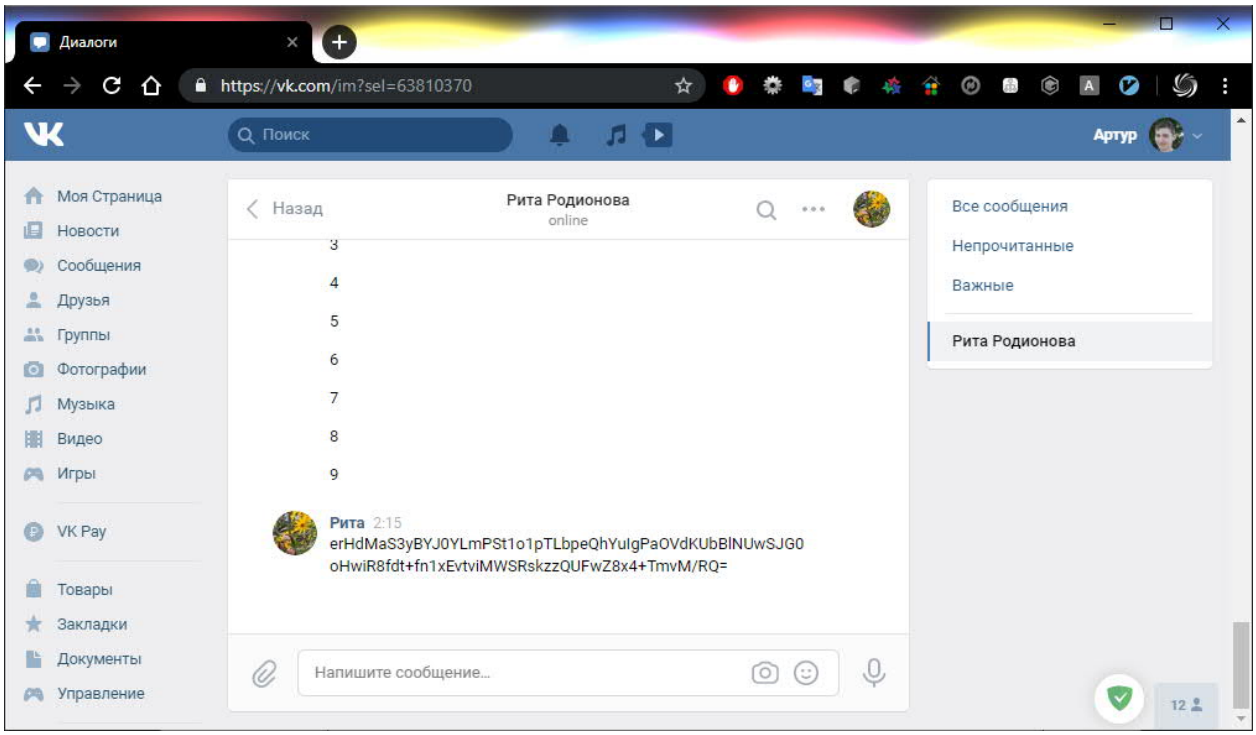

Рисунок 25 – Полученное через социальную сеть сообщение

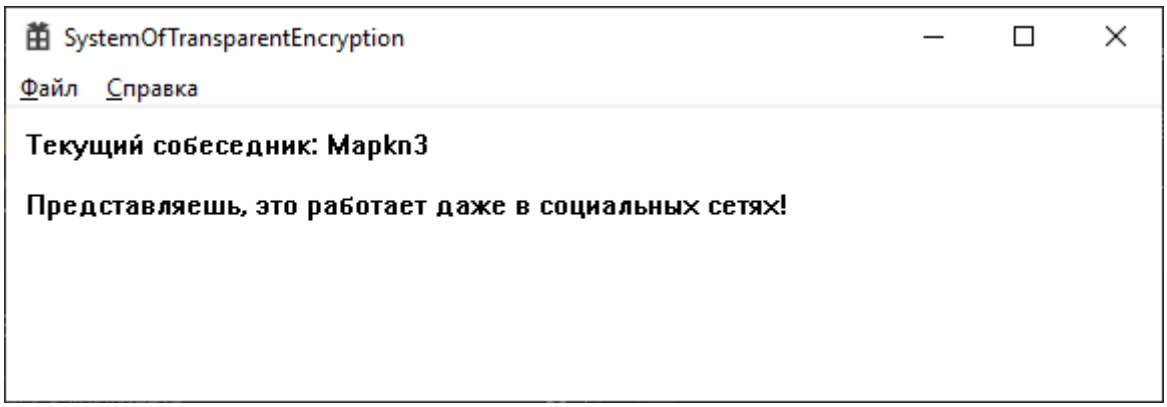

Рисунок 26 – Дешифрованное сообщение, полученное через социальную сеть

ЗАКЛЮЧЕНИЕ

Проанализировав существующие решения в области шифрования передаваемых данных мы пришли к выводу, что они не являются универсальными и не каждое их них легко в настройке и эксплуатации, в связи с чем была проведена разработка собственного инструмента шифрования выборочных данных, способного работать как с разными приложениями обмена данными, так и на разных устройствах.

Целью проектирования архитектуры и разработки приложения являлось решение вопросов удобства использования и универсальности, которая заключается в возможности использования с любой программой передачи текстовых данных. Модульная архитектура позволят использовать приложение не только в операционной системе Windows. При реализации использовались современные алгоритмы шифрования и генерации ключей, такие как AES и Argon2. Была учтена особенность бинарных данных быть воспринятыми как специальные символы и реализовано дополнительное кодирование Base64, что является одним из признаков универсальности, то есть мы не зависим от формы представления данных, а именно, работаем с широко доступным текстовым видом представления и передачи данных.

Испытания, проводимые в условиях, максимально приближенных, или, чаще всего, являющихся реальными условиями использования разработанного программного обеспечения, показали корректность работы инструмента выборочного шифрования, его универсальность, а также продемонстрировали возможность работы в многопользовательском режиме.

В ходе выполнения работы были выполнены все поставленные задачи и достигнута цель работы, приложение реализовано и корректно работает.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1 Защищенная электронная почта X.400: [Электронный ресурс] // МОПНИЭИ Средства Криптографической Защиты Информации – URL: https://security.ru/default.php?target=x400&style=products (дата обращения 16.03.2019).

2 Безопасность ProtonMail: [Электронный ресурс] // ProtonMail – URL: https://protonmail.com/ru/security-details (дата обращения 16.03.2019).

3 Главная страница Delta Chat: [Электронный ресурс] // Delta Chat – URL: https://delta.chat/ru/ (дата обращения 16.03.2019).

4 Главная страница OpenPGP: [Электронный ресурс] // OpenPGP – URL: https://www.openpgp.org/ (дата обращения 16.03.2019).

5 Главная страница OpenVPN: [Электронный ресурс] // OpenVPN – URL: https://openvpn.net/ (дата обращения 16.03.2019).

6 Главная страница ZeroTier: [Электронный ресурс] // ZeroTier – URL: https://www.zerotier.com/ (дата обращения 16.03.2019).

7 Создание классических приложений для компьютеров с Windows: [Электронный ресурс] // Центр разработки для Windows – URL: https://docs.microsoft.com/ru-ru/windows/apps/desktop/ (дата обращения 06.04.2019).

8 The Base16, Base32, and Base64 Data Encodings: [Электронный ресурс] // IETF Tools – URL: https://tools.ietf.org/html/rfc4648/ (дата обращения 07.04.2019).

9 Брюс Шнайер: Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си [Текст] / Брюс Шнайер // Триумф — 2012.

10 Argon2: the memory-hard function for password hashing and other applications: [Электронный ресурс] // GitHub – URL: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf (дата обращения 04.05.2019).

11 Advances Encryption Standart (AES): [Электронный ресурс] // National Institute of Standards and Technology – URL:

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата обращения 04.05.2019).

12 A Future-Adaptable Password Scheme: [Электронный ресурс] // The OpenBSD Project – URL: http://www.openbsd.org/papers/bcrypt-paper.pdf (дата обращения 04.05.2019).

13 Boneh, D. A Graduate Course in Applied Cryptography [Текст] / Dan Boneh, Victor Shoup // Stanford University — 2017.

14 SystemOfTransparentEncryptionWin-VS: [Электронный ресурс] // GitHub – URL: https://github.com/Mapkn3/SystemOfTransparentEncryptionWin-VS (дата обращения 27.05.2019).

ПРИЛОЖЕНИЕ А

Исходный код приложения выборочного шифрования

```
#include <windows.h>
#include <tlhelp32.h>
#include <winuser.h>
#include <stdio.h>
#include <wincrypt.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <time.h>

#include "framework.h"
#include "SystemOfTransparentEncryption.h"
#include "apacheBase64.h"
#include "argon2.h"
#include "aes.hpp"

#pragma comment(lib, "Crypt32.lib")
#pragma comment(lib, "Argon2RefDll.lib")

#define MAX_LOADSTRING 100

#define SALT_SIZE 8
#define NAME_SIZE 21
#define KEY_SIZE 17
#define MAX_USERS 10
#define BACKUP_FILENAME "bu.data"

struct User {
    BYTE name[NAME_SIZE];
    BYTE key[KEY_SIZE];
} users[MAX_USERS];


int userCount = 0;
int currentUser = -1;
uint8_t* password;

// Глобальные переменные:
HINSTANCE hInst; // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING]; // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING]; // имя класса главного
окна

ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

```
    VOID ErrorExit(LPCTSTR);
    VOID generateSalt(uint8_t*, size_t);
    VOID argon2(uint8_t*, uint32_t, uint8_t*, size_t, uint8_t*);
    VOID crypt(PBYTE, SIZE_T, uint8_t*, uint8_t*, size_t, BOOL);
    VOID crypt(PBYTE, SIZE_T, uint8_t*, size_t, BOOL);
    int toBase64(PBYTE, int, PBYTE);
    int fromBase64(PBYTE, PBYTE);
    SIZE_T TextFromClipboard(HWND, PBYTE);
    VOID TextToClipboard(HWND, PBYTE, SIZE_T);
    VOID encryptClipboard(HWND);
    VOID decryptClipboard(HWND);
    BOOL CALLBACK TrySendMessage(HWND, LPARAM);
    VOID getAndEncryptMessage(HWND);
    VOID decryptMessage(HWND);
    VOID addNewUser(PBYTE, PBYTE);
    INT_PTR CALLBACK addUser(HWND, UINT, WPARAM, LPARAM);
    INT_PTR CALLBACK selectUser(HWND, UINT, WPARAM, LPARAM);
    INT_PTR CALLBACK getPassword(HWND, UINT, WPARAM, LPARAM);
    VOID saveFile();
    VOID loadFile();

    PBYTE rawText = NULL;
    int rawTextSize = 0;
    HWND hWnd;

    VOID ErrorExit(LPCTSTR lpszFunction) {
        LPVOID lpMsgBuf;
        LPVOID lpDisplayBuf;
        DWORD dw = GetLastError();

        FormatMessage(
            FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM |
            FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL,
            dw,
            MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
            (LPTSTR)& lpMsgBuf,
            0, NULL);

        lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
(lstrlen((LPCTSTR)lpMsgBuf) + lstrlen(lpszFunction) + 40) *
sizeof(TCHAR));
        wsprintf((LPWSTR)lpDisplayBuf, TEXT("%s failed with
error %d: %s"), lpszFunction, dw, lpMsgBuf);
        MessageBox(NULL, (LPCTSTR)lpDisplayBuf, TEXT("Error"),
MB_OK | MB_SERVICE_NOTIFICATION);

        LocalFree(lpMsgBuf);
        LocalFree(lpDisplayBuf);
        ExitProcess(dw);
    }
```

```
    VOID generateSalt(uint8_t* salt, size_t saltlen) {
        size_t n = strlen(basis_64);
        for (int i = 0; i < saltlen; i++) {
            salt[i] = basis_64[rand() % n];
        }
    }

    VOID argon2(uint8_t* pwd, uint32_t pwdlen, uint8_t* salt,
size_t saltlen, uint8_t* hash) {
        uint32_t t_cost = 2;            // 1-pass computation
        uint32_t m_cost = (1 << 16);    // 64 mebibytes memory
usage
        uint32_t parallelism = 1;       // number of threads and
lanes

        argon2i_hash_raw(t_cost, m_cost, parallelism, pwd,
pwdlen, salt, saltlen, hash, KEY_SIZE - 1);
    }

    VOID crypt(PBYTE text, SIZE_T textSize, uint8_t* key,
uint8_t* salt, size_t saltlen, BOOL isEncrypt) {
        const uint8_t* iv = (const uint8_t*)malloc(17);
        strcpy_s((char*)iv, 17, "Mapkn3InitVector");

        struct AES_ctx ctx;
        AES_init_ctx_iv(&ctx, key, iv);
        if (isEncrypt) {
            AES_CBC_encrypt_buffer(&ctx, text, textSize);
        }
        else {
            AES_CBC_decrypt_buffer(&ctx, text, textSize);
        }
    }

    VOID crypt(PBYTE text, SIZE_T textSize, uint8_t* salt,
size_t saltlen, BOOL isEncrypt) {
        uint8_t* key = (uint8_t*)malloc(KEY_SIZE);
        argon2((uint8_t*)users[currentUser].key, KEY_SIZE - 1,
salt, saltlen, key);
        crypt(text, textSize, key, salt, saltlen, isEncrypt);
    }

    int toBase64(PBYTE rawStr, int rawStrSize, PBYTE base64Str)
{
        int base64StrSize = Base64encode_len(rawStrSize);
        if (base64Str != NULL) {
            Base64encode((char*)base64Str, (char*)rawStr,
rawStrSize);
        }
        return base64StrSize;
    }
```

```
    int fromBase64(PBYTE base64Str, PBYTE rawStr) {
        int rawStrSize = Base64decode_len((const
char*)base64Str);
        if (rawStr != NULL) {
            rawStrSize = Base64decode((char*)rawStr, (const
char*)base64Str);
        }
        return rawStrSize;
    }

    SIZE_T TextFromClipboard(HWND hwnd, PBYTE buffer) {
        UINT nClipboardFormat = CF_TEXT;
        HANDLE pClipboardData = 0;
        PBYTE data = 0;
        SIZE_T dataSize = 0;

        if (IsClipboardFormatAvailable(nClipboardFormat)) {
            if (!OpenClipboard(hwnd)) {
                ErrorExit(TEXT("Open clipboard"));
            }

            pClipboardData = GetClipboardData(nClipboardFormat);
            if (!pClipboardData) {
                CloseClipboard();
                ErrorExit(TEXT("Get clipboard data"));
            }
            data = (PBYTE)GlobalLock(pClipboardData);
            dataSize = strlen((const char*)data);
            if (buffer != NULL) {
                for (int i = 0; i < dataSize; i++) {
                    buffer[i] = data[i];
                }
            }
            GlobalUnlock(pClipboardData);
            if (!CloseClipboard()) {
                ErrorExit(TEXT("Close clipboard"));
            }
        }
        return dataSize;
    }

    VOID TextToClipboard(HWND hwnd, PBYTE data, SIZE_T dataSize)
{
        UINT nClipboardFormat = CF_TEXT;
        HGLOBAL clipboardData = 0;
        PBYTE buffer = 0;

        if (!OpenClipboard(hwnd)) {
            ErrorExit(TEXT("Open clipboard"));
        }
        if (!EmptyClipboard()) {
            ErrorExit(TEXT("Empty clipboard"));
```

```
        }
        clipboardData = GlobalAlloc(GMEM_DDESHARE |
GMEM_MOVEABLE, dataSize);
        buffer = (PBYTE)GlobalLock(clipboardData);
        memset(buffer, 0, dataSize);
        for (int i = 0; i < dataSize; i++) {
            buffer[i] = data[i];
        }
        GlobalUnlock(clipboardData);
        SetClipboardData(nClipboardFormat, clipboardData);
        if (!CloseClipboard()) {
            ErrorExit(TEXT("Close clipboard"));
        }
    }

    VOID encryptClipboard(HWND hWnd) {
        PBYTE textFromClipboard = NULL;
        SIZE_T textFromClipboardSize = TextFromClipboard(hWnd,
textFromClipboard);
        textFromClipboard =
(PBYTE)malloc(textFromClipboardSize);
        memset(textFromClipboard, 0, textFromClipboardSize);
        TextFromClipboard(hWnd, textFromClipboard);

        if (textFromClipboard != NULL) {
            SIZE_T textSize = textFromClipboardSize;
            if (textFromClipboardSize % 16 != 0) {
                textSize += 16 - (textFromClipboardSize % 16);
            }
            PBYTE text = (PBYTE)malloc(textSize);
            memset(text, 0x00, textSize);
            for (int i = 0; i < textFromClipboardSize; i++) {
                text[i] = textFromClipboard[i];
            }

            uint8_t* salt = (uint8_t*)malloc(SALT_SIZE);
            memset(salt, 0x00, SALT_SIZE);
            generateSalt(salt, SALT_SIZE);

            crypt(text, textSize, salt, SALT_SIZE, TRUE);

            PBYTE base64 = NULL;
            int base64Size = toBase64(text, textSize, base64);
            base64 = (PBYTE)malloc(base64Size);
            memset(base64, 0, base64Size);
            toBase64(text, textSize, base64);

            PBYTE result = NULL;
            int resultSize = base64Size + SALT_SIZE;
            result = (PBYTE)malloc(resultSize);
            memset(result, 0, resultSize);
            int k = 0;
```

```
            for (int i = 0; i < SALT_SIZE; i++) {
                result[k++] = salt[i];
            }
            for (int i = 0; i < base64Size; i++) {
                result[k++] = base64[i];
            }
            TextToClipboard(hWnd, result, resultSize);
            free(text);
            free(result);
            free(textFromClipboard);
        }
        else {
            MessageBox(NULL, TEXT("Не найден текст в буфере
обмена"), TEXT("Информация"), MB_OK | MB_SERVICE_NOTIFICATION |
MB_ICONINFORMATION);
        }
    }

    VOID decryptClipboard(HWND hWnd) {
        PBYTE textFromClipboard = NULL;
        SIZE_T textFromClipboardSize = TextFromClipboard(hWnd,
textFromClipboard);
        textFromClipboard =
(PBYTE)malloc(textFromClipboardSize);
        memset(textFromClipboard, 0, textFromClipboardSize);
        TextFromClipboard(hWnd, textFromClipboard);

        if (textFromClipboard != NULL) {
            uint8_t* salt = (uint8_t*)malloc(SALT_SIZE);
            memset(salt, 0x00, SALT_SIZE);

            int base64Size = textFromClipboardSize - SALT_SIZE;
            PBYTE base64 = (PBYTE)malloc(base64Size);
            memset(base64, 0, base64Size);
            int k = 0;
            for (int i = 0; i < SALT_SIZE; i++) {
                salt[i] = textFromClipboard[k++];
            }
            for (int i = 0; i < base64Size; i++) {
                base64[i] = textFromClipboard[k++];
            }

            PBYTE text = NULL;
            int textSize = fromBase64(base64, text);
            text = (PBYTE)malloc(textSize);
            memset(text, 0, textSize);
            textSize = fromBase64(base64, text);
            text = (PBYTE)malloc(textSize);
            memset(text, 0, textSize);
            fromBase64(base64, text);

            crypt(text, textSize, salt, SALT_SIZE, FALSE);
```

```
            rawTextSize = textSize;
            rawText = (PBYTE)malloc(rawTextSize);
            memset(rawText, 0x00, rawTextSize);
            for (int i = 0; i < rawTextSize; i++) {
                rawText[i] = text[i];
            }

            free(text);
            free(textFromClipboard);
        }
    }

    BOOL CALLBACK TrySendMessage(HWND hwnd, LPARAM msg) {
        SendMessage(hwnd, msg, 0, 0);
        return TRUE;
    }

    VOID getAndEncryptMessage(HWND hForeground) {
        PBYTE backUpClipboard = NULL;
        SIZE_T backUpClipboardSize =
TextFromClipboard(hForeground, backUpClipboard);
        backUpClipboard = (PBYTE)malloc(backUpClipboardSize);
        memset(backUpClipboard, 0, backUpClipboardSize);
        TextFromClipboard(hForeground, backUpClipboard);

        EnumChildWindows(hForeground, TrySendMessage, WM_CUT);

        PBYTE textFromClipboard = NULL;
        SIZE_T textFromClipboardSize =
TextFromClipboard(hForeground, textFromClipboard);
        textFromClipboard =
(PBYTE)malloc(textFromClipboardSize);
        memset(textFromClipboard, 0, textFromClipboardSize);
        TextFromClipboard(hForeground, textFromClipboard);
        if (strcmp((const char*)backUpClipboard, (const
char*)textFromClipboard) == 0) {
            MessageBox(NULL, TEXT("Выбранный текст совпадает с
текстом в буфере обмена"), TEXT("Ошибка"), MB_OK |
MB_SERVICE_NOTIFICATION | MB_ICONERROR);
            return;
        }

        encryptClipboard(hForeground);
        EnumChildWindows(hForeground, TrySendMessage, WM_PASTE);
        free(backUpClipboard);
    }

    VOID decryptMessage(HWND hForeground) {
        PBYTE backUpClipboard = NULL;
        SIZE_T backUpClipboardSize =
TextFromClipboard(hForeground, backUpClipboard);
```

```
        backUpClipboard = (PBYTE)malloc(backUpClipboardSize);
        memset(backUpClipboard, 0, backUpClipboardSize);
        TextFromClipboard(hForeground, backUpClipboard);

        EnumChildWindows(hForeground, TrySendMessage, WM_COPY);

        PBYTE textFromClipboard = NULL;
        SIZE_T textFromClipboardSize =
TextFromClipboard(hForeground, textFromClipboard);
        textFromClipboard =
(PBYTE)malloc(textFromClipboardSize);
        memset(textFromClipboard, 0, textFromClipboardSize);
        TextFromClipboard(hForeground, textFromClipboard);
        if (strcmp((const char*)backUpClipboard, (const
char*)textFromClipboard) == 0) {
            MessageBox(NULL, TEXT("Выбранный текст совпадает с
текстом в буфере обмена"), TEXT("Ошибка"), MB_OK |
MB_SERVICE_NOTIFICATION | MB_ICONERROR);
            return;
        }
        decryptClipboard(hForeground);
        TextToClipboard(hForeground, backUpClipboard,
backUpClipboardSize);
        free(backUpClipboard);
    }

    VOID addNewUser(PBYTE name, PBYTE key) {
        if (userCount <= MAX_USERS) {
            userCount += 1;
            currentUser += 1;
            strcpy_s((char*)users[currentUser].name, NAME_SIZE,
(char*)name);
            strcpy_s((char*)users[currentUser].key, KEY_SIZE,
(char*)key);
        }
        InvalidateRect(hWnd, NULL, TRUE);
     }



    int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                          _In_opt_ HINSTANCE hPrevInstance,
                          _In_ LPWSTR    lpCmdLine,
                          _In_ int       nCmdShow)
    {
        UNREFERENCED_PARAMETER(hPrevInstance);
        UNREFERENCED_PARAMETER(lpCmdLine);

        password = (uint8_t*)malloc(KEY_SIZE);
        memset(password, 0x00, KEY_SIZE);
        DialogBox(hInst, MAKEINTRESOURCE(IDD_PASSWORD), hWnd,
getPassword);
```

```
        if (strlen((const char*)password) == 0) {
            return 0;
        }
        loadFile();

        // Инициализация глобальных строк
        LoadStringW(hInstance, IDS_APP_TITLE, szTitle,
MAX_LOADSTRING);
        LoadStringW(hInstance,
IDC_SYSTEMOFTRANSPARENTENCRYPTION, szWindowClass,
MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Выполнить инициализацию приложения:
        if (!InitInstance (hInstance, nCmdShow))
        {
            return FALSE;
        }

        HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_SYSTEMOFTRANSPARENTENCRYPTION));

        MSG msg;

        srand(time(NULL));
        if (!RegisterHotKey(NULL, 1, MOD_ALT, 'E')) {
            ErrorExit(TEXT("E: "));
        }
        if (!RegisterHotKey(NULL, 2, MOD_ALT, 'D')) {
            ErrorExit(TEXT("D: "));
        }
        if (!RegisterHotKey(NULL, 3, MOD_ALT, 'N')) {
            ErrorExit(TEXT("N: "));
        }
        if (!RegisterHotKey(NULL, 4, MOD_ALT, 'C')) {
            ErrorExit(TEXT("C: "));
        }
        if (!RegisterHotKey(NULL, 11, MOD_ALT | MOD_SHIFT, 'E'))
{
            ErrorExit(TEXT("only E: "));
        }
        if (!RegisterHotKey(NULL, 12, MOD_ALT | MOD_SHIFT, 'D'))
{
            ErrorExit(TEXT("only  D: "));
        }
        // Цикл основного сообщения:
        while (GetMessage(&msg, nullptr, 0, 0))
        {
            if (msg.message == WM_HOTKEY) {
                if (userCount == 0 && (msg.wParam == 1 ||
msg.wParam == 2)) {
```

```
                    MessageBox(NULL, TEXT("Для начала добавьте
собеседника"), TEXT("Предупреждение"), MB_OK |
MB_SERVICE_NOTIFICATION | MB_ICONWARNING);
                } else {
                    switch (msg.wParam) {
                        case 1:

getAndEncryptMessage(GetForegroundWindow());
                            break;
                        case 2:

decryptMessage(GetForegroundWindow());
                            InvalidateRect(hWnd, NULL, TRUE);
                            break;
                        case 3:
                            DialogBox(hInst,
MAKEINTRESOURCE(IDD_ADD_USER), hWnd, addUser);
                            break;
                        case 4:
                            DialogBox(hInst,
MAKEINTRESOURCE(IDD_SELECT_USER), hWnd, selectUser);
                            break;
                        case 11:

encryptClipboard(GetForegroundWindow());
                            MessageBox(NULL, TEXT("Текст в
буфере обмена зашифрован"), TEXT("Информация"), MB_OK |
MB_SERVICE_NOTIFICATION | MB_ICONINFORMATION);
                            break;
                        case 12:

decryptClipboard(GetForegroundWindow());
                            InvalidateRect(hWnd, NULL, TRUE);
                            MessageBox(NULL, TEXT("Текст в
буфере обмена дешифрован"), TEXT("Информация"), MB_OK |
MB_SERVICE_NOTIFICATION | MB_ICONINFORMATION);
                            break;
                    }
                }
            }

            if (!TranslateAccelerator(msg.hwnd, hAccelTable,
&msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }

        if (!UnregisterHotKey(NULL, 12)) {
            ErrorExit(TEXT("Un only D: "));
        }
```

```
        if (!UnregisterHotKey(NULL, 11)) {
            ErrorExit(TEXT("Un only E: "));
        }
        if (!UnregisterHotKey(NULL, 4)) {
            ErrorExit(TEXT("Un C: "));
        }
        if (!UnregisterHotKey(NULL, 3)) {
            ErrorExit(TEXT("Un N: "));
        }
        if (!UnregisterHotKey(NULL, 2)) {
            ErrorExit(TEXT("Un D: "));
        }
        if (!UnregisterHotKey(NULL, 1)) {
            ErrorExit(TEXT("Un E: "));
        }
        return (int) msg.wParam;
    }



    //
    //  ФУНКЦИЯ: MyRegisterClass()
    //
    //  ЦЕЛЬ: Регистрирует класс окна.
    //
    ATOM MyRegisterClass(HINSTANCE hInstance)
    {
        WNDCLASSEXW wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style          = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc    = WndProc;
        wcex.cbClsExtra     = 0;
        wcex.cbWndExtra     = 0;
        wcex.hInstance      = hInstance;
        wcex.hIcon          = LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_SYSTEMOFTRANSPARENTENCRYPTION));
        wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
        wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName   =
MAKEINTRESOURCEW(IDC_SYSTEMOFTRANSPARENTENCRYPTION);
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));

        return RegisterClassExW(&wcex);
    }

    //
    //   ФУНКЦИЯ: InitInstance(HINSTANCE, int)
    //
```

```
     //   ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное
окно
     //
     //   КОММЕНТАРИИ:
     //
     //        В этой функции маркер экземпляра сохраняется в
глобальной переменной, а также
     //        создается и выводится главное окно программы.
     //
     BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
     {
         hInst = hInstance; // Сохранить маркер экземпляра в
глобальной переменной

         hWnd = CreateWindowW(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, 900, 500, nullptr,
nullptr, hInstance, nullptr);

         if (!hWnd)
         {
            return FALSE;
         }

         ShowWindow(hWnd, nCmdShow);
         UpdateWindow(hWnd);

         return TRUE;
     }

     //
     //  ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)
     //
     //  ЦЕЛЬ: Обрабатывает сообщения в главном окне.
     //
     //  WM_COMMAND  - обработать меню приложения
     //  WM_PAINT    - Отрисовка главного окна
     //  WM_DESTROY  - отправить сообщение о выходе и вернуться
     //
     //
     LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
     {
         switch (message)
         {
         case WM_COMMAND:
             {
                 int wmId = LOWORD(wParam);
                 // Разобрать выбор в меню:
                 switch (wmId)
                 {
                 case IDM_ADD_USER:
```

```
                    DialogBox(hInst,
MAKEINTRESOURCE(IDD_ADD_USER), hWnd, addUser);
                    break;
                case IDM_SELECT_USER:
                    DialogBox(hInst,
MAKEINTRESOURCE(IDD_SELECT_USER), hWnd, selectUser);
                    break;
                case IDM_ABOUT:
                    DialogBox(hInst,
MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam,
lParam);
                }
            }
            break;
        case WM_PAINT:
            {
                PBYTE user = NULL;
                size_t userLen = 0;
                if (userCount > 0) {
                    const char* title = "Текущий собеседник: ";
                    int titleLen = strlen(title);
                     int nameLen = 0;
                    for (; users[currentUser].name[nameLen] !=
0x00; nameLen++) {}
                    userLen = titleLen + nameLen;
                    user = (PBYTE)malloc(userLen);
                    memset(user, 0x00, userLen);

                    for (int i = 0; i < titleLen; i++) {
                        user[i] = title[i];
                    }
                    for (int i = 0; i < nameLen; i++) {
                        user[titleLen + i] =
users[currentUser].name[i];
                    }
                }
                PAINTSTRUCT ps;
                HDC hdc = BeginPaint(hWnd, &ps);
                if (user != NULL) {
                    TextOutA(hdc, 10, 10, (LPCSTR)user,
userLen);
                }
                TextOutA(hdc, 10, 40, (LPCSTR)rawText,
rawTextSize);
                EndPaint(hWnd, &ps);
                free(user);
```

```
            }
            break;
        case WM_DESTROY:
            saveFile();
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
    }

    // Обработчик сообщений для окна "О программе".
    INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
            return (INT_PTR)TRUE;

        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
        }
        return (INT_PTR)FALSE;
    }

    INT_PTR CALLBACK addUser(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        PCHAR lpszKey = (PCHAR)malloc(KEY_SIZE);
        memset(lpszKey, 0x00, KEY_SIZE);
        BYTE cchKey;

        PCHAR lpszName = (PCHAR)malloc(NAME_SIZE);
        memset(lpszName, 0x00, NAME_SIZE);
        BYTE cchName;

        switch (message)
        {
        case WM_INITDIALOG:
            // Set the default push button to "Cancel."
            SendMessage(hDlg,
                DM_SETDEFID,
                (WPARAM)IDCANCEL,
```

```
                (LPARAM)0);

            return (INT_PTR)TRUE;

        case WM_COMMAND:
            // Set the default push button to "OK" when the user
enters text.
            if (HIWORD(wParam) == EN_CHANGE &&
                LOWORD(wParam) == IDC_KEY_EDIT)
            {
                SendMessage(hDlg,
                    DM_SETDEFID,
                    (WPARAM)IDOK,
                    (LPARAM)0);
            }
            switch (wParam)
            {
            case IDOK:
                // Get number of characters.
                cchKey = (BYTE)SendDlgItemMessage(hDlg,
                    IDC_KEY_EDIT,
                    EM_LINELENGTH,
                    (WPARAM)0,
                    (LPARAM)0);
                if (cchKey != KEY_SIZE - 1)
                {
                    MessageBox(hDlg,
                        L"Длина пароля должна быть 16 символов",
                        L"Ошибка",
                        MB_OK | MB_ICONERROR);

                    //EndDialog(hDlg, TRUE);
                    return (INT_PTR)FALSE;
                }

                // Put the number of characters into first word
of buffer.
                *((LPBYTE)lpszKey) = cchKey;

                // Get the characters.
                SendDlgItemMessageA(hDlg,
                    IDC_KEY_EDIT,
                    EM_GETLINE,
                    (WPARAM)0,
                    (LPARAM)lpszKey);

                // Get number of characters.
                cchName = (BYTE)SendDlgItemMessage(hDlg,
                    IDC_NAME_EDIT,
                    EM_LINELENGTH,
                    (WPARAM)0,
                    (LPARAM)0);
```

```
                if (cchName > NAME_SIZE - 1)
                {
                    MessageBox(hDlg,
                        L"Длина имени должна быть 20 символов",
                        L"Ошибка",
                        MB_OK | MB_ICONERROR);

                    //EndDialog(hDlg, TRUE);
                    return (INT_PTR)FALSE;
                }

                // Put the number of characters into first word
of buffer.
                *((LPBYTE)lpszName) = cchName;

                // Get the characters.
                SendDlgItemMessageA(hDlg,
                    IDC_NAME_EDIT,
                    EM_GETLINE,
                    (WPARAM)0,
                    (LPARAM)lpszName);

                addNewUser((PBYTE)lpszName, (PBYTE)lpszKey);

                EndDialog(hDlg, TRUE);
                return (INT_PTR)TRUE;

            case IDCANCEL:
                EndDialog(hDlg, TRUE);
                return (INT_PTR)TRUE;
            }
            return 0;
        }
        return (INT_PTR)FALSE;

        UNREFERENCED_PARAMETER(lParam);
    }

    INT_PTR CALLBACK selectUser(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        switch (message)
        {
        case WM_INITDIALOG:
            BYTE name[NAME_SIZE];
            for (int i = 0; i < userCount; i++) {
                memset(name, 0x00, NAME_SIZE);
                strcpy_s((char*)name, NAME_SIZE,
(char*)users[i].name);
                SendDlgItemMessageA(hDlg, IDC_USER_COMBO,
CB_ADDSTRING, 0, (LPARAM)name);
            }
```

```
            SendDlgItemMessageA(hDlg, IDC_USER_COMBO,
CB_SETDROPPEDWIDTH, 0, (LPARAM)name);
            SendDlgItemMessage(hDlg, IDC_USER_COMBO,
CB_SETCURSEL, currentUser, 0);
            // Set the default push button to "Cancel."
            SendMessage(hDlg,
                DM_SETDEFID,
                (WPARAM)IDCANCEL,
                (LPARAM)0);

            return (INT_PTR)TRUE;

        case WM_COMMAND:
            // Set the default push button to "OK" when the user
enters text.
            if (HIWORD(wParam) == CBN_SELCHANGE)
            {
                SendMessage(hDlg,
                    DM_SETDEFID,
                    (WPARAM)IDOK,
                    (LPARAM)0);
            }
            switch (wParam)
            {
            case IDOK:
                currentUser = SendDlgItemMessageA(hDlg,
IDC_USER_COMBO, CB_GETCURSEL, 0, 0);
                InvalidateRect(hWnd, NULL, TRUE);

                EndDialog(hDlg, TRUE);
                return (INT_PTR)TRUE;

            case IDCANCEL:
                EndDialog(hDlg, TRUE);
                return (INT_PTR)TRUE;
            }
            return 0;
        }
        return (INT_PTR)FALSE;

        UNREFERENCED_PARAMETER(lParam);
    }

    INT_PTR CALLBACK getPassword(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        PCHAR lpszPassword = (PCHAR)malloc(KEY_SIZE);
        memset(lpszPassword, 0x00, KEY_SIZE);
        BYTE cchPassword;

        switch (message)
        {
```

```
        case WM_INITDIALOG:
            // Set the default push button to "Cancel."
            SendMessage(hDlg,
                DM_SETDEFID,
                (WPARAM)IDCANCEL,
                (LPARAM)0);

            return (INT_PTR)TRUE;

        case WM_COMMAND:
            // Set the default push button to "OK" when the user
enters text.
            if (HIWORD(wParam) == EN_CHANGE &&
                LOWORD(wParam) == IDC_PASSWORD_EDIT)
            {
                SendMessage(hDlg,
                    DM_SETDEFID,
                    (WPARAM)IDOK,
                    (LPARAM)0);
            }
            switch (wParam)
            {
            case IDOK:
                // Get number of characters.
                cchPassword = (BYTE)SendDlgItemMessage(hDlg,
                    IDC_PASSWORD_EDIT,
                    EM_LINELENGTH,
                    (WPARAM)0,
                    (LPARAM)0);
                if (cchPassword != KEY_SIZE - 1)
                {
                    MessageBox(hDlg,
                        L"Длина пароля должна быть 16 символов",
                        L"Ошибка",
                        MB_OK | MB_ICONERROR);

                    //EndDialog(hDlg, TRUE);
                    return (INT_PTR)FALSE;
                }

                // Put the number of characters into first word
of buffer.
                *((LPBYTE)lpszPassword) = cchPassword;

                // Get the characters.
                SendDlgItemMessageA(hDlg,
                    IDC_PASSWORD_EDIT,
                    EM_GETLINE,
                    (WPARAM)0,
                    (LPARAM)lpszPassword);

                for (int i = 0; i < KEY_SIZE; i++) {
```

```
                    password[i] = lpszPassword[i];
                }

                EndDialog(hDlg, TRUE);
                return (INT_PTR)TRUE;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            }
            return 0;
        }
        return (INT_PTR)FALSE;

        UNREFERENCED_PARAMETER(lParam);
    }

    VOID saveFile() {
        FILE* f;
        fopen_s(&f, BACKUP_FILENAME, "w");
        if (!f) {
            MessageBox(NULL, L"Не найден файл собеседников",
L"Внимание", MB_OK | MB_ICONWARNING);
            return;
        }
        for (int i = 0; i < userCount; i++) {
            char* line = (char*)malloc(KEY_SIZE + NAME_SIZE -
1);
            memset(line, 0x00, KEY_SIZE + NAME_SIZE - 1);
            int j = 0;
            for (; j < KEY_SIZE - 1; j++) {
                line[j] = users[i].key[j];
            }
            for (; j < KEY_SIZE + NAME_SIZE - 1 &&
users[i].name[j - KEY_SIZE + 1] != '\0'; j++) {
                line[j] = users[i].name[j - KEY_SIZE + 1];
            }
            line[j++] = '\n';

            SIZE_T textSize = j;
            if (j % 16 != 0) {
                textSize += 16 - (j % 16);
            }
            PBYTE text = (PBYTE)malloc(textSize);
            memset(text, 0x00, textSize);
            for (int i = 0; i < j; i++) {
                text[i] = line[i];
            }

            uint8_t* salt = (uint8_t*)malloc(SALT_SIZE);
            memset(salt, 0x00, SALT_SIZE);
            generateSalt(salt, SALT_SIZE);
```

```
            crypt(text, textSize, password, salt, SALT_SIZE,
TRUE);

            PBYTE base64 = NULL;
            int base64Size = toBase64(text, textSize, base64);
            base64 = (PBYTE)malloc(base64Size);
            memset(base64, 0x00, base64Size);
            toBase64(text, textSize, base64);

            PBYTE result = NULL;
            int resultSize = base64Size + SALT_SIZE + 1;
            result = (PBYTE)malloc(resultSize);
            memset(result, 0x00, resultSize);
            int k = 0;
            for (int i = 0; i < SALT_SIZE; i++) {
                result[k++] = salt[i];
            }
            for (int i = 0; i < base64Size; i++) {
                result[k++] = base64[i];
            }
            fprintf(f, "%s\n", result);
        }
        fclose(f);
    }

    VOID loadFile() {
        PBYTE key = (PBYTE)malloc(KEY_SIZE);
        PBYTE name = (PBYTE)malloc(NAME_SIZE);
        char* line = (char*)malloc(53);
        PBYTE user;
        int userSize;
        memset(line, 0x00, 53);
        FILE* f;
        fopen_s(&f, BACKUP_FILENAME, "r");
        if (!f) {
            MessageBox(NULL, L"Не найден файл собеседников",
L"Внимание", MB_OK | MB_ICONWARNING);
            return;
        }
        while (fgets(line, 53, f)) {
            if (strlen(line) == 0) {
                break;
            }
            uint8_t* salt = (uint8_t*)malloc(SALT_SIZE);
            memset(salt, 0x00, SALT_SIZE);

            int base64Size = 53 - SALT_SIZE;
            PBYTE base64 = (PBYTE)malloc(base64Size);
            memset(base64, 0x00, base64Size);
            int k = 0;
            for (int i = 0; i < SALT_SIZE; i++) {
                salt[i] = line[k++];
```

```
            }
            for (int i = 0; i < base64Size; i++) {
                base64[i] = line[k++];
            }

            PBYTE text = NULL;
            int textSize = fromBase64(base64, text);
            text = (PBYTE)malloc(textSize);
            memset(text, 0x00, textSize);
            textSize = fromBase64(base64, text);
            text = (PBYTE)malloc(textSize);
            memset(text, 0x00, textSize);
            fromBase64(base64, text);

            crypt(text, textSize, password, salt, SALT_SIZE,
FALSE);

            userSize = textSize;
            user = (PBYTE)malloc(userSize);
            memset(user, 0x00, userSize);
            for (int i = 0; i < userSize; i++) {
                user[i] = text[i];
            }

            currentUser++;
            memset(key, 0x00, KEY_SIZE);
            memset(name, 0x00, NAME_SIZE);
            int j = 0;
            for (; j < KEY_SIZE - 1; j++) {
                users[currentUser].key[j] = user[j];
            }
            for (; j < KEY_SIZE + NAME_SIZE - 1 && user[j] !=
'\n'; j++) {
                users[currentUser].name[j - KEY_SIZE + 1] =
user[j];
            }
            userCount++;
            memset(line, 0x00, 53);
        }
        fclose(f);
    }
```