Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Информатика
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С.Кузнецов
подпись инициалы, фамилия
«____» _____ 2017 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Модель и программный комплекс системы безопасного обмена
информацией в компьютерных сетях общего доступа
тема

09.04.04 «Программная инженерия»
код и наименования направления

09.04.04.01 «Программное обеспечение вычислительной техники
и автоматизированных систем»
код и наименования магистерской программы

Научный руководитель	_____	рук. НУЛ ИБ, к.т.н	А.Н.Шниперов
	подпись, дата	должность, ученная степень	инициалы, фамилия
Выпускник	_____		А.П.Чистяков
	подпись, дата		инициалы, фамилия
Рецензент	_____	рук. ИнТК СФУ, к.т.н	И.А.Карлов
	подпись, дата	должность, ученная степень	инициалы, фамилия

Красноярск 2017

АННОТАЦИЯ

Магистерская диссертация по теме «Модель и программный комплекс системы безопасного обмена информацией в компьютерных сетях общего доступа» содержит 88 страниц текстового документа, 33 иллюстраций, 21 таблиц, 7 формул, 56 использованных источников, и 4 приложений.

Представленная работа посвящена проблемам защиты конфиденциальной информации в открытых компьютерных сетях от несанкционированного доступа. В работе исследуются современные системы мгновенного информационного обмена, рассматриваются их недостатки в контексте информационной безопасности. С целью устранения выявленных недостатков, предлагается новый способ безопасного информационного обмена в открытых компьютерных сетях. На основе которого, разработана модель и программный комплекс. В работе отражены технологические аспекты разработки и надёжности программного комплекса.

ЗАЩИТА ИНФОРМАЦИИ, ЗАЩИТА ТЕЛЕКОММУНИКАЦИЙ, ОКОНЕЧНОЕ ШИФРОВАНИЕ, РАСПРЕДЕЛЕНИЕ КЛЮЧЕЙ ШИФРОВАНИЯ, АУТЕНТИФИКАЦИЯ ИНФОРМАЦИИ, КРИПТОСИСТЕМА.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность. Вопросы, связанные с обеспечением защиты передаваемой информации по открытым компьютерным сетям общего доступа, не теряют своей актуальности и по сей день, несмотря на многочисленные предложенные способы и попытки их решений как со стороны индивидуальных разработчиков, так и со стороны частных и государственных компаний. Особенно заметно это стало с интенсивным ростом популярности использования программных систем мгновенного информационного обмена сообщениями (*Instant messaging*, *IM*) – мессенджеры, а также повсеместным распространением абонентских устройств подключения к сети Интернет, и практическим применением информационно-коммуникационных технологий (ИКТ) в жизни современного общества, в том числе и для коммуникаций. Сегодня *IM* системы используются уже не только в личных целях, они также внедряются в бизнес среду, государственные компании и учреждения как основное средство для быстрой и дешёвой коммуникации между сотрудниками на расстоянии, о чем свидетельствуют последние новостные события. По собственным подсчетам, некоторых *IM* систем, число зарегистрированных пользователей достигло величины в 1 миллиард человек.

Следствием этого, *IM* системы хранят и обрабатывают огромное количество информации, среди которой может быть информация, предоставляющая экономическую, политическую, и иную ценность. Очевидно, что информация подобного рода, всегда привлекает злоумышленников, и требует повышенного уровня защиты. Практически все распространённые *IM* системы построены на централизованной архитектуре и несмотря на серьёзные механизмы защиты информации, применяемые разработчиками, имеют общий и существенный недостаток, – отсутствие доверия к центральной части (сервис-провайдеру) *IM* систем. При этом центральная сторона может быть скомпрометирована как злоумышленниками, так и владельцем системы по тем или иным причинам. Другими словами, может быть реализована атака «человек в середине» (*Man-in-the-middel*, *MITM*), которая в свою очередь несет ряд угроз нарушения конфиденциальности и целостности информации пользователей в процессе информационного обмена. Наличие таких угроз, особенно в контексте того, что большая часть распространённых *IM* систем принадлежит иностранным компаниям, существенно ограничивает или делает невозможным их использование для обмена конфиденциальной информацией.

Таким образом формализовалась научно-техническая задача по организации безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа, с учётом возможной компрометации центральной стороны

информационной системы. В свою очередь решение данной задачи влечёт за собой разработку способа безопасного обмена, модели информационной системы и различного алгоритмического обеспечения.

Основная цель и задачи работы. Целью настоящей работы заключается в разработке информационной системы для безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа.

В ходе выполнения работы были поставлены и решены следующие основные задачи:

- изучить и проанализировать техническую документацию современных и наиболее распространённых систем мгновенного обмена информацией на предмет обнаружения возможных критических угроз безопасности;
- разработать новый способ безопасного обмена информацией, нивелирующий обнаруженные критические угрозы;
- разработать модель информационной системы для безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа;
- разработать сопутствующее алгоритмическое обеспечение;
- осуществить программную реализацию разработанной информационной системы;
- оценить качественные и количественные показатели предлагаемой программной реализации информационной системы, в том числе нагрузочную способность;
- предложить масштабируемую архитектуру информационной системы, учитывающую потенциальный рост количества абонентов.

Объект исследования. Безопасный обмен конфиденциальной информацией в открытых компьютерных сетях общего доступа.

Предмет исследования. Системы мгновенного обмена информацией с повышенным уровнем защищенности.

Методы исследования. Для решения поставленных задач использовались методы защиты информации, теория криптографии, теория сетей и сетевых протоколов, теория множеств, методы построения программных комплексов, методы построения распределённых информационных систем.

Научная новизна. Новыми являются следующие результаты работы:

- новая модель информационной системы для обмена конфиденциальной информацией в открытых компьютерных сетях, отличительная особенность которой заключается в устранении угрозы

типа «человек посередине» в условиях компрометации серверной стороны;
- алгоритмы, реализующие инфраструктуру ключей шифрования по нескольким каналам и протоколам связи с целью разделения общего секрета.

Практическая значимость результатов работы:
- разработан программный комплекс, реализующий предлагаемую информационную систему;
- предложена масштабируемая архитектура информационной системы, учитывающая потенциальный рост количества абонентов.

Апробация. Результаты работы докладывались и обсуждались на следующих научно-технических конференциях: Международная научно-техническая конференция студентов, аспирантов, и молодых ученых «Научная сессия ТУСУР», г. Томск (2016, 2017 гг.), доклады отмечены II и III местом соответственно; Всероссийская молодежная школа-семинар по проблемам информационной безопасности «ПЕРСПЕКТИВА-2016», г. Таганрог (2016 г.).

Публикации. По теме диссертации опубликовано 3 печатных работ в сборниках тезисов конференций, основные положения и результаты работы приняты к публикации в научно-техническом журнале «Программная инженерия» (перечень ВАК), получено авторское свидетельство о государственной регистрации программы для ЭВМ на разработанный программный комплекс.

Структура и объем работы. Диссертационная работы состоит из введения, 4 глав, заключения, списка литературы (56 наименований) и 4 приложений. Основной текст содержит 88 страниц, включающий в себя 33 рисунков, 21 таблиц, и 7 формул.

Содержание работы

Во *введении* описывается общая характеристика работы, обосновывается актуальность диссертационной работы, определяется цель, формулируются задачи исследования, описывается структура и содержание работы.

Первая глава посвящена обзору технической документации, используемых криптографических протоколов и алгоритмов в наиболее распространенных аналогичных системах обмена информации, с целью выявления потенциальных угроз безопасности. В частности, рассматриваются системы *WhatsApp*, *Viber*, *Telegram*, *Threema*. Формулируется постановка задачи.

Вторая глава рассматривает предлагаемый способ для обмена конфиденциальной информацией в компьютерных сетях общего доступа, учитывающий возможную компрометацию центральной части, и, практически, нивелирующий угрозу конфиденциальности передаваемой информации. В данной главе описывается решение для обеспечения защиты информации от

неправомерного доступа. Приводится модель разработанной информационной системы. Детально описываются используемые протоколы, инфраструктура ключей, подсистема управления доступом.

Третья глава детально описывает архитектурные решения программного комплекса «*ruMessanger*». Рассматриваются используемые инструменты для разработки, описывается подход к проектированию и реализации программного комплекса. Дается описание подсистем, модулей, протоколов взаимодействия частей программного комплекса, приводится архитектура базы данных. Также в данной главе приводятся результаты и методики испытаний программного продукта, нацеленные на обеспечение качества. Описываются средства и порядок тестирования, сценарий и процесс тестирования.

Четвертая глава посвящена вопросам масштабирования архитектуры разработанного программного комплекса. Предлагается способ по увеличению производительности серверной стороны программного комплекса.

Глава 1. Обзор и анализ существующих решений

Коммуникация между людьми была, есть и будет одной из важнейших составляющих деятельности человека. Так в 21 веке для коммуникации на расстоянии большинство людей использует информационно-коммуникационные технологии (ИКТ) и сеть Интернет, ставшие неотъемлемой частью жизни современного общества, и основой для развития его потенциала. То есть, люди пользуются электронной почтой (*email*) и системами мгновенного обмена сообщениями (*Instant messaging*, *IM*) – мессенджеры. В настоящее время, электронная почта в области личной коммуникации отошла на второй план, и используется в основном для формальной переписки, не требующей быстрого ответа. Возможность обмена сообщениями в реальном времени, позволила *IM* системам занять первое место по числу пользователей и объемах передаваемой информацией. Так, например, только приложением-мессенджером *WhatsApp* пользуются более 1 миллиарда человек в 180 странах [1], а в день передается около 42 миллиардов сообщений [2]. Следствием этого, особо актуальной становиться задача по обеспечению безопасного обмена информацией.

Рассматривая современные *IM* системы в разрезе обеспечения конфиденциальности передаваемой информации, можно особо выделить такие системы, как *WhatsApp*, *Viber*, *Telegram*, *Threema*, за их наибольшую популярность и серьёзные механизмы защиты информации. Перед тем как остановиться на подробном рассмотрении этих систем. Заметим, что представленные системы, и подавляющее большинство других подобных систем, построены на клиент-серверной архитектуре. Потому, что эта архитектура повышает управляемость *IM* системы в целом, обладает более высоким уровнем защищённости данных [3], обладает возможностью масштабирования, а также увеличивает надежность хранения обрабатываемых и резервных данных, за счёт передачи ответственности за сохранность данных на серверную сторону, что позволяет снизить пользовательские затраты на обслуживание.

1.1 Система обмена сообщениями *WhatsApp*

Система *WhatsApp* позволяет людям обмениваться сообщениями (включая текстовые сообщения, передавать изображения, видео, голосовые сообщения, файлы) в режиме реального времени и осуществлять звонки по всему миру. Она была основана Яном Кумом и Брайаном Эктоном в США, и представлена в 2009 году. На сегодняшний день *WhatsApp* принадлежит *Facebook Inc.* и занимает лидирующее место по количеству пользователей среди *IM* систем. В 2016 году

разработчики внедрили в *WhatsApp* технологию оконечного шифрования (*End-to-end encryption*, *E2EE*) [4] для всех типов клиентских приложений. По заявлению разработчиков, сервера *WhatsApp* хранят только не принятые сообщения абонентов, а остальные сообщения хранятся в памяти оконечного устройства. Рассмотрим наиболее важные этапы работы *WhatsApp* с позиции информационной безопасности.

1.1.1 Регистрация пользователя и создания личных ключей

При установки клиентского приложения *WhatsApp* для мобильного устройства, осуществляется аутентификация пользователя с помощью смс-кода. Далее, генерируются три ассиметричных ключа шифрования на эллиптической кривой *Curve*25519 [5]. Генерируемые ассиметричные ключи, приведены в таблице 1, используются для установления сеанса связи между пользователями (абонентами) системы *WhatsApp*.

Таблица 1 – Ассиметричные ключи пользователя

Обозначение	Описание
Identity Key = (IK_{pk}, IK_{sk})	Долговременная, идентификационная ключевая пара пользователя.
Signed Pre Key = (SK_{pk}, SK_{sk})	Среднесрочная, подписывающая ключевая пара пользователя.
One-Time Pre Keys = (OK_{pk}, OK_{sk})	Список из разовых, кратковременных, ключей пользователя.

На этапе регистрации пользователя в системе *WhatsApp*, клиентское приложение осуществляет передачу открытых ключей: IK_{pk}, SK_{pk}, OK_{pk} – серверной части. Как отмечается в [6], сервер *WhatsApp* хранит только открытые ключи пользователей, и ни имеет доступа к закрытым ключам.

1.1.2 Создание защищенного соединения

Конфиденциальность передаваемой информации в системе *WhatsApp* между клиентским и серверным приложениями базируется на протоколе *Noise Pipes* [7], включающий в себя протокол Диффи-Хеллмана на эллиптических кривых (*Elliptic Curve Diffie-Hellman*, *ECDH*) для выработки ключей шифрования, алгоритме шифрования *AES* в режиме счетчика с аутентификацией Галуа (*Galois Counter Mode*, *GCM*) [8], и хэш-функции *SHA*-256 [9].

Для установления безопасного соединения между пользователями используется протокол *Signal Protocol* [10], разработанный организацией *Open Whisper Systems*. Протокол позволяет организовать между пользователями *E2EE*, путем выработки общего долгосрочного секретного ключа между пользователями – мастер-ключа. Процесс создания защищенного сеанса связи [6] между

отправителем и получателем (*S* и *R* соответственно), т.е. мастер-ключа, можно описать следующим алгоритмом:

Шаг 1: Отправитель запрашивает от сервера открытые ключи получателя, т.е. IK_{pk}^{R}, SK_{pk}^{R}, OK_{pk}^{R};

Шаг 2: Сервер возвращает открытые ключи. Открытый ключ OK_{pk}^{R} используется только один раз, и после запроса, он удаляется из хранилища сервера. Если получатель не пополнил пустой список разовых открытых ключей *One-Time Pre Keys*, то открытый ключ OK_{pk}^{R} не возвращается;

Шаг 3: Отправитель сохраняет открытые ключи получателя;

Шаг 4: Отправитель генерирует на *Curve*25519 эфемерную ключевую пару, т.е. $(EK_{pk}^{S}, EK_{sk}^{S})$;

Шаг 5: Отправитель загружает свою идентификационную ключевую пару, т.е. *Identity Key* $= IK^{S}$;

Шаг 6: Отправитель вычисляет мастер-секрет, используя для этого свои закрытые ключи и открытые ключи получателя, т.е. $MS = a \,||\, b \,||\, \mathrm{c} \,||\, \mathrm{d}$, где $a = ECDH(IK_{sk}^{S}, SK_{pk}^{R})$, $b = ECDH(EK_{sk}^{S}, IK_{pk}^{R})$, $c = ECDH(EK_{sk}^{S}, SK_{pk}^{R})$, $d = ECDH(EK_{sk}^{S}, OK_{pk}^{R})$. Если OK_{pk}^{R} отсутствует, то вычисление d не производится;

Шаг 7: Отправитель, используя *HKDF* [11] и мастер-секрет (*MS*), создает *Root* (*RK*) и *Chain* (*CK*) ключи (см. раздел 1.1.3), т.е. *RK*=*CK*=*HKDF*(*MS*). Необходимые для создания ключа шифрования сообщений.

После создания сеанса связи, получатель осуществляет прием открытых ключей отправителя и вычисляет общий секрета, что можно описать следующим алгоритмом:

Шаг 1: Отправитель передает получателю сообщение, содержащее настройки сеанса связи, т.е. IK_{pk}^{S}, SK_{pk}^{S}, OK_{pk}^{S}, EK_{pk}^{S};

Шаг 2: Получатель принимает сообщение от отправителя, затем вычисляет мастер-секрет (*MS*), используя свои закрытые ключи и полученные открытые ключи отправителя, т.е. $MS = a \,||\, b \,||\, \mathrm{c} \,||\, \mathrm{d}$, где $a = ECDH(IK_{sk}^{R}, SK_{pk}^{S})$, $b = ECDH(EK_{sk}^{R}, IK_{pk}^{S})$, $c = ECDH(EK_{sk}^{R}, SK_{pk}^{S})$, $d = ECDH(EK_{sk}^{R}, OK_{pk}^{S})$;

Шаг 3: Получатель удаляет использованный отправителем $(OK_{pk}^{R}, OK_{sk}^{R})$;

Шаг 4: Получатель, используя *HKDF* и мастер-секрет (*MS*), создает *Root* (*RK*) и *Chain* (*CK*) ключи (смотрите раздел 1.1.3), т.е. *RK*=*CK*=*HKDF*(*MS*).

Таким образом, считается, что сеанс защищенной связи был установлен, и пользователи могут безопасно отправлять сообщения друг другу.

1.1.3 Обмен сообщениями

Защита передаваемых сообщений достигается путем их шифрования по алгоритму *AES* в режиме сцепления блоков шифртекста (*Cipher Block Chaining*, *CBC*), а для проверки целостности сообщений, используется код аутентичности сообщения (*Message authentication code*, *MAC*) вычисляемый по *HMAC-SHA*256 [12]. Как было упомянуто в разделе 1.1.2, для создания ключа шифрования сообщений *MK* между двумя пользователями, вырабатываются три симметричных ключа, которые приведены в таблице 2.

Таблица 2 – Симметричные ключи, используемые для защиты сообщения

Обозначение	Описание
Root Key = RK	Корневой ключ, размером 32 байт. Вырабатывается из мастер-секрета (*MS*), предназначен для выработки *Chain Key*.
Chain Key = CK	Ключ цепочки, размером 32 байт. Вырабатывается из *Root Key*, предназначен для выработки *Message Key*.
Message Key = MK	Эфемерный (разовый) ключ сообщения, размером 80 байт. Вырабатывается из *Chain Key*, предназначен для шифрования сообщения. Из 80 байт, 32 байта используются в качестве ключа для *AES*, следующие 32 байта используются в качестве ключа для *HMAC-SHA*256, а оставшиеся 16 байт используются как вектор инициализации.

С каждым отправленным сообщением осуществляется обновление ключа сообщения *MK* по алгоритму «*Double Ratchet*» [13].

1.1.3.1 Алгоритм *Double Ratchet* в системе *WhatsApp*

Алгоритм «*Double Ratchet*» [13] позволяет двум сторонам создать общий, секретный, симметричный ключ шифрования с последующим обновлением ключа шифрования для каждого передаваемого сообщения. Данный алгоритм обладает свойством совершенно прямой секретности (*Perfect forward secrecy*, *PFS*) [14], что обеспечивает его широкое распространение, например, в таких известных *IM* системах, как *Signal*, *WhatsApp*, *Viber*, *Facebook Messenger*, и др. Алгоритм «*Double Ratchet*» представляет собой комбинацию двух алгоритмов «*Symmetric-key ratchet*» и «*DH Ratchet*» [13]. Рассмотрим алгоритм «*Double Ratchet*» в контексте системы *WhatsApp*.

Для каждого нового сообщения отправитель вычисляет ключ сообщения. Изначально полагается, что *CK=MK*. На первом этапе выполняется алгоритм «*Symmetric-key ratchet*». Он вычисляет хэш-код от ключа цепочки *CK* по

HMAC-SHA256, что позволяет выработать новый ключ сообщения *MK*. Выше написанное можно сформулировать следующим алгоритмом:

Шаг 1: Отправитель вычисляет *MK=HMAC-SHA*256(*CK*, 0*x*01);

Шаг 2: Пересчитать *CK=HMAC-SHA*256(*CK*, 0*x*02).

Второй этап, называемый «*DH Ratchet*», выполняется при получении сообщения от отправителя. Помимо шифртекста, сообщение содержит открытый ключ EK_{pk}^{S}, новой эфемерной пары $(EK_{pk}^{S}, EK_{sk}^{S})$, которая была сгенерирована отправителем на *Curve25519* перед отправкой сообщения. Аналогично отправителю, получатель генерирует новую эфемерную пару $(EK_{pk}^{R}, EK_{sk}^{R})$ на *Curve*25519, открытый ключ EK_{pk}^{R} передается отправителю. Затем, получатель вырабатывает новые *CK* и *RK:*

Шаг 1: Получатель вычисляет эфемерный секрет, т.е. $ES = ECDH(EK_{sk}^{R}, EK_{pk}^{S})$;

Шаг 2: Пересчитать $CK, RK = HKDF(RK, ES)$.

Если отправитель передал второе и последующие сообщения до получения ответа от получателя, то новые значения для *CK* и *MK* вычисляются согласно алгоритму «*Symmetric-key ratchet*», описанному на первом этапе.

1.1.4 Проверка аутентичности собеседника

При обмене сообщениями, возможна ситуация, при которой злоумышленник оказывается посредником между отправителем и получателем. То есть, злоумышленник реализует атаку «человек посередине» (*Man-in-the-middel*, *MITM*) [4, 14]. Для защиты от *MITM*-атак пользователям предлагается по сторонним каналам связи, либо при личной встречи верифицировать их *QR*-коды или 60-значный номер.

Согласно [6], *QR*-код формируется из:

- версии;
- идентификатора обоих пользователей;
- 32-битного открытого ключа IK_{pk}, обоих пользователей.

Когда пользователь сканируют *QR*-код собеседника, то сравниваются открытые ключи IK_{pk} идентификационной пары. Это позволяет гарантировать, что содержащиеся ключи в *QR*-коде получены от сервера.

Верификационный 60-значный номер вычисляется путем соединения (конкатенации) двух 30-значных отпечатков, полученных для каждого открытого ключа IK_{pk} идентификационной пары. Вычисление 30-значного отпечатка, выполняется по ниже приведенному алгоритму:

Шаг 1: Итеративно, 5200 раз, вычисляется хэш-код по функции SHA-512 [9] от открытого ключа IK_{pk} и идентификатора пользователя;

Шаг 2: Извлечь первые 30-байт хэш-кода, полученного на шаге 1;

Шаг 3: Разделить 30-байтный результат из шага 2, на шесть блоков по 5-байт;

Шаг 4: Преобразовать каждый 5-байтовый блок в 5-значную цифру. Для этого 5-байтовый блок представляется как целое число без знака, которое уменьшается по модулю 100000;

Шаг 5: Соединить шесть групп из пяти цифр в 30-значную цифру.

1.2 Система обмена сообщениями *Telegram*

Система *Telegram* была разработана российскими разработчиками Павлом и Николаем Дуровым в 2013 году. В отличие от *WhatsApp*, система *Telegram* изначально создавалась для безопасного обмена сообщениями и медиафайлами различных форматов. Поэтому, *Telegram* имеет два режима общения: обычные чаты – используют шифрование типа «клиент-сервер», секретные чаты – используют оконечное шифрование и не сохраняют историю сообщений на серверах *Telegram*. По умолчанию, обмен сообщениями осуществляется в режиме обычных чатов. Рассмотрим подробнее техническую составляющую *Telegram* с позиции организации защиты передаваемой информации между пользователями.

1.2.1 Регистрация пользователя и создания личных ключей

При установке клиентского приложения *Telegram*, пользователь указывает номер телефона, на который отправляется одноразовый смс-код для проведения процедуры аутентификации. После чего запускается протокол авторизации устройства [15]:

Шаг 1: Клиент отправляет на сервер запрос, содержащий случайную строку *nonce* (128 бит);

Шаг 2: В ответ клиенту, сервер отвечает другой случайной строкой *server_nonce* (128 бит), целым числом *n* (64 бит), отпечатком публичного ключа *RSA*. Отпечаток публичного ключа это младшие 64 бита от результата вычисления хэш-функции *SHA1* по открытому ключу сервера $server_{pk}$;

Шаг 3: Клиент раскладывает *n* на два простых числа *p* (2048 бит) и *q* так, что $p < q$. Это служит проверкой того, что пользователь обладает $server_{pk}$, который храниться на его устройстве. Ключ выбирается из списка открытых ключей сервера такой, который соответствует пришедшему отпечатку;

Шаг 4: Клиент выбирает новую случайную строку *new_nonce* (256 бит), не содержащую *nonce* и *server_nonce*. Далее, клиент формирует контейнер,

содержащий *nonce*, *server_nonce*, *new_nonce*, *n*, *p* и *q*. Затем, пользователь передает серверу зашифрованный контейнер алгоритмом *RSA* с помощью ключа $server_{pk}$;

Шаг 5: Сервер отвечает параметрами g, *p*, g_a, протокола Диффи-Хеллмана, зашифрованными алгоритмом *AES-256* в режиме расширения неопределенного искажения (*Infinite Garble Extension*, *IGE*) [16] с помощью временного ключа *temp_key* и вектора инициализации *IV*. На рисунке 1 показана схема вычисления *temp_key* и *IV* из *server_nonce* и *new_nonce*;

Шаг 6: Клиент генерирует случайный закрытый ключ *b* (2048 бит), затем вычисляет открытый параметр $g_b = g^b \bmod p$ и $auth\_key = (g_a)^b \bmod p$. Далее, клиент передает серверу зашифрованный g_b по алгоритму *AES*-256 в режиме *IGE* с помощью ключа *temp_key* и вектора инициализации *IV*.

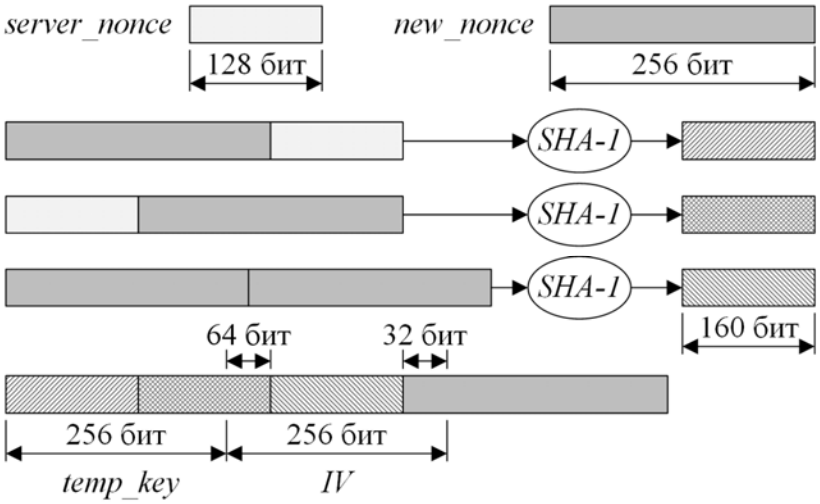

Рисунок 1 – Вычисление временного ключа *temp_key* и вектора инициализации *IV* во время регистрации

На шаге 3, при разложении целочисленного числа *n* на простые множители *p* и *q*, должно выполняться условие:

$$\begin{cases} 2^{2047} < p < 2^{2048}, \\ q = \dfrac{p-1}{2}, \\ p < q, \\ p, q \in P, \end{cases} \qquad (1.2.1.1)$$

где P – множество простых чисел. Числа p и q проверяются на простоту тестом Миллера-Рабина. Возвращаемые, сервером, значения на шаге 5 удовлетворяют условию, что $g \in \{2,3,4,5,6,7\}$ и $1 < g_a < p-1$.

Таким образом, клиент и сервер имеют общий секретный ключ *auth_key*. Он используется для защиты информации передаваемой между сервером и клиентом, а также в обычных чатах. Считается, что устройство зарегистрировано и авторизировано. Если пользователь подключается с нового устройства, то выше приведенный протокол повторяется.

1.2.2 Создание защищенного соединения

Как было описано в разделе 1.2.1, защита клиент-серверного канала связи строится на подходе *Pinning* [17] типа *Public key pinning* с *RSA* (2048-бит). Это значит, что приложение имеет встроенный публичный ключ сервера, который можно использовать для проверки подписи или шифрования, но его нельзя использовать для подписи сообщений. Личный (закрытый) ключ сервера хранится на сервере и изменяется очень редко. То есть, 2048-битный ключ *RSA*, используется для зашифровывания сообщений при генерации ключа авторизации *auth_key* по протоколу Диффи-Хеллмана.

В секретном чате с помощью протокола Диффи-Хеллмана, между двумя пользователями, создается общий секретный ключ *shared_key* (2048 бит), который периодически обновляется в случае зашифровывания и расшифровывания более 100 сообщений, либо при использовании ключа более одной недели. Однако, пользовательский закрытый ключ *a* в протоколе генерируется следующим образом:

$$a = r_{client} \oplus r_{server}, \tag{1.2.2.1}$$

где r_{client} – 2048 битное случайное число, сгенерированное пользователем;

r_{server} – 2048 битное случайное число, сгенерированное сервером для повышения стойкости ключа *a*, в случае ненадежного генератора пользователя.

Открытые параметры протокола Диффи-Хеллмана фиксированы и хранятся на сервере. При выпуске новой версии приложения, параметры могут быть обновлены. Следует отметить, что к открытым параметрам предъявляются требования аналогичные разделу 1.2.1.

1.2.3 Обмен сообщениями

Конфиденциальность передаваемых сообщений в *Telegram* достигается за счет протокола *MTProto* [18, 19]. Однако сначала рассмотрим структуру сообщения, подлежащую зашифровыванию, приведенную на рисунке 2.

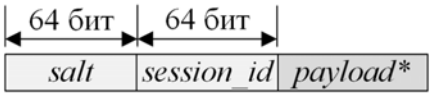

Рисунок 2 – Структура сообщения, подлежащего зашифровыванию в *MTProto*

Ниже приведены ключевые моменты структуры сообщения, которые представлены на рисунке 2:

- *salt* – случайное 64 битное число, предназначено для защиты против атак повторного воспроизведения (*replay attack*). Периодически изменяется для каждого сообщения;
- *session_id* – случайное 64 битное число, сгенерированное клиентом для того, чтобы различать сессии;
- *payload* – полезная нагрузка, т.е. пакет с конфиденциальными данными. На рисунке 3 приведена структура *payload* для сообщений типа «клиент-сервер» и «клиент-клиент» в обычных чатах, а на рисунке 4 показана структура *payload* для сообщений типа «клиент-клиент» в секретных чатах.

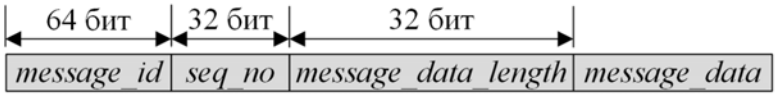

Рисунок 3 – Содержимое *payload* для сообщений типа «клиент-сервер» и «клиент-клиент» в обычных чатах

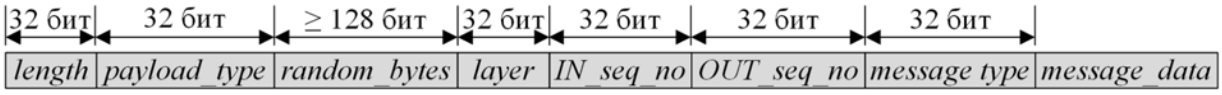

Рисунок 4 – Содержимое *payload* для сообщений типа «клиент-клиент» в секретных чатах

Рассмотрим элементы структуры *payload* из рисунка 3:

- *message_id* – случайное 64 битное число, используемое для уникальной идентификации сообщения внутри сессии;
- *seq_no* – 32 битное число, используется как порядковый номер сообщения;
- *message_data_length* – 32 битное число, длинна данных сообщения;
- *message_data* – сообщения пользователя, произвольной длинны.

Ниже описаны элементы структуры *payload* из рисунка 4:

- *length* – 32 битное число, длинна полезной нагрузки *payload*;
- *payload_type* – 32 битное число, тип полезной нагрузки;
- *random_bytes* – 128 битное число, используется для защиты сообщения от атак повторного воспроизведения;
- *layer* – 32 битное число, обозначает версию протокола;

- *IN_seq_no* – 32 битное число, количество отправленных сообщений к инициатору чата;
- *OUT_seq_no* – 32 битное число, количество отправленных сообщений от инициатора чата;
- *message_type* – 32 битное число, обозначает тип сообщения;
- *message_date* – сообщения пользователя, произвольной длинны.

Также, *payload* из рисунка 4 содержит три заголовка (*header*), которые содержат информацию о протоколе и типе приложенных медиафайлов.

Сообщения типа «клиент-сервер» и «клиент-клиент» в обычных чатах защищаются по схеме шифрования изображённой на рисунке 5.

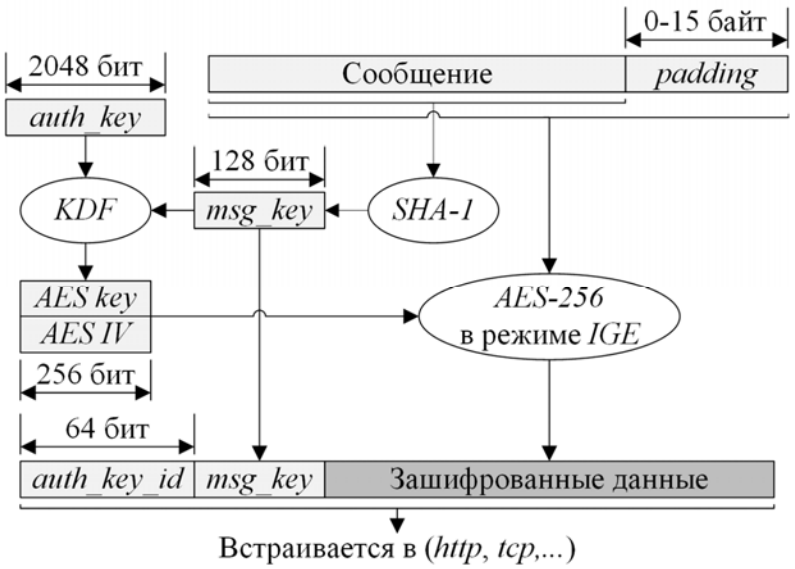

Рисунок 5 – Схема шифрования сообщений типа «клиент-сервер» и «клиент-клиент» в обычных чатах

Полезная нагрузка *payload* в сообщениях типа «клиент-клиент» для секретных чатов подвергается дополнительному зашифровыванию по аналогичной схеме как на рисунке 5. Однако, вместо *auth_key* и *auth_key_id* используется общий секретный ключ пользователей *shared_key*, и его отпечаток *key_fingerprint*. Отпечаток *key_fingerprint* равен последним 64 битам от результата *SHA*-1(*shared_key*). Ниже приведено описание элементов схемы шифрования, изображенных на рисунке 5:

- *padding* – случайные 0-15 байт, добавляются для того, чтобы размер блока шифрования был равен 128 битам;
- *msg_key* – младшие 128 бит, взятые из хэш-кода вычисленного по *SHA*-1 от сообщения;
- *auth_key_id* – младшие 64 бит, взятые из хэш-кода вычисленного по *SHA*-1 от *auth_key*;

• *AES key* и *IV* – ключ и вектор инициализации, размер каждого равен 256 бит, необходимы для алгоритма *AES*-256 в режиме *IGE*, вычисляются по функции формирования ключа (*key derivation function*, *KDF*), как показано на рисунке 6.

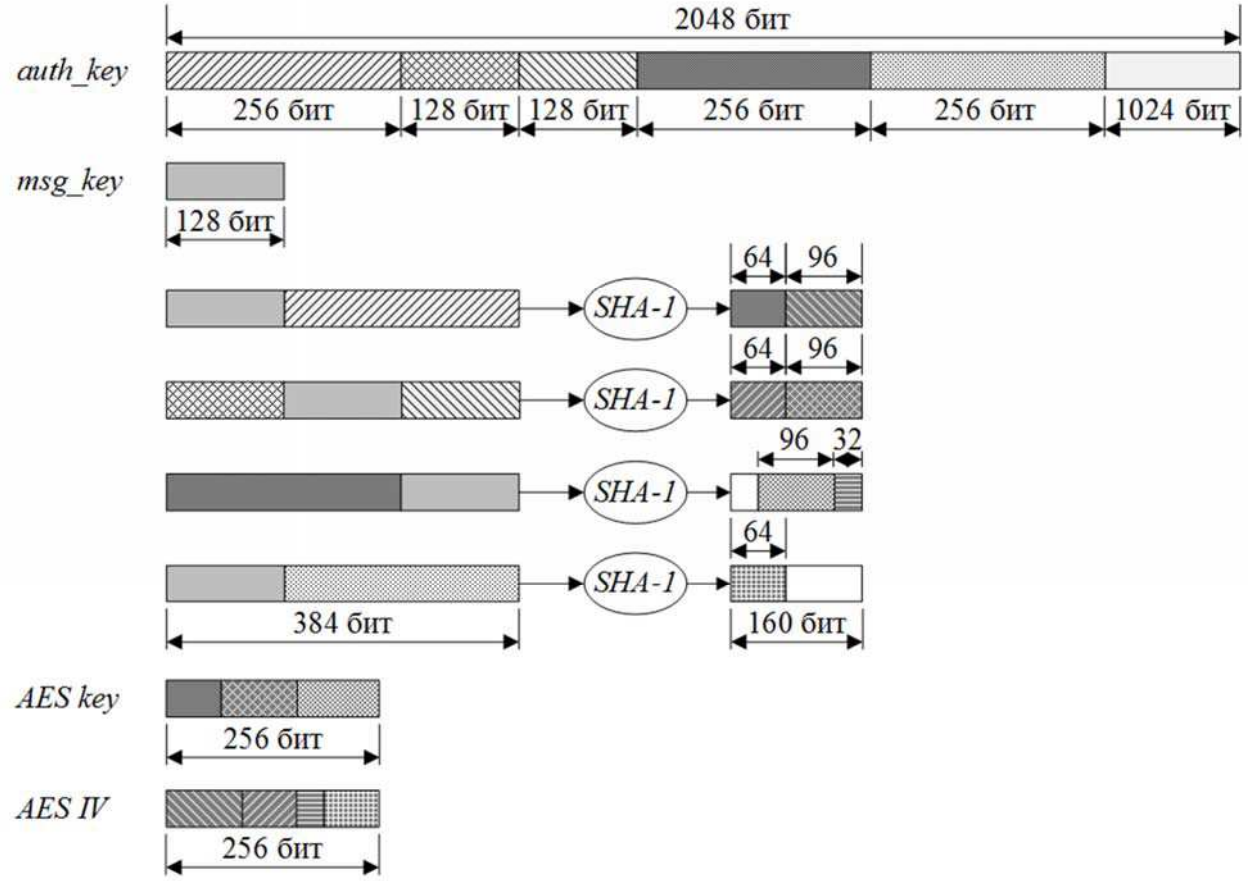

Рисунок 6 – Вычисление ключа *AES key* и *IV*

1.2.4 Проверка аутентичности собеседника

Защита от *MITM*-атак в секретных чатах, системы *Telegram*, построена на визуальном сравнении общего ключа шифрования (*shared_key*) пользователей по сторонним каналам связи, либо при личной встречи [20]. Визуализированное изображение представляет собой цветную полигональную сетку размером 8*x*8, полученную из последних 128 бит от результата *SHA*-1(*shared_key*).

1.3 Система обмена сообщениями *Viber*

Первая версия *Viber* была выпущена разработчиками Игорем Магазинником и Тальмоном Марко в 2010 году, для бесплатного общения через аудио/видео звонки с целью замены сотовой связи и *CMC*. Сегодня у *Viber*, по собственным данным, около 800 миллионов зарегистрированных пользователей [21], из них 76 миллионов – в Российской Федерации [22]. Начиная с версии 6.0 система *Viber* также стала использовать концепцию *E2EE* для защиты информации передаваемой

пользователями. Для создания *E2EE* в системе *Viber* применяется алгоритм «*Double Ratchet*» из *Signal Protocol* [10], однако в отличии от *WhatsApp*, который использует реализацию от *Open Whisper Systems*, разработчики *Viber* реализовали алгоритм «*Double Ratchet*» самостоятельно. Рассмотрим обеспечение информационной безопасности в *Viber* более подробно.

1.3.1 Регистрация пользователя и создания личных ключей

Как сообщается в [23], система *Viber* подразделяет устройства пользователя на основное (первичное) и необязательное (вторичное). Основное устройство – это мобильный телефон под управлением *iOS*, *Android* или *Windows*, с зарегистрированным номером телефона в *Viber*. Необязательным устройством является планшет, либо настольный ПК. Основное и необязательное устройства связаны между собой учетной записью пользователя, состоящей из одного основного устройства и неограниченного числа необязательных устройств.

При установки *Viber* на основное (первичное) устройство, создаются ассиметричные пары ключей, которые представлены в таблице 3.

Таблица 3 – Ключи пользователя

Обозначение	Описание
ID Key = *IK* = (IK_{pk}, IK_{sk})	Долговременная, идентификационная ключевая пара пользователя.
PreKeys = {*Handshake Key*, *Ratchet Key*}	Набор среднесрочных ключей, используемых для создания защищенного сеанса связи между двумя устройствами.
Handshake Key = *HK* = (HK_{pk}, HK_{sk})	Ключевая пара, используется совместно с *ID Key* для создания корневого ключа.
Ratchet Key = *RK* = (RK_{pk}, RK_{sk})	Ключевая пара, используется для создания сеансового ключа между пользователями.

Данные ключи генерируются на эллиптической кривой *Curve*25519 [5, 24], имеют размерность 256 бит, и состоят из открытого и закрытого ключей. Пара *ID Key* генерируется только на основном устройстве, а серия пар *PreKeys* генерируется каждым устройством для учетной записи. Закрытые ключи хранятся на устройстве пользователя, а открытые ключи загружаются на сервер *Viber*. Необязательное (вторичное) устройство получает закрытый ключ IK_{sk} от основного (первичного) устройства с помощью алгоритма описанного в разделе 1.3.1.1.

1.3.1.1 Регистрация необязательного (вторичного) устройства

Во время регистрации необязательного устройства (обозначается как *SD*), осуществляется передача IK_{sk}^{PD} от основного устройства (обозначается как *PD*) по следующему алгоритму:

Шаг 1: Необязательное (вторичное) устройство генерирует эфемерную пару ключей $EK^{SD} = (EK_{pk}^{SD}, EK_{sk}^{SD})$ на *Curve*25519, размерностью 256 бит;

Шаг 2: Необязательное устройство генерирует *QR*-код, состоящий из *UDID* (общедоступный уникальный идентификатор устройства, сгенерированный *Viber*) и EK_{pk}^{SD};

Шаг 3: Используя основное (первичное) устройство, пользователь извлекает *UDID* и EK_{pk}^{SD} из *QR*-кода, путем его сканирования;

Шаг 4: Основное устройство генерирует пару ключей $EK^{PD} = (EK_{pk}^{PD}, EK_{sk}^{PD})$ на *Curve*25519, размерностью 256 бит. Затем, устройство вырабатывает общий секретный ключ *SK*, т.е. $SK = SHA256(ECDH(EK_{sk}^{PD}, EK_{pk}^{SD}))$;

Шаг 5: Основное устройство зашифровывает IK_{pk} с помощью ключа *SK* посредством поточного шифра *Salsa20* [24], на основе шифртекста вычисляется код аутентичности сообщения (*MAC*-код) по *HMAC-SHA256*. Далее, основное устройство передает EK_{pk}^{PD}, шифртекст, и код аутентичности на целевое (необязательное) устройство, используя для этого считанный *UDID* и сервера *Viber*;

Шаг 6: Получив сообщение, необязательное устройство вычисляет *SK*, т.е. $SK = SHA256(ECDH(EK_{sk}^{SD}, EK_{pk}^{PD}))$. Далее, устройство расшифровывает шифртекст с помощью ключа *SK* и проверяет целостность полученных данных.

Таким образом, необязательному устройству становиться известен закрытый ключ IK_{sk}^{PD}, который был создан основным устройством.

1.3.2 Создание защищенного соединения

Защищенное соединение между отправителем (инициатор) и получателем, *S* и *R* соответственно, устанавливается по следующему алгоритму:

Шаг 1: Отправитель отправляет запрос на сервер *Viber* с номером телефона получателя;

Шаг 2: Сервер отвечает отправителю открытыми ключами получателя, т.е. IK_{pk}^{R} и серией открытых ключей *PreKeys* по одному на каждое зарегистрированное устройство получателя, например, HK_{pk}^{R} и RK_{pk}^{R};

Шаг 3: Отправитель генерирует на *Curve*25519 две пары ключей, размерностью 256 бит, называемые $HK^{S} = (HK_{pk}^{S}, HK_{sk}^{S})$ и $RK^{S} = (RK_{pk}^{S}, RK_{sk}^{S})$;

Шаг 4: Отправитель вырабатывает корневой ключ *rootKey*, т.е.

$rootKey = SHA256(a \| b \| c)$, где $a = ECDH(IK_{sk}^{S}, HK_{pk}^{R})$, $b = ECDH(HK_{sk}^{S}, IK_{pk}^{R})$, $c = ECDH(HK_{sk}^{S}, HK_{pk}^{R})$;

Шаг 5: Отправитель вычисляет сессионный ключ *sessionKey* и обновляет *rootKey*, следующим способом:

$$\begin{aligned} &tempKey = HMAC-SHA256(rootKey, ECDH(RK_{sk}^{S}, RK_{pk}^{R})); \\ &rootKey = HMAC-SHA256(tempKey, "root"); \\ &sessionKey = HMAC-SHA256(tempKey, "msg"); \end{aligned} \quad (1.3.2.1)$$

Шаг 6: Отправитель передает получателю сообщение через сервер *Viber*, содержащее: IK_{pk}^{S}, HK_{pk}^{S}, RK_{pk}^{S}, идентификаторы использованных открытых ключей *PreKeys* получателя. Данное сообщение служит началом безопасного сеанса связи между пользователями;

Шаг 7: Получив сообщение отправителя, получатель может вычислить *rootKey* и *sessionKey*. Корневой ключ *rootKey* вычисляется по аналогии с шагом 4, т.е. $rootKey = SHA256(a \| b \| c)$, где $a = ECDH(IK_{sk}^{R}, HK_{pk}^{S})$, $b = ECDH(HK_{sk}^{R}, IK_{pk}^{S})$, $c = ECDH(HK_{sk}^{R}, HK_{pk}^{S})$. Сессионный ключ *sessionKey* вычисляется следующим образом:

$$\begin{aligned} &tempKey = HMAC-SHA256(rootKey, ECDH(RK_{sk}^{R}, RK_{pk}^{S})); \\ &rootKey = HMAC-SHA256(tempKey, "root"); \\ &sessionKey = HMAC-SHA256(tempKey, "msg"). \end{aligned} \quad (1.3.2.2)$$

Установив защищенный сеанс связи, пользователи могут использовать его для обмена сообщениями.

1.3.3 Обмен сообщениями

После того, как безопасное соединение было установлено между устройством отправителя и всеми устройствами получателя, ниже приведенный алгоритм обеспечивает конфиденциальность передаваемых сообщений:

Шаг 1: Отправитель создается разовый эфемерный 128 битный симметричный ключ *messageKey* для зашифровывания сообщения *M* по алгоритму шифрования *Salsa*20 [24], т.е. $CM = Salsa20(M, messageKey)$;

Шаг 2: Отправитель зашифровывает *messageKey* с использованием сеансового ключа *sessionKey* для каждого защищенного соединения, созданного между устройствами получателя, т.е. $CS_i = Salsa20(messageKey, sessionKey_i)$, где

$i \in N$ - количество устройств (защищенных соединений), которым доставляется сообщение;

Шаг 3: Отправитель формирует сообщение, состоящее из *CM* и множества CS_i. Сформированное сообщение, отправитель передает на сервер *Viber*;

Шаг 4: Сервер принимает сообщение, и доставляет каждому устройству получателя соответствующий CS_i и *CM*.

Каждый раз, когда меняется инициатор отправки сообщения в чате, т.е. отправитель сообщения становится получателем, отправляющее устройство генерирует новый $RK^S = (RK^S_{pk}, RK^S_{sk})$ и обновляет *sessionKey* согласно (1.3.2.1). Новый открытый ключ RK^S_{pk} передается получателю совместно с сообщением. Получатель вычисляет общий секрет по *ECDH* с помощью своего закрытого ключа RK^R_{sk} из последней пары RK^R и нового открытого ключа RK^S_{pk}.

1.3.4 Проверка аутентичности собеседника

В системе *Viber* защита от компрометаций «человека по середине» устроена в контексте аудио/видео звонка. При вызове пользователь может посмотреть *QR*-код, который вычисляется следующим образом:

Шаг 1: Оба пользователя вычисляют общий секрет *shared_secret* по *ECDH* с помощью своего закрытого ключа и открытого ключа собеседника, т.е. $shared\_secret = ECDH(IK^S_{sk}, IK^R_{pk}) = ECDH(IK^R_{sk}, IK^S_{pk})$;

Шаг 2: Вычислить по хэш-функции *SHA*256 хэш-код от *shared_secret*, т.е. $h = SHA256(shared\_secret)$. Далее, *h* обрезается до 160 бит и преобразуются в строку из 48 десятичных знаков (0-9).

Таким образом, оба пользователя должны увидеть одну и туже строку чисел. Сравнивая числовую последовательность, путем проговаривания её друг другу, пользователи могут быть уверены, что *MITM*-атака не выполняется.

1.4 Система обмена сообщениями *Threema*

Проприетарная разработка швейцарской компании *Threema GmbH*, выпустившая первую версию *Threema* в 2012 году для устройств *iPhone*. На декабрь 2015 года насчитывается более 3.7 миллионов пользователей, а также список поддерживаемых платформ пополнился устройствами на базе *Android* и *Windows Phone* [25]. Как и другие аналогичные *IM* системы, в *Threema* вопросы безопасности и конфиденциальности пользовательских данных [26] поставлены на первое место, защищая всю передаваемую информацию по концепции *E2EE* с использованием открытой криптографической библиотеки *NaCl* [27, 28]. Однако, уникальной особенностью системы является то, что для использования приложения не нужен

номер телефона или адрес электронной почты. Эта особенность позволяет пользоваться *Threema* анонимно. Рассмотрим, как в система *Threema* обеспечивает конфиденциальность и целостность пользовательской информации.

1.4.1 Регистрация пользователя и создания личных ключей

Согласно [26], в процессе установки клиентского приложения *Threema* на устройство пользователя, выполняется следующий алгоритм:

Шаг 1: Приложение генерирует ключевую пару (IK_{pk}, IK_{sk}) на *Curve*25519, размером 256 бит;

Шаг 2: Приложение отправляет открытый ключ IK_{pk} пары на сервер;

Шаг 3: Сервер принимает, сохраняет, назначает открытому ключу случайный *Threema ID*, состоящий из 8-ми символов типа *A-Z*/0-9. Далее, сервер отправляет *Threema ID* приложению;

Шаг 4: Приложение сохраняет полученный *Threema ID* вместе с ключевой парой в безопасном хранилище на устройстве.

1.4.2 Создание защищенного соединения

Система *Threema* для защиты клиент-серверного канала использует протокол *TLS* версии 1.2 наряду с подходом *Certificate Pinning* [17], а для создания *E2EE* между абонентами использует открытые ключи абонентов, которые хранятся на сервере вместе с *Threema ID* [26]. Пользователь, желающий установить защищенное соединение, может запросить открытый ключ собеседника по его *Threema ID*, после чего, возможен обмен сообщениями.

1.4.3 Обмен сообщениями

Защиту передаваемых сообщений в *Threema*, между отправителем и получателем (*S* и *R* соответственно), можно описать схемой, приведенной на рисунке 7, либо следующим алгоритмом:

Шаг 1: Отправитель, используя *ECDH* на кривой *Curve*25519 и хэш-функцию *HSalsa*20 [29], вычисляет общий секрет *shared_secret* (256 бит), т.е. $shared\_secret = HSalsa20(ECDH(IK_{sk}^{S}, IK_{pk}^{R}))$;

Шаг 2: Отправитель генерирует псевдослучайную последовательность чисел *nonce*, размером 192 бит, которая используется как уникальный номер сообщения;

Шаг 3: Отправитель зашифровывает сообщение поточным шифром *XSalsa*20 [29] с помощью *shared_secret*, используется как ключ шифрования, и последовательности *nonce*;

Шаг 4: Отправитель, используя функцию *Poly*1305-*AES* [30], вычисляет код аутентичности сообщения (*MAC*-код) размером 128 бит. Часть потока из *XSalsa*20 используется в качестве ключа для *Poly*1305;

Шаг 5: Отправитель передает получателю: шифртекст сообщения, *MAC*-код, *nonce*.

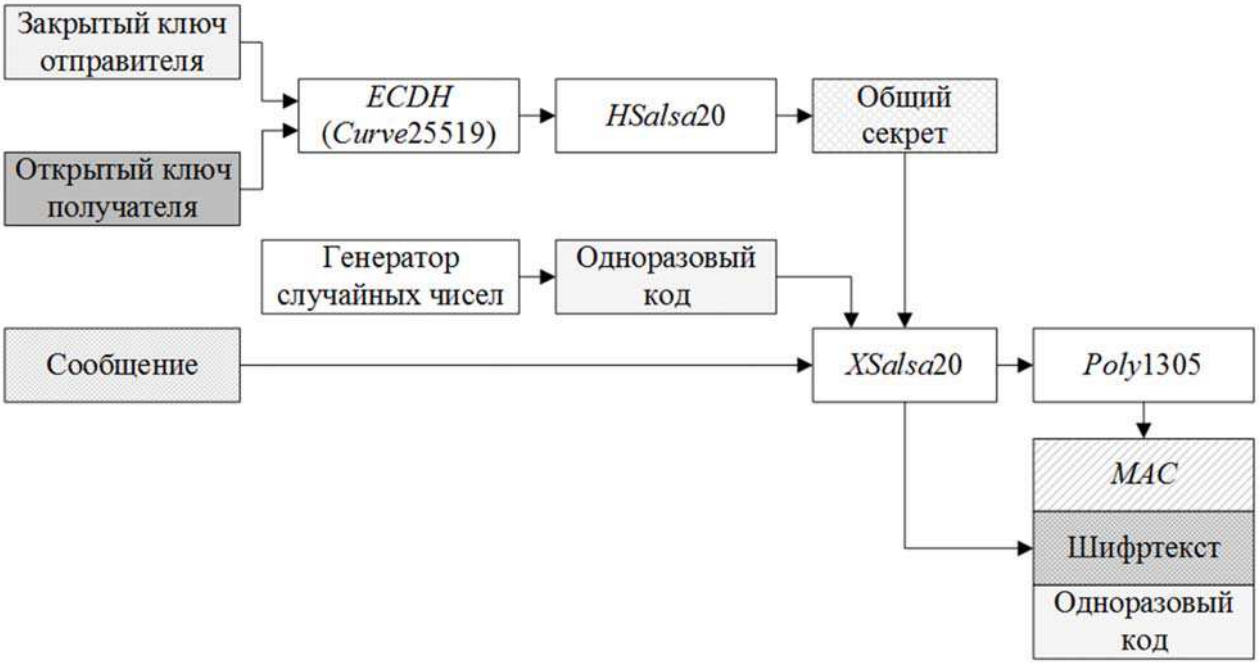

Рисунок 7 – Схема преобразования сообщения

1.4.4 Проверка аутентичности собеседника

Для повышения безопасности общения пользователя в системе *Threema*, каждый пользовательский контакт имеет индикатор «уровня проверки», состоящий из следующих уровней:

– красный (уровень 1). Это значит, что пользователь не может быть уверен в том, что собеседник является тем, кем за себя выдает. С технической точки зрения, это значит, что идентификатор собеседника (*Threema ID*) был введен вручную, либо открытый ключ собеседника был получен с сервера, поскольку сообщение было получено в первый раз. Также соответствующий собеседник не был найден в списке контактов телефонов по телефонному номеру или электронной почте (*email*);

– жёлтый (уровень 2). Это значит, что пользователь может быть уверен в том, что собеседник тот, за кого себя выдает. Поскольку собеседник был найден в телефонном списке контактов, а также сервер проверил собеседника. Проверка осуществляется путем отправки смс-сообщения на номер телефона, либо отправкой гиперссылки на электронную почту;

– зелёный (уровень 3). Это подразумевает, что пользователь проверил идентификатор собеседника и его открытый ключ путем личной встречи, и сканирования *QR*-кода. Формат *QR*-кода имеет вид: «*identity*, *publicKeyHex*», где *identity* - это 8-ми символьный *Threema ID*, и *publicKeyHex* – это шестнадцатеричное (в нижнем регистре) представление 32 байтного открытого ключа.

1.5 Выводы по главе

Проведенный в рамках данной главы обзор существующих и наиболее распространенных *IM* систем с повышенным уровнем безопасности позволил обнаружить общий и существенный недостаток: отсутствие доверия к центральной стороны (сервис-провайдеру) *IM* систем. При этом центральная сторона может быть скомпрометирована как злоумышленниками, так и владельцем системы по тем или иным причинам. Другими словами, может быть реализована атака «человек посередине» (*MITM*) [4], которая в свою очередь несет ряд угроз нарушения конфиденциальности и целостности информации пользователей в процессе информационного обмена. Наличие таких угроз, особенно в контексте того, что большая часть распространённых *IM* систем принадлежит иностранным компаниям, существенно ограничивает или делает невозможным их использование для обмена конфиденциальной информацией. Схожая атака может иметь место и в случае компрометации базы данных (БД) открытых ключей пользователей, хранящихся на серверах *IM* системы. Злоумышленник может временно подменить открытые ключи абонентов, хранящиеся в БД. Подмена ключей, знание структуры передаваемых данных и идентификаторов абонентов, также позволит злоумышленнику создать сеанс связи от чужого имени и скомпрометировать весь информационный обмен. Поскольку, в случае компрометации сервера, злоумышленник может скомпрометировать передаваемые значения открытых ключевых параметров своими значениями, выработать с каждым участником информационного обмена общий ключ, а затем получить полный доступ к конфиденциальной информации. Данная *MITM*-атака носит вполне вероятный характер и уже была продемонстрирована в [31] на примере системы *Signal* (*TextSecure*), а также в аналитическом обзоре [32] протокола *Telegram*.

Таким образом, существует актуальная научно-техническая задача по организации безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа, с учётом обозначенных недостатков, существующих *IM* систем. Решение данной задачи включает в себя разработку модели и архитектуры информационной системы, протоколов сетевого взаимодействия, а также алгоритмического обеспечения.

Глава 2. Предлагаемый способ конфиденциального обмена и модель информационной системы

2.1 Используемые криптографические примитивы

Чтобы обеспечить безопасность коммуникации в компьютерных сетях общего доступа, разрабатываемая модель ИС для конфиденциального обмена информацией должна иметь базис основанный на стойких криптографических примитивов, поскольку наличие изъяна в криптографическом алгоритме по умолчанию означает, что предлагаемая модель ИС ненадежна. Поэтому следует описать используемые криптографические примитивы, и обосновать причину их выбора для решения поставленных задач.

2.1.1 Способ генерации случайных чисел

Для генерации случайных последовательностей, включая ключевую информацию, используется криптографически стойкий генератор псевдослучайных чисел (КСГПСЧ). Технологическая реализация КСГПСЧ в предлагаемой ИС основана на библиотеке криптографических алгоритмов *WebCrypto GOST* [33], включающая в себя КСГПСЧ. Как отмечается в документации криптографической библиотеки, генератор создает криптографически стойкую последовательность псевдослучайных чисел [34] требуемой длинны, следующим образом: с помощью криптографического генератора *JavaScript Web Crypto* [35] (если среда выполнения поддерживает), либо не криптографического генератора *Math.random* (в ином случае) генерируется последовательность, которая смешивается с энтропией с помощью битовой операции *XOR*. В качестве источника энтропии выступает текущее время, смешанное по *XOR* с движением мыши (положение курсора), либо с кодом нажатой клавиши.

2.1.2 Используемая хэш-функция

Одним из аспектов безопасности информации является обеспечение целостности информации, для решения данной задачи используется российская хэш-функция ГОСТ Р 34.11-2012 [36] с длинной выхода 512 бит. Выбор в качестве хеш-функции *H* стандарта ГОСТ Р 34.11-2012 обусловлен тем, что данная хеш-функция удовлетворяет современным требованиям к криптографической стойкости.

2.1.3 Используемая функция формирования симметричного ключа шифрования на основе секретного значения

Обще известно, что безопасность защищаемой информации определяется в первую очередь криптографической стойкостью ключа шифрования, поскольку

используемый алгоритм шифрования может быть известен злоумышленнику исходя из требований Керкгоффса к криптосистемам. Использование секретной фразы (пароля) введенной абонентом в качестве симметричного ключа шифрования «как есть» без предварительной обработки, порождает ряд серьезных проблем безопасности, в частности, ограничение ключевого пространства, что порождает криптографическую не стойкость шифртекста, а это ведет к нарушению конфиденциальности информации.

Для решения данной проблемы, выработка криптографического ключа абонента производится на основе абонентской секретной фразы (пароля) *PP* по стандарту формирования ключа на основе пароля (*Password-Based Key Derivation Function*, *PBKDF*), в частности *PBKDF*2 [37] описанный в *RFC* 2898 (*PKCS* #5) [38]:

$$k_a = PBKDF2(H, PP, S_1, C, L), \qquad (2.1.3.1)$$

где *PBKDF2* – функция формирования ключа на основе пароля;

H – хэш-функия ГОСТ Р 34.11-2012 с длинной выхода 512 бит;

PP – секретная фраза абонента;

S_1 – случайная последовательность размерностью 256 бит;

C – количество итераций, значение равное 4000;

L – желаемая длинна ключа в битах, значение равное 256 бит.

Используемые значения параметров в функции *PBKDF*2 были выбраны исходя из рекомендаций Национального института стандартов и технологий США (*National Institute of Standards and Technology*, *NIST*), описанных в источнике [37].

Выбор *PBKDF2* обусловлен тем, что алгоритм в декабре 2010 года был рекомендована *NIST* для использования в криптографических системах, рекомендован стандартом *RFC* 2898 (*PKCS* #5), а также *PBKDF2*, в отличие от аналогов *PBKDF1* или *KDF_GOSTR3411_2012_256*, генерирует ключ не фиксированной длинны, что позволяет использовать *PBKDF2* в различных криптографических алгоритмах, а не только в тех, которые используют ключи определенной длинны.

2.1.4 Используемый алгоритм симметричного шифрования

Конфиденциальность информации обеспечивается российским блочным шифром ГОСТ Р 34.12-2015 [39] в режиме обратной связи по выходу (*OFB*) [40], в данной работе обозначается как *SE*. Выбор режима шифрования с обратной связью по выходу (*OFB*) обусловлен следующими его положительными сторонами:

– отсутствие необходимости в дополнении последнего блока незначащими, обратимыми байтами. Отсутствие дополнения сокращает расходы на пересылку, что особенно важно для большого количества коротких сообщений;

– возможность организации потоковой передачи данных виде/аудио-звонка, что является одной из функций ИС;

– локализация распространения ошибок передачи в пределах одного блока. Это весьма актуально в условиях потоковой передачи данных виде/аудио-звонка при плохой мобильной связи, обеспечивающей соединение с сетью Интернет.

2.1.5 Используемый алгоритм ассиметричного шифрования

Конфиденциальная информация

2.2 Предлагаемый способ обмена конфиденциальной информацией

Конфиденциальная информация

2.3 Модель информационной системы для конфиденциального обмена

Конфиденциальная информация

2.4 Оценка безопасности предлагаемого способа

Предлагаемый способ для безопасного обмена информацией в компьютерных сетях общего доступа, исходя из постановки задачи, основывается на следующей неформальной модели нарушителя:

– нарушитель может быть внешним и оказывать воздействие на абонентское устройство из компьютерной сети, например, посредством активного сниффинга;

– нарушитель может быть внутренним и оказывать воздействие на центральный узел ИС (сервер), например, иметь доступ к базе данных ИС;

– нарушитель обладает всей информацией, необходимой для подготовки и проведения атак, за исключением информации, доступ к которой со стороны нарушителя исключается, например, информация о закрытых ключах абонента;

– нарушитель обладает всем необходимым для проведения атак (знаниями и средствами).

При этом, в рамках данной работы, рассматриваются следующие виды атак на предлагаемую ИС:

– пассивная и активная формы атаки «человек по середине» (*MITM*-атака), включая перехват, искажение и вставку;

– подмена (*impersonation*) абонента;

– повторное навязывание сообщения (*replay attack*), включая:

  – задержку передачи сообщений (*forced delay*);

  – отражение (*reflection attack*);

– комбинированная атака (*interleaving attack*).

В рамках данной неформальной модели нарушителя и рассматриваемых потенциальных атак определим, что под понятием безопасного обмена

информацией понимается комплекс требований и мер, реализованных в предлагаемой ИС с целью, обеспечения доступа к информации уполномоченных пользователей, предотвращения несанкционированного доступа к ИС, обеспечения конфиденциальности, доступности и целостности передаваемой информации.

Оценим предпринятые меры защиты информации от рассматриваемого класса атак с позиции рассматриваемой модели нарушителя:

– защита от *MITM*-атаки со стороны внешнего нарушителя на множество сообщений типа «абонент-сервер» достигается путем применения известного подхода к распределению ключей – *Pinning*, типа *Public key*. Этот подход позволяет неявным образом аутентифицировать сервер, и предотвратить возможность, злоумышленника, завладеть сессионным ключом k_{ses}, так как для этого необходимо обладать серверной частью закрытого ключа k_S^{pri}. Даже если нарушитель гипотетически сможет выступить в качестве посредника в *MITM*-атаке и подменит сообщение абонента, то он не сможет выдать принимающему абоненту зашифрованный ответ по ожидаемому ключу k_{ses}. Также стоит отметить, что во время создания сеанса связи создается ассоциативная связь между идентификатором абонента в ИС, сеансовым ключом k_{ses}, идентификатором сеанса id_{ses} и токеном (маркером) безопасности. Эта ассоциация сохраняется как у абонента, так и на сервере в списке активных сессии, что позволяет серверу разграничивать доступ, и предотвращать неправомерный доступ к информации;

– защита от *MITM*-атаки со стороны внешнего нарушителя на множество сообщений типа «абонент-абонент» основывается на использовании счётчика сообщений и кода целостности. Чтобы сформировать правильный код целостности или узнать значения счётчика, нарушителю необходимо знать закрытый ключ k^{pri}, подменяемого абонента, либо подменить абонентский открытый ключ k^{pub}, т.е. скомпрометировать все используемые каналы связи в ИС, что является трудновыполнимой задачей. В ином, случае злоумышленник будет вынужден использовать свою часть закрытого ключа $k'^{pri} \neq k^{pri}$, что легко выявляет факт подмены абонента;

– для защиты от повторного навязывания сообщений используются одноразовые коды и метки времени, по которым, в том числе, можно определять актуальность («свежесть») сообщений;

– для защиты передаваемых открытых ключей от *MITM*-атаки внутренним нарушителем используются различные и независимые каналы связи и механизмы обеспечения целостности передаваемых данных. Отметим, что абонент-отправитель для передачи частей ключа абоненту-получателю может случайным образом выбирать канал отправки, за исключением уже выбранного канала связи. В случае если нарушитель скомпрометировал один из каналов связи, например,

подменил часть ключа k_1 на k_1', то абонент-получатель сможет обнаружить подмену, путем сравнения значения вычисленного хэш-кода открытого ключа со значением, который на сервере опубликовал абонент-отправитель, т.е. $H(k_1 \bullet k_2) \neq H(k_1' \bullet k_2)$;

– для защиты от внутреннего нарушителя конфиденциальной информации абонента, которая хранится в базе данных ИС, применяется код целостности, а также шифрование с применением закрытого ключа абонента k^{pri} или симметричного ключа абонента *k*, созданного на основе секретной фразы абонента;

– для защиты от несанкционированного доступа предлагаемая ИС реализует функции аутентификации (в том числе и многофакторной), авторизации и управления доступом.

Предлагаемая ИС в полном объёме реализует данные меры по защите информации, что позволяет говорить о безопасности предлагаемого способа обмена конфиденциальной информацией в компьютерных сетях общего доступа в рамках рассматриваемой модели нарушителя и класса потенциальных атак.

2.5 Выводы по главе

Данная глава раскрывает сущность используемых криптографических примитивов с обоснованием их выбора, описывает предлагаемый способ обмена ключевой информацией. Формализуется и предлагается модель информационной системы для обмена конфиденциальной информацией в компьютерных сетях общего доступа. Кроме того, в данной главе описана инфраструктура и алгоритмы распределения ключей шифрования, которые построены исключительно на основе Российских стандартов в области криптографии, а также протоколы сетевого взаимодействия. Помимо этого, в главе рассматриваются аспекты безопасности предлагаемого подхода, описывается неформальная модель нарушителя, рассматриваются потенциальные атаки на ИС, определяется понятие безопасности, а также описываются предпринятые меры защиты информации от рассматриваемого класса атак в рамках модели нарушителя.

Глава 3. Разработка и тестирование информационной системы

3.1 Разработка информационной системы

На основании главы 2 и приложения А могут быть спроектированы и разработаны программные части программного комплекса, а также сопутствующие подсистемы. Однако, для выполнения данной задачи, сперва необходимо определить используемую методику проектирования и методологию разработки.

3.1.1 Подход к разработке

Для разработки программного комплекса системы безопасного обмена информацией в компьютерных сетях общего доступа была использована итерационная модель (*Iterative model*) жизненного цикла, выбор обусловлен, хорошей изученностью данной модели и возможностью выявления дефектов на раннем этапе разработки. Проектирование программного комплекс основывалось на нисходящем объектно-ориентированном подходе [42, 43], т.е. для проектируемой системы сначала определяются высокоуровневые компоненты, а затем конкретные. Результатом этого является, четко определенная, модульная архитектура, которая описана в следующих разделах данной главы.

3.1.2 Проектирование архитектуры информационной системы

Исходя из ранее описанного, программный комплекс (ПК) построен на трёхуровневой архитектуре «клиент-сервер» и спроектирована без привязанности к технологической составляющей, общий вид которой представлен на рисунке 15.

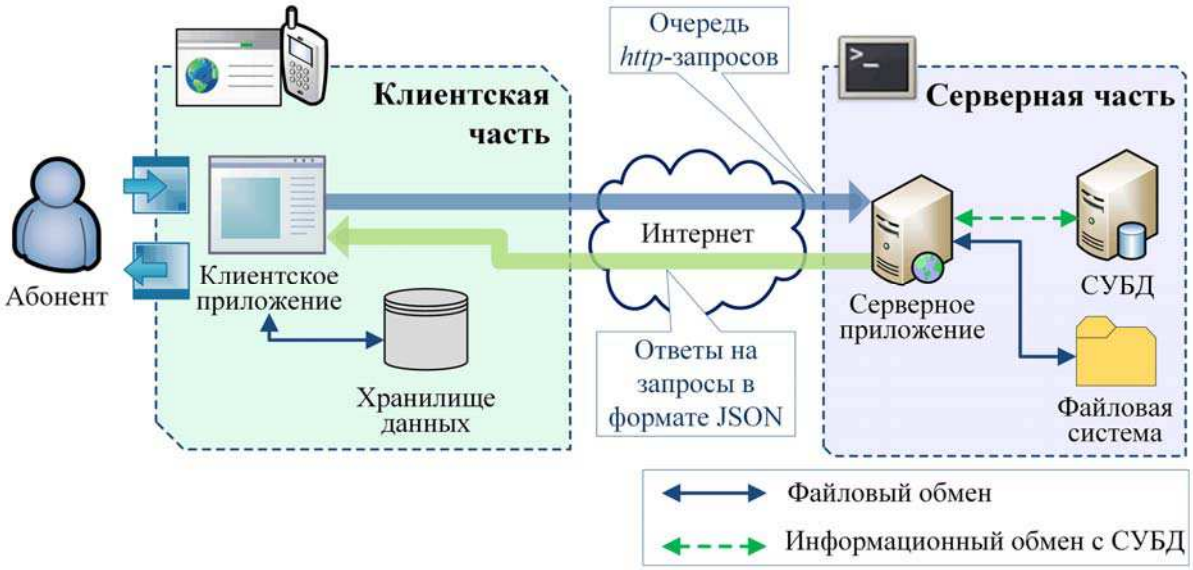

Рисунок 15 – Общий вид архитектуры программного комплекса

Предполагается, что клиентская часть разворачивается на платформе мобильного устройства, либо на отдельном сервере в виде *web*-приложения с предоставлением сервиса для абонента. Клиентская часть состоит из клиентского приложения, отвечающее за интерактивное взаимодействие с абонентом, и хранилища данных – предназначено для хранение ключевой информации. При взаимодействии абонента с клиентской частью, клиентское приложение формирует *http*-запросы и посылает их на обработку серверной части, после чего результат обработки отображается на интерфейсе клиентского приложения. Серверная часть отвечает за обработку *http*-запросов, присланных с клиентской части, а также за хранения абонентской информации. На рисунке 15 серверная часть представлена в монолитном представлении, состоит из серверного приложения – слой бизнес-логики, системы управления базой данных (СУБД) и файловой системой сервера – слой хранения данных. Рассмотрим архитектуру клиентской и серверной частей приложений более детализировано.

3.1.2.1 Архитектура серверной части

Серверное приложение работает под управлением терминала (командной строки), реализует логику обработки данных, координирует работу клиентской части и взаимодействие со слоем хранения данных, взаимодействует со сторонними сервисами, а также реализует программный интерфейс (*API*), посредством которого осуществляется взаимодействие с клиентским приложением. На рисунке 16 приведена архитектура серверной части.

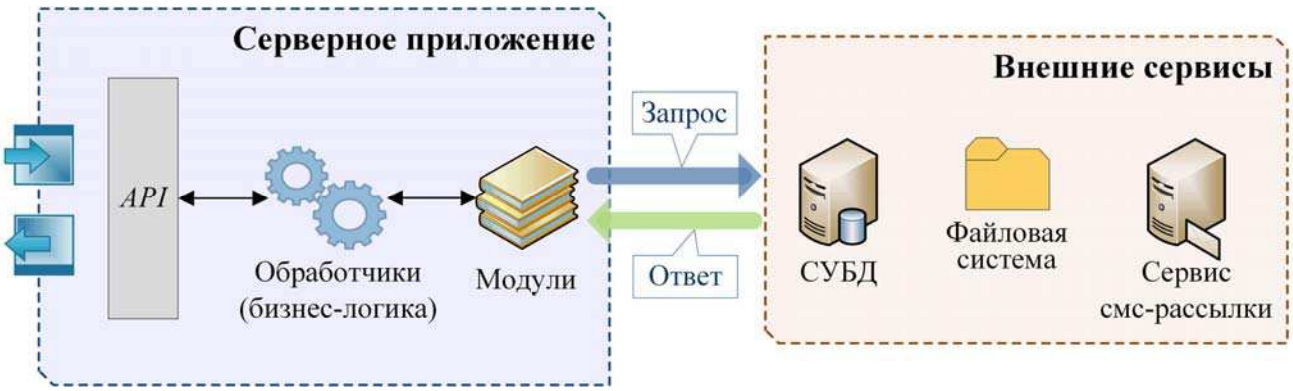

Рисунок 16 – Архитектура серверной части

На рисунке 16 в общем виде представлена схема обработки *http*-запросов из предоставляемого программного интерфейса. Для каждого приходящего запроса, определенного в *API*, маршрутизатор вызывает нужный обработчик (функцию обратного вызова). Он (обработчик) реализует бизнес-логику обработки запроса, в работе которой могут использоваться модули, приведенные на рисунке 17, и описанные в таблице 7.

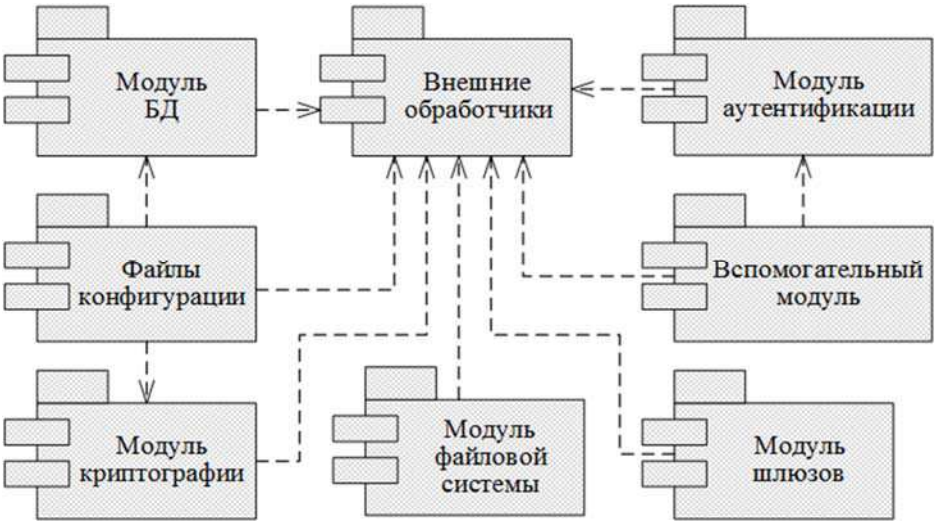

Рисунок 17 – Архитектура серверного приложения на уровне модулей и их взаимодействия между собой виде диаграммы компонентов

Таблица 7 – Описание модулей и их составных элементов

Модуль	**Описание**
Внешние обработчики	Отвечают за обработку *http*-запросов, получаемых от маршрутизатора запросов, и пересылают полученные данные на клиентскую часть.
Модуль файловой системы	Обеспечивает возможность работы с файловой системой для хранения файловых данных абонента, состоит из: • *FileSystemUtils* – определяет методы, реализующие работу с файловой системой сервера.
Модуль БД	Данный модуль предназначен для взаимодействия с базой данных, состоит из: • сущностей (модели) – см. таблицу 9; • *DataBaseUtils* – определяет методы, реализующие бизнес-логику предметной области по работе с базой данных.
Модуль аутентификации	Модуль определяющий работу с сессиями, состоит из: • *Session* – определяет сущность сессии; • *SessionUtils* – определяет методы по работе со списком сессии.
Файлы конфигурации	Содержит набор конфигурационных файлов: • *apiUrl* – определяет список *url*-адресов, для которых будут созданы обработчики *http*-запросов; • *config* – определяет настройки серверной части, строку подключения к БД, используемую сервером ассиметричную пару ключей.
Вспомогательный модуль	Содержит сущность типа *Helper*, которая реализует вспомогательный функционал общего назначения.
Модуль криптографии	Модуль отвечающий за работу с криптографическими средствами защиты информации (СКЗИ), состоит из следующих сущностей: • *AsymCrypto* – реализует функции по работе с асимметричной криптографией;

Окончание таблицы 7

Модуль	Описание
Модуль криптографии (продолжение)	• *SymCrypto* – реализует функции по работе с симметричной криптографией; • *CryptoHelper* – реализует вспомогательные функции по работе с криптографическими объектами; • *CrtyptoUtils* – является точкой входа для работы с СКЗИ, делегируя операции выше указанным сущностям, тем самым выступая промежуточным звеном между бизнес-логикой серверного приложения и вспомогательными функциями по работе с криптографическими алгоритмами.
Модуль шлюзов	Отвечает за взаимодействие с различными сторонними сервисами, состоит из: • *smsGate* – предназначена для работы с смс-шлюзом; • *GatewayUtils* – является точкой входа для работы со сторонними сервисами, делегируя операции выше указанной сущности.

Как правило, выше представленные программные модули имеют общую условную архитектуру, приведенную на рисунке 18 и описываемую далее.

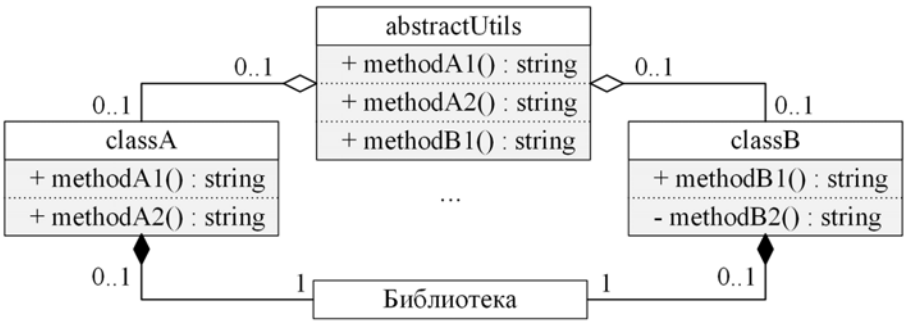

Рисунок 18 – Архитектура абстрактного модуля виде диаграммы классов

Для лучшего понимания, описание приводится с нижнего уровня абстракции, на котором располагается программная библиотека или сторонний сервис. Нижний уровень предоставляет функционал для выполнения определенных тематических задач, например, библиотека реализующая криптографические алгоритмы или доступ к базе данных. Следующий уровень абстракции, представляет слой по работе с функционалом конкретной библиотеки, например, на этом уровне могут определяться операции выполняющие предварительную обработку входных аргументов перед использованием их в библиотечной функции. Верхний уровень абстракции агрегирует созданные сущности на предыдущем уровне, и делегирует им свои обязанности. То есть, верхний уровень эта единая точка входа для работы с модулем, в некоторых случаях может отсутствовать. Сущность, создаваемая на верхнем уровне абстракции, имеет специальное наименование, которое начинается

с ключевых слов, отражающих функциональное предназначение сущности, и заканчивается словом *Utils*.

3.1.2.2 Архитектура клиентской части

Клиентское приложение работает через абонентское устройство, и осуществляет интерактивное взаимодействие с абонентом (пользователем) с помощью графического интерфейса. На рисунке 19 показана схема архитектуры клиентской части.

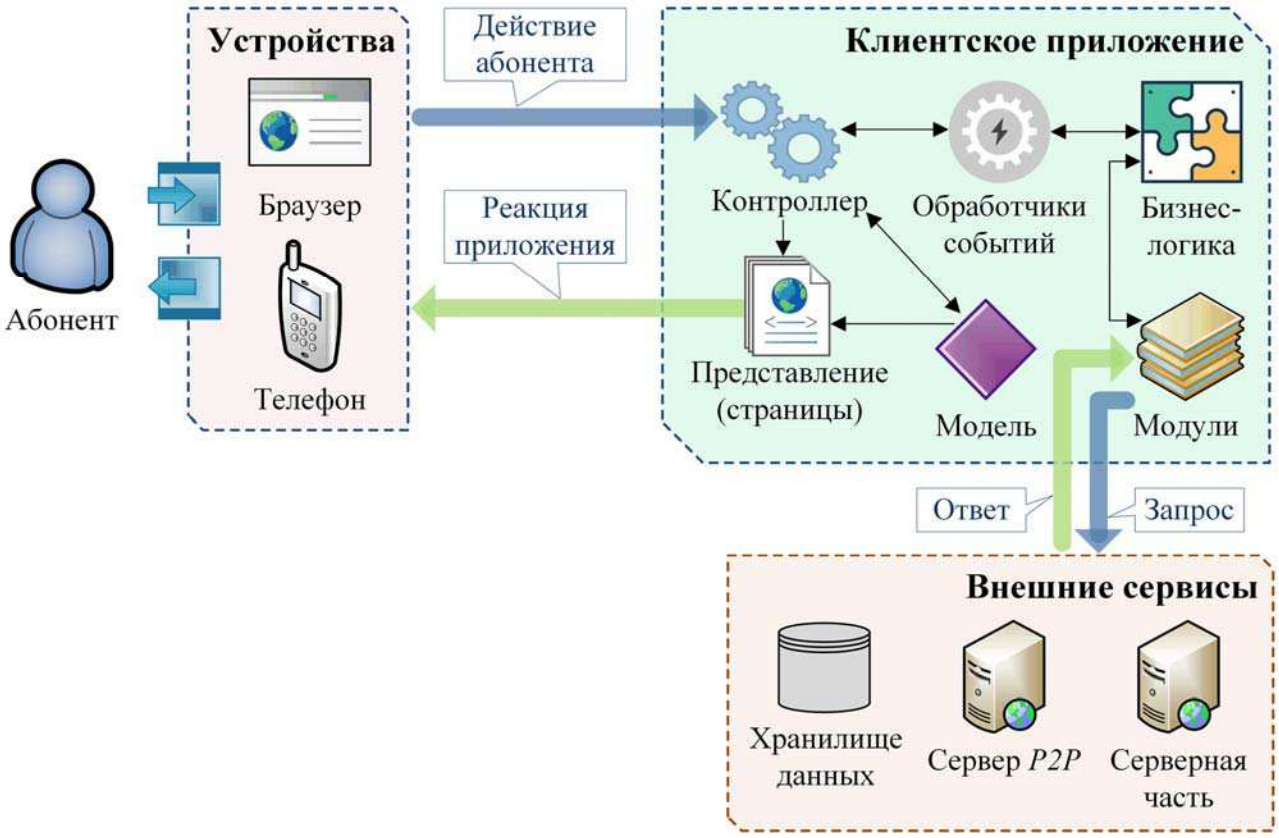

Рисунок 19 – Архитектура клиентского приложения

Из рисунка 19 видно, что основой архитектуры клиентского приложения является шаблон проектирования *MVC* (*Model-View-Controller*) [44], который изолирует бизнес-данные (модель) и бизнес-логику от пользовательского интерфейса (представление) с помощью третьего компонента (контроллер), осуществляющего управление логикой и вводом пользовательских данных, а также координирует модели и представления. Опишем архитектуру клиентского приложения более подробно. В результате взаимодействия абонента с графическим интерфейсом приложения, инициируются события, например, нажатие по кнопке, которые передаются в контроллер на обработку. Контроллер осуществляет обработку событий путем передачи управления обработчику событий вместе с моделью.

После того, как событие будет обработано, контроллер обращается к представлению с запросам на изменение, отметим, что представление может использовать свойства модели. Обработчик событий содержит слой бизнес-логики, который взаимодействует с модулями клиентского приложения для достижения требуемых результатов. Модули клиентского приложения, представлены на рисунке 20, и описаны в таблице 8. Архитектура модуля клиентского приложения аналогична архитектуре модуля серверного приложения – см. раздел 3.1.2.1.

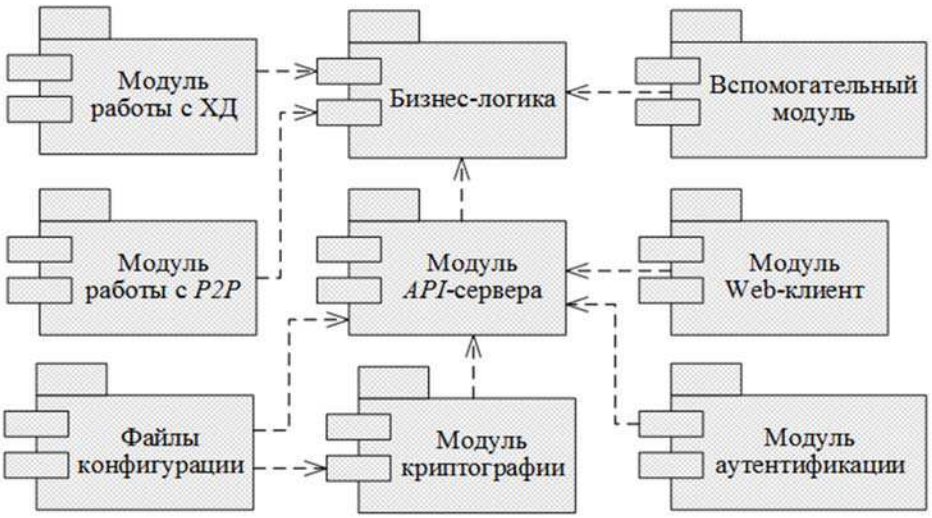

Рисунок 20 – Архитектура клиентского приложения на уровне модулей и их взаимодействия между собой виде диаграммы компонентов

Таблица 8 – Описание модулей и их составных элементов

Модуль	**Описание**
Бизнес-логика	Модуль отвечает за обработку основной логики приложения и взаимодействия между модулями и другими компонентами приложения.
Модуль работы с ХД	Модуль отвечающий за взаимодействие с локальным хранилищем данных, например, встраиваемой базой данных, *web*-хранилищем, и т.д. В данной реализации, модуль отсутствует, поскольку полностью компенсируется существующим функционалом *web*-хранилища типа локальное хранилище (*localStorage*).
Вспомогательный модуль	Содержит сущность типа *ValidationHelper*, которая реализует проверку водимых данных пользователем.
Модуль работы с *P2P*	Модуль состоит из сущности *peerJS*, которая предоставляет прослойку между клиентским приложением и библиотекой, а также организует работу с пиринговой сетью (*Peer-to-Peer, P2P*).
Модуль *API*-сервера	Модуль состоит из сущности *clientApi*, которая определяет взаимодействие с сервером через предоставляемый программный интерфейс (*API*).
Модуль криптографии	Модуль отвечающий за работу с криптографическими средствами защиты информации (СКЗИ), аналогичен модулю, описанному в таблице 7.

Окончание таблицы 8

Модуль	Описание
Файлы конфигурации	Содержит набор конфигурационных файлов: • *apiUrl* – определяет список поддерживаемых *url*-адресов для обращения к *http*-запросам, предоставляемых серверной частью; • *config* – содержит строку подключения к серверной части, и открытый ключ сервера.
Модуль *Web*-клиент	Модуль состоит из *httpClient*, и предоставляет надстройку над общими методами обмена (приема и передачи) данными с ресурсами в сети Интернет, указанными с помощью *URI*.
Модуль аутентификации	Модуль состоит из сущности *session*, которая определяет свойства сессии на стороне клиентской части.

3.1.3 Проектирование архитектуры базы данных

На рисунке 21 представлена логическая схема базы данных (БД) в виде диаграммы «сущность-связь» (*Entity-Relationship*, *IR*) с использованием нотации информационного проектирования (*Information Engineering*, *IE*), а в таблице 9 приводиться описание сущностей.

Таблица 9 – Описание сущностей

Имя сущности	Описание
User	Таблица зарегистрированных пользователей (абонентов) в ИС, состоит из следующих атрибутов: • *userId* – идентификатор записи в БД; • *phoneHash* – хэш-код от мобильного телефонного номера; • *userName* – псевдоним (фамилия и имя) пользователя в ИС; • *passAuth* – код аутентичности абонентского ключа, позволяет осуществить проверку подлинности пользователя путём сравнения введённого им пароля; • *salt* – одноразовая случайная последовательность, используемая при создании симметричного абонентского ключа на основе секретной фразы; • *isOnline* – флаг, указывающий, находится ли пользователь в ИС на текущее время.

Setting	Таблица для хранения настроек аккаунта пользователя в ИС, состоит из следующих атрибутов:
	• *settingId* – идентификатор записи в БД;
	• *userId* – идентификатор пользователя, которому принадлежит настройка;
	• *setting* – запись, представляющая из себя *JSON*-объект, хранящая настройки аккаунта, и имеет следующую структуру:
	`{`
	`"notifyEnabled": true, // показывать уведомления.`
	`"timeLifeOfKeyInHours": 24 // время действия ключей абонента,`
	`// в часах.`
	`}`

Продолжение таблицы 9

Имя сущности	Описание
Contact	Таблица, содержащая список дружественных контактов пользователя в ИС, состоит из следующих атрибутов: • *contactId* – идентификатор записи в БД; • *userId* – идентификатор пользователя, которому принадлежит запись; • *friendUserId* – идентификатор пользователя, который находиться в списке контактов у пользователя с идентификатором *userId* . Заметим, что обратное справедливо: если пользователь *A* дружит с *B*, то пользователь *B* также дружит с *A*.
Dialog	Таблица, содержащая список созданных пользователем диалогов в ИС, состоит из следующих атрибутов: • *dialogId* – идентификатор записи в БД; • *name* – название диалога; • *created* – дата и время создания диалога.
DialogParticipant	Таблица участников (собеседников) диалога, состоит из следующих атрибутов: • *dialogParticipantId* – идентификатор записи в БД; • *dialogId* – идентификатор диалога; • *userId* – идентификатор пользователя, который является участником диалога с идентификатором *dialogId*.

Message	Таблица входящих и исходящих сообщений пользователя, состоит из следующих атрибутов:
	• *messageId* – идентификатор записи в БД;
	• *toUserId* – идентификатор пользователя, которому адресовано данное сообщение;
	• *privateKeyId* – идентификатор закрытого ключа, которым расшифровывается данное сообщение;
	• *isReaded* – флаг, указывающий, прочитано ли сообщение пользователь;
	• *eds* – код (метка) целостности;
	• *sendDate* – дата и время отправки сообщения;
	• *cipheretext* – запись, представляющая из себя шифртекст полученный от *JSON*-объекта, который имеет следующую структуру:
	`{`
	`"plaintext": 'Я дома!', // содержимое сообщения.`
	`"fromUserId": 5, // идентификатор отправителя.`
	`"counter": '324…452', // значение счетчика сообщений.`
	`"messageType": 'text' // тип сообщения.`
	`}`
	Тип сообщения (*messageType*) определяет содержимое сообщения (*plaintext*), состоит из следующих типов:
	• *text* – текст, т.е. *plaintext* содержит текстовую информацию веденую абонентом;
	• *file* – файл, т.е. *plaintext* содержит *JSON*-объект, формата:

Продолжение таблицы 9

Имя сущности	Описание

Message (продолжение)	`{`
	`  "fileKey":  '2g…j', // ключ для расшифровки файла.`
	`  "nameFile": 'fr…h'  // уникальное имя файла в файловой`
	`  // системе сервера.`
	`}`

PrivateKey	Таблица для хранения резервных копий ассиметричных закрытых ключей шифрования пользователя, состоит из следующих атрибутов: • *privateKeyId* – идентификатор записи в БД; • *userId* – идентификатор пользователя, которому принадлежит данный ключ; • *begin* – дата и время с которой началось время жизни ключа; • *end* – дата и время с которой закончиться время жизни ключа; • *privateKey* – запись, представляющая из себя шифртекст закрытого ключа пользовательской (абонентской) пары.
PublicKey	Таблица для хранения резервной копии действующего ассиметричного открытого ключа шифрования пользователя, состоит из следующих атрибутов: • *publicKeyId* – идентификатор записи в БД; • *privateKeyId* – идентификатор на закрытую ключевую пару, которая соответствует данному открытому ключу; • *userId* – идентификатор пользователя, которому принадлежит данных ключ; • *publicKey* – резервная копия, действующего открытого ключа пользователя (абонента), запись представлена виде шифртекста; • *hashPublicKey* – хэш-код, действующего открытого ключа пользователя, по которому проверяется правильность полученного открытого ключа при распределении ключей; • *salt* – одноразовая случайная последовательность, используемая при вычислении хэш-кода *hashPublicKey*.

EventSchema	Таблица, позволяющая уведомлять пользователя о событиях, происходящих относительно него. Состоит из следующих атрибутов:
	• *eventSchemaId* – идентификатор записи в БД;
	• *fromUserId* – идентификатор пользователя, который является инициатором события;
	• *toUserId* – идентификатор пользователя, которому принадлежит данное событие;
	• *eventType* – наименование типа события;
	• *isActual* – флаг, актуальности события;
	• *created* – дата и время создания события;
	• *data* – запись, содержащая дополнительные параметры события, представляется виде *JSON*-объекта, и имеет следующую структуру:
	`{`
	`  "newKeyValue": '342rh…x', // значение ключа в защищенном виде.`
	`  "dialogId": 2  // идентификатор диалога.`
	`}`

Окончание таблицы 9

Имя сущности	Описание
EventType	Справочник типов событий, состоит из следующих атрибутов: • *eventTypeId* – идентификатор записи в БД; • *typeName* – наименование типа события, в ИС приняты следующие типы событий: o *addNewContact* – запрос на добавление пользователя в список контактов; o *addNewParticipant* – уведомление о добавлении пользователя в диалог; o *exchangeKey* – запрос на обмен открытыми ключами между двумя пользователями (абонентами); o *newKey* – уведомление о передаче части открытого ключа.
AuthCredential	Таблица предназначена для регистрации переданных ИС одноразовых кодов по смс-сообщению с целью аутентификации пользователя, состоит из следующих атрибутов: • *authCredentialId* – идентификатор записи в БД; • *smsCodeHash* – хэш-код вычисленный из одноразового смс-кода, отправленного по смс на мобильный номер телефона; • *sessionId* – идентификатор сеанса связи; • *startTime* – дата и время, начала действия высланного смс-кода; • *endTime* – дата и время, окончания действия высланного смс-кода.

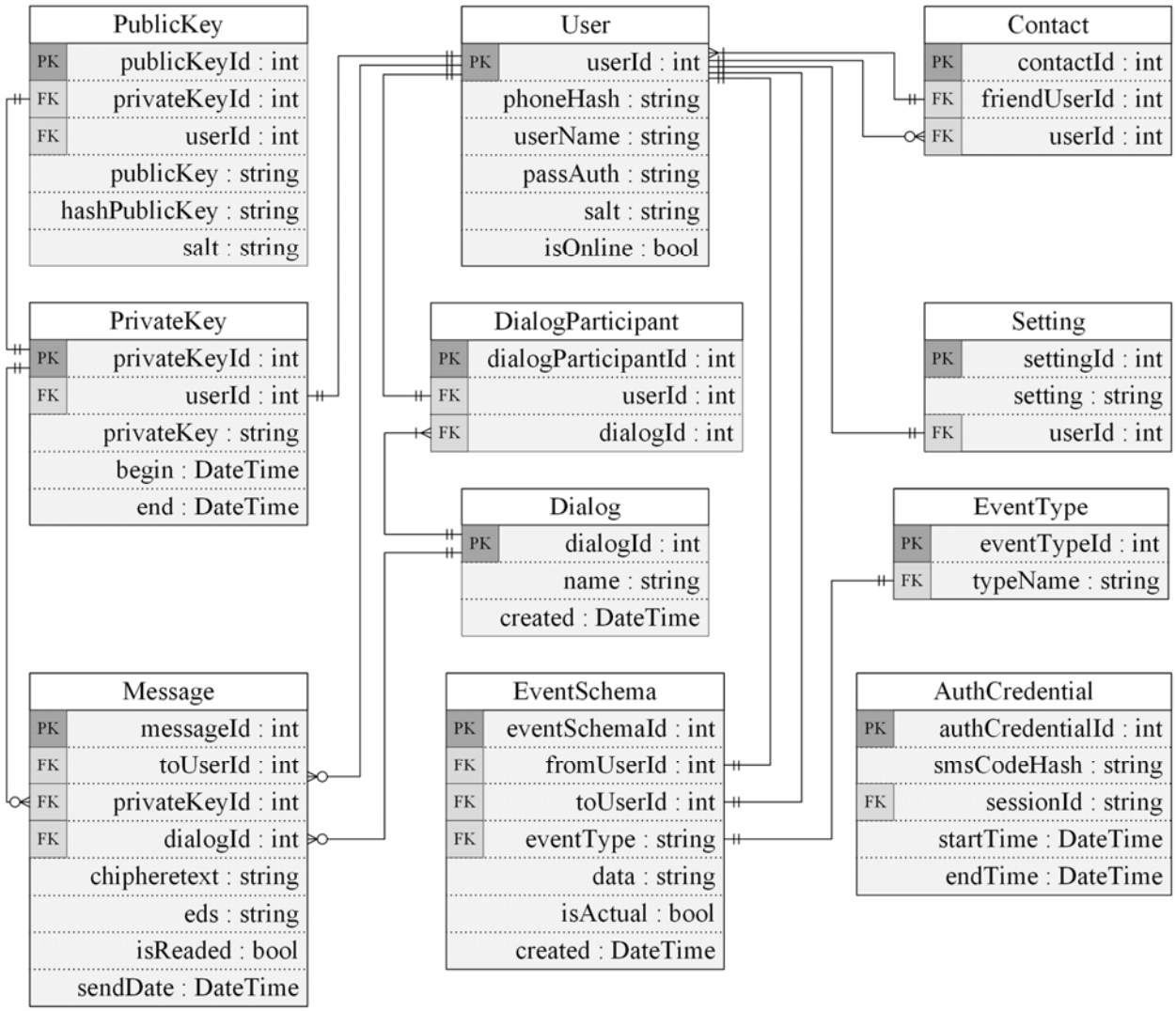

Рисунок 21 – *ER*-диаграмма разрабатываемой ИС в нотации *IE*

3.1.4 Разработка протоколов взаимодействия

3.1.4.1 Протокол взаимодействия «абонент-сервер»

Протокол обеспечивает выполнение следующих сценариев, указанных в таблице 10.

Таблица 10 – Сценарий использования протокола

Сценарий использования	Исполнители	Описание
Вход в систему	Клиентская часть	Клиентская часть посылает серверу запрос на получение одноразовой последовательности, используя для этого хэш-код вычисленный от мобильного номера телефона. Клиентское приложение посылает серверу сообщение, содержащее аутентификационные данные: номер телефона и код аутентичности абонентского ключа.

Продолжение таблицы 10

Сценарий использования	**Исполнители**	**Описание**
Вход в систему (продолжение)	Клиентская часть (продолжение)	После этого клиенту приходит на номер телефона одноразовый смс-код, хэш-код от которого он отсылает на сервер для подтверждения действия.
	Серверная часть	Сервер проверяет правильность этих параметров на соответствие внутренней базе данных. После этого сервер посылает клиенту либо подтверждение о входе, либо отказ. При подтверждении входа, статус клиента в системе переводиться в «Онлайн».
Регистрация в системе	Клиентская часть	Клиентское приложение посылает серверу запрос, содержащий номер телефона. После этого клиенту высылается на номер телефона одноразовый смс-код для подтверждения регистрации. Клиентское приложение посылает серверу сообщение, содержащее аутентификационные данные: хэш-код номера телефона, имя и фамилию пользователя, сгенерированные случайные последовательности, хэш-код одноразового смс-кода, код целостности открытого ключа абонента, код аутентичности абонентского ключа. Более подробно смотрите в раздел 2.3.3.1.1.
	Серверная часть	Сервер проверяет хэш-код одноразового смс-кода, полученного от клиента, на соответствие внутренней базе данных о пользователе. После этого он посылает клиенту либо подтверждение о регистрации, либо отказ.
Создание диалога	Клиентская часть	Клиент посылает запрос серверу на создание диалога, запрос содержит название создаваемого диалога, по умолчанию, название формируется из имен и фамилий участников диалога.
	Серверная часть	Сервер создает диалог в БД и добавляет абонента в качестве участника диалога. После этого он посылает клиенту либо подтверждение об успешно выполненном запросе, либо отказ.
Добавление абонента в диалог	Клиентская часть	Клиент посылает серверу запрос на добавление абонента в качестве участника диалога, используя для этого идентификатор диалога и идентификатор добавляемого абонента.
	Серверная часть	Сервер добавляет абонента в диалог и создает событие в базе данных уведомляющее добавленного абонента об его участии в диалоге. После этого он посылает клиенту либо подтверждение об успешно выполненном запросе, либо отказ.
Добавить/ обновить абонентский ключ	Клиентская часть	Клиент посылает серверу запрос на обновление абонентской пары ключей, используя для этого идентификатор абонента, новые значения шифртекста открытого и закрытого ключей, код целостности

Продолжение таблицы 10

Сценарий использования	**Исполнители**	**Описание**
Добавить/ обновить абонентский ключ (продолжение)	Клиентская часть (продолжение)	открытого ключа, одноразовую последовательность и период действия.
	Серверная часть	Сервер создает, либо изменяет соответствующие записи в БД, а затем возвращает результат выполнения запроса.
Запрос событий	Клиентская часть	Клиент посылает серверу запрос на получение актуальных событий, используя для этого идентификатор пользователя.
	Серверная часть	Сервер формирует список актуальных событий и посылает их клиенту.
Создание события	Клиентская часть	Клиент посылает серверу запрос на создание события, используя для этого идентификатор пользователя в ИС являющегося инициатором события, идентификатор пользователя в ИС являющегося получателем события, идентификатор типа события, и запись, содержащую дополнительные параметры события – см. раздел 3.1.3.
	Серверная часть	Сервер создает соответствующую запись в БД и возвращает результат выполнения запроса.
Обмен сообщениями между абонентами	Клиентская часть	Клиент посылает серверу запрос на доставку сообщения, используя для этого идентификатор абонента-получателя в ИС, шифртекст сообщения, метку целостности, а также идентификатор закрытого ключа абонента-получателя, который соответствует открытому ключу, использованному при зашифровывании сообщения.
	Серверная часть	Сервер создает соответствующую запись в БД и возвращает результат.
Передача файла	Серверная часть – прием файла	Сервер принимает файл, размещает его на файловой системе, и назначает ему новое, уникальное идентификационное наименование. В случае успеха, возвращает клиенту назначенное наименование.
	Клиентская часть – отправитель	Клиент посылает зашифрованный файл серверной части, в ответ получает уникальное идентификационное наименование файла. Затем, отправитель посылает получателю сообщение типа «абонент-абонент», содержащее симметричный ключ для расшифровывания файла, и уникальное идентификационное наименование, по которому получатель может скачать файл.
	Клиентская часть – получатель	Клиент получает от отправителя сообщение, затем запрашивает файл по идентификационному наименованию от сервера.
	Серверная часть – передача файла	Сервер принимает запрос на выгрузку файла, содержащий уникальное идентификационное наименование. Затем, сервер передает файл клиентской

Окончание таблицы 10

Сценарий использования	**Исполнители**	**Описание**
Передача файла (продолжение)	Серверная часть – передача файла (продолжение)	части (получателю). В случае успешной передачи, сервер удаляет файл из файлового хранилища.
Запрос списка контактов	Клиентская часть	Клиент посылает серверу запрос на получение списка контактов пользователя.
	Серверная часть	Сервер формирует этот список и посылает его клиенту.
Запрос истории переписки	Клиентская часть	Клиент посылает запрос серверу, содержащий идентификатор диалога, количественное ограничение и (или) временной интервал. По умолчанию, количественное ограничение равно 10. Последующая часть сообщений загружается по востребованию, для этого используется временной промежуток, шаг промежутка равен 1 месяцу.
	Серверная часть	Сервер извлекает сообщения из базы данных, принадлежащие указанному диалогу, заданному временному интервалу и (или) количественному ограничению. Если ограничения не заданы, то возвращаются все сообщения из истории. После этого сервер пересылает эти сообщения обратно клиенту.
Запрос на удаление сообщения	Клиентская часть	Клиент посылает сообщение серверу, содержащее идентификатор сообщения.
	Серверная часть	Сервер удаляет сообщение из базы данных с указанным идентификатором. После этого сервер уведомляет клиента о состоянии выполненной операции.
Запрос на удаление истории переписки	Клиентская часть	Клиент посылает серверу запрос на удаление всей истории переписки пользователя, содержащий идентификатор диалога.
	Серверная часть	Сервер удаляет все сообщения из базы данных. После этого сервер уведомляет клиента о состоянии выполненной операции.
Поиск пользователя	Клиентская часть	Клиент посылает сообщение серверу с запросом, содержащим хэш-код номера телефона.
	Серверная часть	Сервер ищет соответствие в базе данных. Затем пересылает результат поиска клиенту.
Получение информации о пользователе	Клиентская часть	Клиент посылает сообщение серверу с запросом, содержащим идентификатор пользователя.
	Серверная часть	Сервер ищет пользователя по его идентификатору в БД. Затем пересылает открытую информацию о найденном пользователе клиенту.
Выход клиента из системы	Клиентская часть	Клиент посылает уведомление серверу о своем выходе.
	Серверная часть	Сервер помечает состояние пользователя как в не сети. И завершает сеанс связи, удалив сессию.

Сообщения (*http*-запросы), которыми обмениваются клиент и сервер, можно условно поделить на несколько тематических групп, представлены в таблице 11, а также на категории в соответствии с разделом 2.3.3.3 второй главы, представлены в таблице 12.

Таблица 11 – Сообщения типа «клиент-сервер»

Тип сообщения	**Инициатор сообщения**	**Описание**
auth – работа с аутентификацией		
logIn	Клиентская часть	Запрос входа в систему. Сообщение должно содержать аутентификационные данные: см. раздел 2.3.3.1.2.
	Серверная часть	Сообщение подтверждение входа, содержит токен (аутентификационный билет) и идентификатор абонента в ИС.
signUp	Клиентская часть	Запрос регистрации абонента в ИС. Сообщение должно содержать сведения об пользователе и аутентифицирующие данные: см. раздел 2.3.3.1.1.
	Серверная часть	Подтверждение регистрации.
logOut	Клиентская часть	Запрос на выход из системы.
	Серверная часть	Отказ во входе в систему, либо уведомление о завершении работы сервера.
existPhone	Клиентская часть	Запрос на идентификацию телефонного номер в ИС по БД. Сообщение должно содержать хэш-код от мобильного номера телефона, вычисленного по хэш-функции *H*.
	Серверная часть	Подтверждение наличия телефонного номера в ИС.
sendSmsCode	Клиентская часть	Запрос на отправку смс-кода на мобильный номер телефона абонента. Сообщение должно содержать мобильный номер телефона.
	Серверная часть	Подтверждение отправки смс-кода.
createSession	Клиентская часть	Запрос на создание защищенного соединения. Сообщение должно содержать симметричный ключ шифрования сессии.
	Серверная часть	Сообщение, содержащее идентификатор сессии.
checkPasswordAuth	Клиентская часть	Запрос на проверку соответствия посылаемого кода аутентичности абонентского ключа, коду, хранящемуся в БД. Сообщение должно содержать, идентификатор абонента, код аутентичности абонентского ключа, номер мобильного телефона абонента.
	Серверная часть	Подтверждения о соответствии, в случае успеха, посылает смс-код на мобильный номер телефона абонента.

Продолжение таблицы 11

Тип сообщения	Инициатор сообщения	Описание
getUserSalt	Клиентская часть	Запрос на получение случайной последовательности, которая была использована для создания симметричного ключа абонента. Сообщение должно содержать идентификатор абонента.
	Серверная часть	Сообщение, содержащее случайную последовательность.
settings – работа с настройками аккаунта		
getSetting	Клиентская часть	Запрос настройки аккаунта абонента. Сообщение, содержащее идентификатор абонента.
	Серверная часть	Сообщение, содержащее настройку абонентского аккаунта.
editSetting	Клиентская часть	Запрос на изменение настройки аккаунта абонента. Сообщение, содержащее настройку аккаунта, и идентификатор абонента.
	Серверная часть	Подтверждение изменения настройки абонентского аккаунта.
editPhone	Клиентская часть	Запрос на изменение номера мобильного телефона связанного с аккаунтом абонента. Сообщение, содержащее новый номер мобильного телефона, смс-код и идентификатор абонента.
	Серверная часть	Подтверждение изменения абонентского аккаунта.
users – работа с абонентской информацией		
getUserInfo	Клиентская часть	Запрос открытых сведений о пользователе. Сообщение должно содержать идентификатор пользователя.
	Серверная часть	Сообщение, содержащее открытую информацию об пользователе.
contacts – работа с контактами		
addContact	Клиентская часть	Запрос на добавление абонента в список дружественных контактов.
	Серверная часть	Подтверждение добавления аккаунта в список дружественных контактов.
getContacts	Клиентская часть	Запрос списка дружественных контактов. Сообщение должно содержать идентификатор абонента.
	Серверная часть	Сообщение, содержащее список дружественных контактов.
deleteContact	Клиентская часть	Запрос на удаление выбранного абонента из списка дружественных контактов.
	Серверная часть	Подтверждение удаления абонента из списка дружественных контактов.

Продолжение таблицы 11

Тип сообщения	**Инициатор сообщения**	**Описание**
dialogs – работа с диалогами		
getDialogs	Клиентская часть	Запрос списка диалогов, в которых абонент является участником (собеседником). Сообщение должно содержать идентификатор абонента.
	Серверная часть	Сообщение, содержащее список диалогов, в которых абонент является участником.
createDialog	Клиентская часть	Запрос на создание диалога. Сообщение должно содержать название диалога и идентификатор абонента.
	Серверная часть	Подтверждение создания диалога.
addUserToDialog	Клиентская часть	Запрос на добавление абонента в диалог, в качестве собеседника. Сообщение должно содержать идентификаторы диалога и добавляемого абонента.
	Серверная часть	Подтверждение добавления абонента в диалог.
removeUserFromDialog	Клиентская часть	Запрос на удаление абонента из диалога. Сообщение должно содержать идентификаторы диалога и удаляемого абонента.
	Серверная часть	Подтверждение удаления абонента из диалога.
getDialogParticipants	Клиентская часть	Запрос списка собеседников диалога. Сообщение должно содержать идентификаторы диалога и абонента в ИС.
	Серверная часть	Сообщение, содержащее список собеседников диалога.
messages – работа с сообщениями		
getMessages	Клиентская часть	Запрос списка сообщений (истории переписки) из диалога. Сообщение должно содержать идентификаторы диалога и абонента, а также параметры фильтров: временной промежуток, количественное ограничение и (или) флаг актуальности (не просмотренные) сообщений.
	Серверная часть	Сообщение, содержащее список сообщений. В случае, если идентификаторы диалога и абонента не указаны, то возвращается пустой список. Если параметр фильтра не указан, то возвращаются данные без учета данного фильтра. В случае, если все параметры фильтров не указаны, то возвращается весь список сообщений из диалога.
sendMessage	Серверная часть	Подтверждение пересылки сервером сообщения адресату.
	Клиентская часть	Посылка сообщения абонентом другому абоненту. Сообщение должно содержать идентификатор абонента-получателя, шифртекст сообщения, код целостности и идентификатор

Продолжение таблицы 11

Тип сообщения	Инициатор сообщения	Описание
sendMessage (продолжение)	Клиентская часть (продолжение)	закрытого ключа абонента-получателя, с помощью которого можно расшифровать отправленное сообщение.
deleteMessages	Клиентская часть	Запрос на удаление сообщений из списка сообщений абонента. Сообщение, содержит идентификатор абонента, список идентификаторов сообщений, помеченных на удаление.
	Серверная часть	Подтверждение удаления сообщений абонента. В случае, если список идентификаторов сообщений пусть, удаляются все сообщения из списка сообщений абонента.
files – работа с файлами		
uploadFile	Клиентская часть	Запрос на передачу файла. Сообщение должно содержать файл и идентификатор абонента.
	Серверная часть	Сообщение, содержащее уникальное имя файла.
downloadFile	Клиентская часть	Запрос на загрузку файла. Сообщение должно содержать уникальное имя файла и идентификатор абонента.
	Серверная часть	Сообщение, содержащее запрашиваемый файл.
keys – работа с криптографическими ключами		
addKeyPair	Клиентская часть	Запрос на добавление и использование новой ключевой пары абонента, взамен действующей. Сообщение должно содержать идентификатор абонента, шифртекст ключевой пары, код целостности от нового открытого ключа, одноразовую последовательность и период действия новой пары. По умолчанию, период действия равен 24 часа с момента создания ключевой пары.
	Серверная часть	Подтверждение добавления ключевой пары, с уведомлением участников диалога.
getKeyPair	Клиентская часть	Запрос сведений о действующей ключевой паре абонента. Сообщение должно содержать идентификатор абонента.
	Серверная часть	Сообщение, содержащее шифртекст запрашиваемой ключевой пары абонента, код целостности открытого ключа абонента, одноразовую последовательность и период действия ключевой пары.
getSaltAndHashOf PublicKey	Клиентская часть	Запрос кода целостности действующего открытого ключа абонента, и использованную случайную последовательность при вычислении данного кода целостности. Сообщение должно

Продолжение таблицы 11

Тип сообщения	Инициатор сообщения	Описание
getSaltAndHashOf PublicKey (продолжение)	Клиентская часть (продолжение)	содержать идентификатор абонента.
	Серверная часть	Сообщение, содержащее запрашиваемый код целостности действующего открытого ключа абонента, и использованную случайную последовательность при вычислении кода целостности открытого ключа.
getPrivateKey	Клиентская часть	Запрос шифртекста закрытого ключа абонента. Сообщение должно содержать идентификатор абонента и параметры фильтра: идентификатор ключа, либо период действия.
	Серверная часть	Сообщение, содержащее запрашиваемый шифртекст закрытого ключа абонента. В случае, если идентификатор абонента и оба параметра фильтра не указаны, то возвращается пустое значение.
getAllPrivateKeys	Клиентская часть	Запрос списка шифртекстов закрытых ключей абонента, действующих ранее. Сообщение должно содержать идентификатор абонента и параметры фильтров: список идентификаторов ключей и (или) список временных периодов.
	Серверная часть	Сообщение, содержащее запрашиваемый список шифртекстов закрытых ключей абонента. В случае, если идентификатор абонента и оба параметра фильтра не указаны, то возвращается пустой список. Если параметр фильтра не указан, то возвращаются данные без учета данного фильтра.
events – работа с событиями		
getEvents	Клиентская часть	Запрос списка событий абонента. Сообщение должно содержать идентификатор абонента и параметры фильтров: флаг актуальности (не просмотренные) событий, список типов событий (см. раздел 3.1.3) и (или) список временных периодов.
	Серверная часть	Сообщение, содержащее список событий абонента. В случае, если идентификатор диалога абонента не указан, то возвращается пустой список. Если параметр фильтра не указан, то возвращаются данные без учета данного фильтра. В случае, если все параметры фильтра не указаны, то возвращаются все события абонента.
createEvent	Клиентская часть	Запрос на создание события для абонента. Сообщение должно содержать идентификаторы

Окончание таблицы 11

Тип сообщения	Инициатор сообщения	Описание
createEvent (продолжение)	Клиентская часть (продолжение)	абонентов отправителя и получателя события, тип события (см. раздел 3.1.3) и дополнительные параметры события (см. раздел 3.1.3).
	Серверная часть	Подтверждение создания события.

Таблица 12 – Сообщения типа «абонент-сервер» разделенные на категории доступа к информации

Категория доступа	Сообщения
Без сессионная	*createSession.*
Сессионная без токена	*logIn*, *signUp*, *existPhone*, *getUserSalt*, *sendSmsCode*, *checkPasswordAuth*.
Сессионная с токеном	Остальные.

3.1.4.2 Протокол взаимодействия «абонент-абонент»

Сообщения, которыми обмениваются абоненты в сети *P2P*, можно разделить на несколько типов, которые описаны в таблице 13.

Таблица 13 – Сообщения типа «абонент-абонент»

Тип сообщения	Инициатор сообщения	Описание

exchangeTempKey	Абонент-отправитель	Передача разового ассиметричного ключа абоненту-получателю для создания защищенного соединения, имеет следующую структуру:
		`{`
		`  "publicTempKey": 'x19…d', // эфемерный открытый ключ.`
		`  "peerId": '3' // идентификатор абонента в сети P2P,`
		`  // аналогичен идентификатору в ИС.`
		`}.`

exchangePartOfUserKey	Абонент-отправитель	Передача зашифрованной части открытого ключа абонента, имеет следующую структуру:
		`{`
		`  "publicPartOfKey": 'x19…d', // часть открытого ключа.`
		`  "sequence": '1', // номер части ключа.`
		`  "peerId": '3'`
		`}.`

3.1.5 Используемые инструментальные средства

Предлагаемая модель информационной системы (ИС) для безопасного обмена информацией в компьютерных сетях общего доступа была реализована в виде программного комплекса «*ruMessanger*» [45]. Данный комплекс базируется на открытых и платформо-независимых технологиях *Apache Cordova* и *Node.js*, что, ко всему прочему, позволяет снизить вероятность наличия недекларируемых возможностей, которые могут присутствовать в проприетарных решениях. Программный комплекс был реализован с использованием языка

программирования высокого уровня *JavaScript*, что позволило использовать единую кодовую базу при разработке модулей клиентской и серверной частей программного комплекса, при разработке клиентской части также использовались языки *HTML* и *CSS*. В качестве системы управления базами данных (СУБД) выбрана *MongoDB*, поставляемая в открытых исходных кодах, и имеющая отличные возможности по масштабированию [46] при последующей кластеризации ИС. Взаимодействие абонентов с ИС осуществляется посредством клиентского приложения для мобильного устройства или с помощью браузера. Коммуникация между клиентской частью и серверной осуществляется посредством протокола *http*. Безопасность информационного обмена обеспечивается авторскими механизмами защиты информации, рассмотренными ранее. Рассмотрим более подробно инструментальные средства и компоненты, используемые при разработке серверной и клиентской частей.

3.1.5.1 Общие инструментальные средства

При разработке обеих частей программного комплекса были использованы инструменты, приведенные в таблице 14.

Таблица 14 – Общие инструменты разработки

Наименование инструмента	Версия	Описание
WebCrypto GOST	1.76.0	Библиотека криптографических алгоритмов, реализующая *WebCrypto API* и инфраструктуру открытых ключей (*PKI – Public Key Infrastructure*) для криптографических стандартов ГОСТ.
*JSDoc*3	3.5.0	Генератор документации в *HTML*-формате из комментариев исходного кода на *JavaScript*.
Git	2.9.3	Система контроля версий.

3.1.5.2 Инструментальные средства для разработки серверной части

Как уже было сказано, разработка частей приложения, в том числе серверной части, осуществлялась на языке программирования *JavaScript*. В качестве стандарта для языка *JavaScript* использовался *ES*6 (*ECMAScript*-6), за его новые возможности и синтаксическую выразительность. В таблице 15 перечислены и описаны используемые инструменты и сторонние компоненты при разработке серверной части.

Таблица 15 – Используемые средства при разработке серверной части

Наименование инструмента	Версия	Описание
Sublime Text 3	сборка 3126	Инструмент для редактирования исходного кода и навигации по нему.

Окончание таблицы 15

Наименование инструмента	Версия	Описание
Node.js	6.2.1	Программная платформа, позволяющая выполнять *JavaScript* код вне браузера (на *Backend*) благодаря виртуальной машине *V8*, которая компилирует *JavaScript* в машинный код.
Node-Inspector	0.12.8	Инструмент для отладки *Node.js* приложений на *WebKit* браузерах.
Express	4.14.0	Программный модуль для *Node.js*, используемый для маршрутизации, т.е. связывания *url*-адреса с логикой маршрута.
Body-Parser	1.15.2	Программный модуль для *Node.js* являющийся промежуточным звеном в обработки тела запросов.
Babel	5.0	Программный модуль для *Node.js*, позволяющий использовать стандарт *ES6* на платформах, которые на данный момент не поддерживают его. То есть это транспайлер, переписывающий *JavaScript* код нового стандарта *ES6* в код предыдущего стандарта *ES5*.
MongoDB	3.2.7	Документированная система управления базами данных (СУБД) в которой, структурированные данные хранятся не в виде таблиц, а в виде документов. Документы кодируются в формате *BSON* (двоичная форма *JSON*).
Mongoose	4.6.0	Программный модуль объектного моделирования для *Node.js*, обеспечивающий поддержку объектно-реляционного отображения (*Object-Relation Mapping*, *ORM*), т.е. предоставляет высокоуровневую абстракцию доступа к базе.
RoboMongo	0.9.0	Инструмент с графическим интерфейсом для управления базой данных *MongoDB*.

3.1.5.3 Инструментальные средства для разработки клиентской части

При разработке клиентской части под *Apache Cordova*, по умолчанию использовался стандарт *ES5* для языка *JavaScript*. Это связанно с тем, что компоненты *WebView* мобильных устройств на момент разработки клиентской части не поддерживали стандарт *ES6*, а использование *Babel* могло негативно сказаться на производительности клиентской части. В таблице 16 приведен перечень инструментов и компонентов, используемых при разработке клиентской части программного комплекса.

Таблица 16 – Используемые средства при разработке клиентской части

Наименование инструмента	Версия	Описание
NetBeans	8.1	Интегрированная среда разработки.
Ionics	2.0.0	Библиотека мобильных иконок.
PeerJS	0.3.14	Организует пиринговую (оверлейную, *peer-to-peer*) сеть.

Окончание таблицы 16

Наименование инструмента	Версия	Описание
Apache Cordova	6.2.1	Программная платформа, позволяющая выполнять и разрабатывать мобильные приложения с использованием *JavaScript*, *HTML* и *CSS*.
Framework 7	1.4.2	Фреймворк, предназначенная для создания элементов пользовательского интерфейса в «родном» (*native*) стиле для *iOS* и *Android*.
RequireJS	2.2.0	Модульный загрузчик файлов формата .js, использующий асинхронную модель подгрузки зависимостей *AMD* (*Asynchronous module definition*, асинхронное определение модуля).
Handlebars	4.0.5	Шаблонизатор для создания шаблонов страниц, делает возможным вставку шаблонов в *HTML*-страницы, которые будут обрабатываться с использованием реальных данных.
Lodash	4.6.1	Библиотека, облегчающая работу с массивами в *JavaScript*.
Json	0.0.1	Плагин для *RequireJS*, позволяет загружать файлы формата *json*.
text	2.0.14	Плагин для *RequireJS*, позволяет загружать текстовые ресурсы, т.е. файлы формата *html*, *css*, и т.д.
hbs	0.1.1	Плагин для *RequireJS*, который взаимодействует с плагином *text*. Плагин позволяет загрузить шаблон и выполнить его предварительную обработку.

3.1.5.4 Конвенция программирования

Для наименования используется английский язык (не транслит) в нотации *lowerCamelCase*. Имена сущностей и переменных задаются существительным, а имена функций – глаголом. Комментирование исходного кода ведется на русском или английском языках в нотации *JSDoc*3.

3.2 Тестирование программного продукта

Результатом разработки стал программный комплекс «*ruMessanger*» [45], который позволяет обмениваться конфиденциальной информацией в компьютерных сетях общего доступа. В процессе реализации программного комплекса, а также после завершения этапа реализации, для повышения качества разрабатываемой ИС осуществлялось автоматизированное тестирование с целью проверки корректности работы программного комплекса как на уровне отдельных модулей, логических структур, так и в рамках системы в целом. Помимо этого, целью тестирования также является проверка удобства использования, выявления ресурсоемких и затратных по времени исполнению участков программного кода. Для этого использовались следующие виды тестирования: модульное тестирование,

функциональное тестирование, тестирование производительности серверной части, тестирование удобства пользования.

В связи с объемностью проведенной работы, далее будет приведено описание процесса тестирования программного комплекса на примере конкретного компонента ИС, с целью демонстрации процесса проведения тестирования.

Нельзя не отметить, что для повышения качества разработки программного комплекса был использован принцип «пусть падает» (*«Let it crash»*), который основан на том, что исключительные ситуации не обрабатывались, а в случае их возникновения программа перезапускалась с выводом ошибки. Такой подход позволил оперативно выявлять места синтаксических, лексических и логических ошибок.

3.2.1 План тестирования

Целью составленного плана тестирования является описание процесса тестирования, что позволяет получить представление о проведенных мероприятиях по тестированию программного комплекса на соответствия требованиям описанных в приложении А.

3.2.1.1 Средства и порядок испытаний

Во время испытаний используются следующие технические и программные средства:

- программные средства:
  - *web*-сервер, например, *Embedded Lightweight*;
  - браузеры *Google Chrome* и *Mozilla Firefox*;
  - исходные коды программного комплекса;
- технические средства:
  - ЭВМ в количестве двух единиц с сетевой платой для создания локальной вычислительной сети (*LAN*);
  - сетевой кабель, типа *Ethernet*;
  - клавиатура и мышь.

Порядок проведения испытаний состоит из следующих этапов:

1. ознакомление с документацией о разработанной ИС;
2. ознакомление с порядком проведения испытаний;
3. ознакомление с результатами автоматизированных тестов;
4. проверка основных сценариев ИС, описанных в приложении А.

Переход к следующему этапу возможен только при успешном прохождении предыдущего этапа испытаний. В случае возникновении ошибки, при прохождении сценария, испытания прекращаются. Проведение испытаний завершается выставлением оценки за выполненную работу в виде процента. Оценка «100%»

ставиться за успешное прохождение всех тестов, если ни один тест не прошел успешно – «0%».

3.2.1.2 Методы испытаний

Тестирование может производиться как вручную, т.е. методом «неформального» тестирования с позиции конечного пользователя программного комплекса, так и автоматизировано. Для этого использовались следующие виды тестирования.

3.2.1.2.1 Модульное тестирование

Цель тестирования. Выявить функциональные ошибки на уровне модулей ИС, устранить соответствующие дефекты в них, удостовериться в соответствии требованиям каждого отдельного модуля перед тем, как будет произведена его интеграция.

Описание процесса тестирования. Производиться разработка тестового окружения для каждого модуля, определяются входные данные, и описываются конкретные тестовые примеры с ожидаемым результатом.

Критерий успешности. Все запланированные области протестированы, а также все найденные ошибки и замечания зафиксированы.

3.2.1.2.2 Функциональное тестирование

Цель тестирования. Выявить функциональные ошибки, основные на взаимодействии функционалов путем реализации нетривиальных сценариев.

Описание процесса тестирования. Производиться условное разделение приложения на функционалы, на основании этого разделения производиться целенаправленное тестирование: регистрация и аутентификация в ИС, получение списка контактов и диалогов, обмен сообщениями с абонентом, работа с настройками аккаунта, выход из ИС.

Критерий успешности. Все запланированные области протестированы, а также все найденные ошибки и замечания зафиксированы.

3.2.1.2.3 Нагрузочное тестирование

Цель тестирования. Определить максимальную нагрузку на ИС при которой, ИС способна функционировать в соответствии с требованиями к производительности.

Описание процесса тестирования. Тестируемый объект (например, серверная часть) развертывается на отдельной ЭВМ с предоставлением всех ресурсов тестируемому объекту, не считая ресурсы на работу операционной системы, т.е. создается рабочая среда. По средством сетевой платы и кабеля создается локальная вычислительная сеть, связывающая рабочую среду с инструментом для проведения нагрузочного тестирования. Далее, инструмент

тестирования, постепенно, создает нагрузку (например, виде *http*-запросов) на объект тестирования, и регистрирует информацию о производительности тестируемого объекта, в том числе следующие системные характеристики: загрузка ОЗУ, ЦПУ, жесткого диска. Процесс нагрузочного тестирования заканчивается тогда, когда производительность тестируемого объекта перестает отвечать заданным требованиям производительности.

Критерий успешности. Объект тестирования доведен до состояния при котором, его производительность не удовлетворяет требованиям технического задания (приложение А), и вся снятая информация о системных характеристиках и производительности зафиксирована.

3.2.1.3 Основные этапы тестирования проекта

Этапы работы над проектом приведены в таблице 17.

Таблица 17 – Этапы работы

Этап	Дата начала	Дата завершения
Написание тест плана.	29.02.2016	07.03.2016
Проведение систематического модульного тестирования.	07.03.2016	01.04.2016
Функциональное тестирование.	01.04.2016	01.06.2016
Тестирование производительности серверной части.	10.02.2016	10.03.2017

3.2.2 Модульное тестирование

В качестве объекта тестирования был выбран общий криптографический модуль, реализующий промежуточный слой работы между программным комплексом и криптографической библиотекой *WebCrypto GOST* [33]. На базе данного модуля разрабатывалась как система безопасности комплекса (генерация ключей, обмен ключами, аутентификация сообщений, и т.д.), так и протокол взаимодействия между клиентом и сервером. От работы данного модуля зависит как безопасность, так и правильность работы всего программного комплекса в целом.

Целью тестирования являются проверки:

– правильности взаимодействия со сторонними компонентами и библиотеками;

– правильности логики работы реализованного модуля при различных входных данных.

3.2.2.1 Структура тестируемого модуля

Структура данного модуля была описана в разделе 3.1.2.1, главы 3.

3.2.2.2 Используемые компоненты

Для создания юнит-тестов применяется среда юнит-тестирования *QUnit* [47] разработанная разработчиками известного фреймворка *jQuery* для тестирования *JavaScript* кода. Модульность тестов организуется с помощью загрузчика *RequireJS*.

3.2.2.3 Стратегия выбора входных данных

Входные тестовые данные, согласно теории предложенной Гленфордом Майерсом (*Glenford Myers*) [48], делаться на два класса эквивалентности: допустимый класс эквивалентности – допустимые («хорошие») входные данные, и недопустимый класс эквивалентности – все остальные, недопустимые («плохие») входные данные. Такое разделения позволит более комплексно и понятно построить юнит-тесты, что скорее всего, повысит вероятность обнаружения ошибки.

3.2.2.4 Организация структуры тестов и их расположение

Структура тестового модуля (*cryptoUnit*) с юнит-тестами, организующего тестирование криптографического модуля, приведена на рисунке 22 и организованна по аналогии с тестируемым объектом.

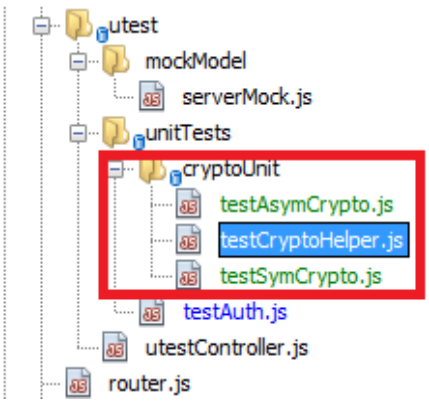

Рисунок 22 – Организация модуля *cryptoUnit*.

Архитектура файла, содержащего юнит-тесты при условии использования *RequrieJS* выглядит следующим образом, как показано на рисунке 23.

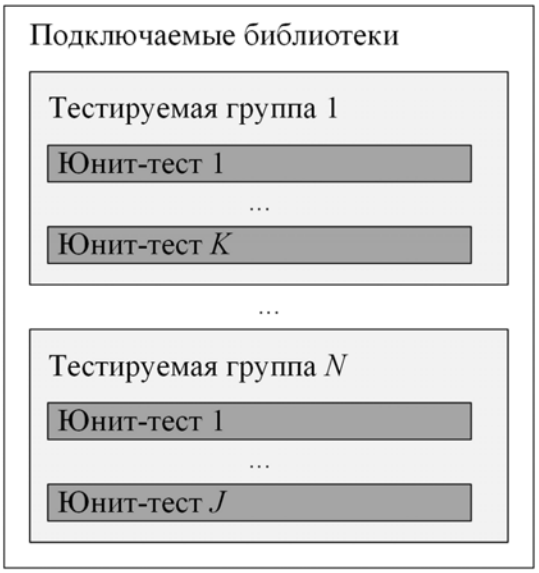

Рисунок 23 – Архитектура файла, содержащего юнит-тесты.

Следует отметить, что *QUnit* позволяет выносить юнит-тесты за пределы группы. Ознакомиться с содержанием всех файлов криптографического модуля, можно с помощью исходных файлов проекта. В листинге Б.1, приложения Б, демонстрируется примеры написанных юнит-тестов для сущности *SymCrypto*, реализующей функции по работе с симметричной криптографией.

3.2.2.5 Результаты тестирования

В результате проведенного тестирования были обнаружены и исправлены ошибки, возникшие по причине того, что при написании программного кода был учтен не весь класс недопустимых («плохих») входных данных, а также по невнимательности разработчика. Ниже, на рисунках 24-26 представлены снимки экрана репозитория, где можно увидеть основные моменты изменения исходного кода программных классов из криптографического модуля, до и после проведения автоматизированного юнит-тестирования.

```
30  30      // Метод формирования ЭЦП
31  31      AsymCrypto.prototype.sign = function(message, privateKey) {
32      -       message = gostCoding.Chars.decode(message);
33      -       var Hex = gostCoding.Hex;
        32  +       if (!privateKey) {
        33  +           return null;
        34  +       }
        35  +
        36  +       var data = typeof message === 'string' ? Hex.decode(message, true) : null;
        37  +       if (!data) {
        38  +           return null;
        39  +       }
        40  +
34  41            if (algorithm.ukm) {
35  42                algorithm.ukm = Hex.decode(algorithm.ukm, true);
36  43            }
37      -       var data = typeof message === 'string' ? Hex.decode(message, true) : message;
38  44
39  45            privateKey = Hex.decode(privateKey, true);
40      -
41  46            var signature = Hex.encode(cipher.sign(privateKey, data), true);
        47  +
42  48            return signature;
43  49        };
```

Рисунок 24 – Результат исправлений метода *sign* класса *AsymCrypto* после юнит-тестирования.

```
45  51      // Метод проверки ЭЦП
46      -   AsymCrypto.prototype.verify = function(signature, publicKey) {
47      -       var Hex = gostCoding.Hex;
        52  +   AsymCrypto.prototype.verify = function(message, signature, publicKey) {
        53  +       if (!signature || !publicKey) {
        54  +           return false;
        55  +       }
        56  +
        57  +       var data = typeof message === 'string' ? Hex.decode(message, true) : null;
        58  +       if (!data) {
        59  +           return false;
        60  +       }
        61  +
48  62            if (algorithm.ukm) {
49  63                algorithm.ukm = Hex.decode(algorithm.ukm, true);
50  64            }
51      -       var data = typeof signature === 'string' ? Hex.decode(signature, true) : signature;
52      -
        65  +
53  66            publicKey = Hex.decode(publicKey, true);
54  67
55  68            var result = cipher.verify(publicKey, Hex.decode(signature, true), data);
56  69            return result;
57  70        };
58  71
59  72        return AsymCrypto;
60  73    });
```

Рисунок 25 – Результат исправлений метода *verify* класса *AsymCrypto* после юнит-тестирования.

```
21  21      /**
22  22       * Конструктор
23  23       * @constructor
24  24       */
25  25      function SymCrypto() {
26          -       if (algorithm.iv) {
        26  +       if (algorithm.iv && typeof algorithm.iv === 'string') {
27  27              algorithm.iv = gostCoding.Hex.decode(algorithm.iv);
28  28          }
29  29          cipher = new GostCipher(algorithm);
30          -       hex = gostCoding.Hex;
31  30      }
```

Рисунок 26 – Результат исправления конструктора класса *SymCrypto* после юнит-тестирования.

Поясним изменения, показанные на рисунке 26. Во время проведения юнит-тестирования, возникла ситуация, при которой значение свойства *iv* (вектор инициализации), повторно создаваемого экземпляра класса *SymCrypto*, оказалось инициализированным и преобразованным к типу *ArrayBuffer*, что приводило к ошибке. Поскольку, свойство *iv* инициализировано, то условный оператор (строка 26) истинен, а значит, что срабатывает метод *decode*, ожидающий параметр типа *String*. Следовательно, в библиотеке *WebCrypto GOST* генерировалось исключение о несовместимости типов. Также причиной возникновения данной ошибки, стала неправильная организация объявления свойств *algorithm*, *Hex*, *gostCoding* и *cipher*. Эти свойства и конструктор класса *SymCrypto* (строка 25) определяются на равном уровне видимости в загрузчике *RequireJS*, в то время как свойства должны иметь область видимости на один уровень ниже, чем у конструктора *SymCrypto*. Выше описанную ошибку, удалось обнаружить только после проведения тестовых испытаний, которые порождали асинхронный режим работы с классом *SymCrypto*. На рисунке 27 показаны зафиксированные изменения в криптографическом модуле, на примере класса *CryptoHelper*, устраняющие выше описанную ошибку.

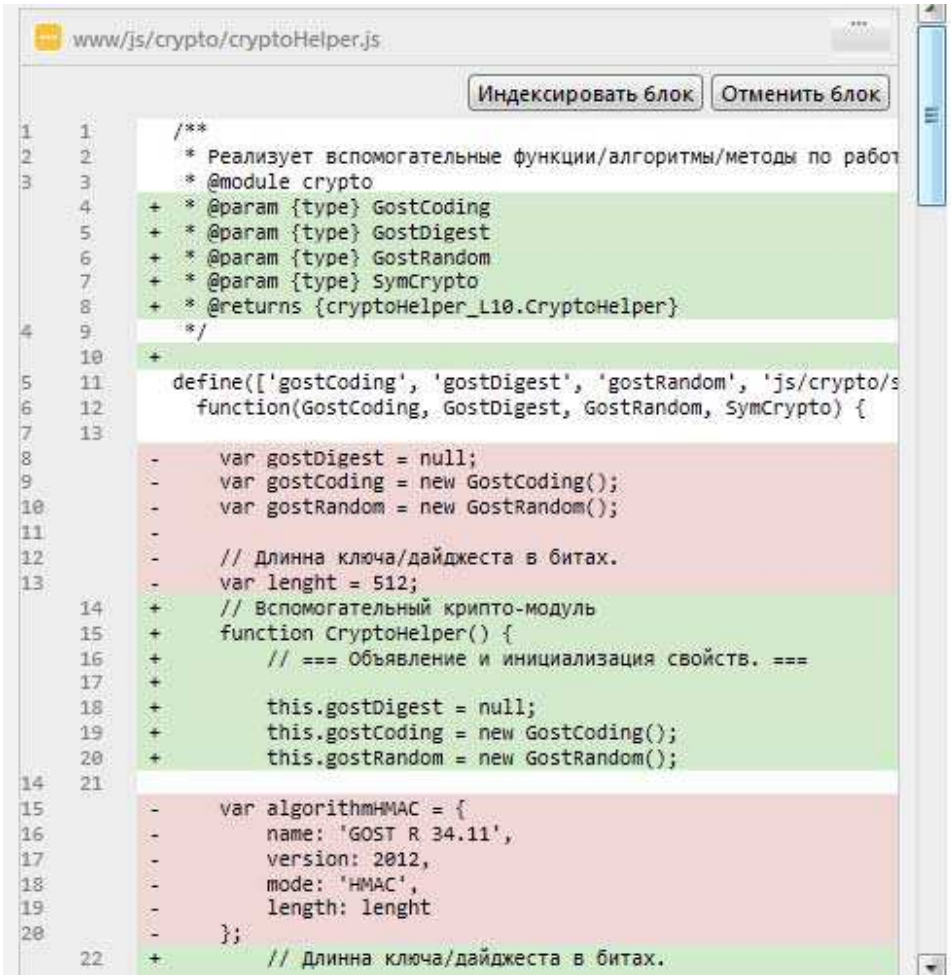

Рисунок 27 – Исправление ошибки, выявленной ранее, на примере класса *CryptoHelper*

На рисунке 28 показаны результаты запуска модульных тестов для криптографического модуля.

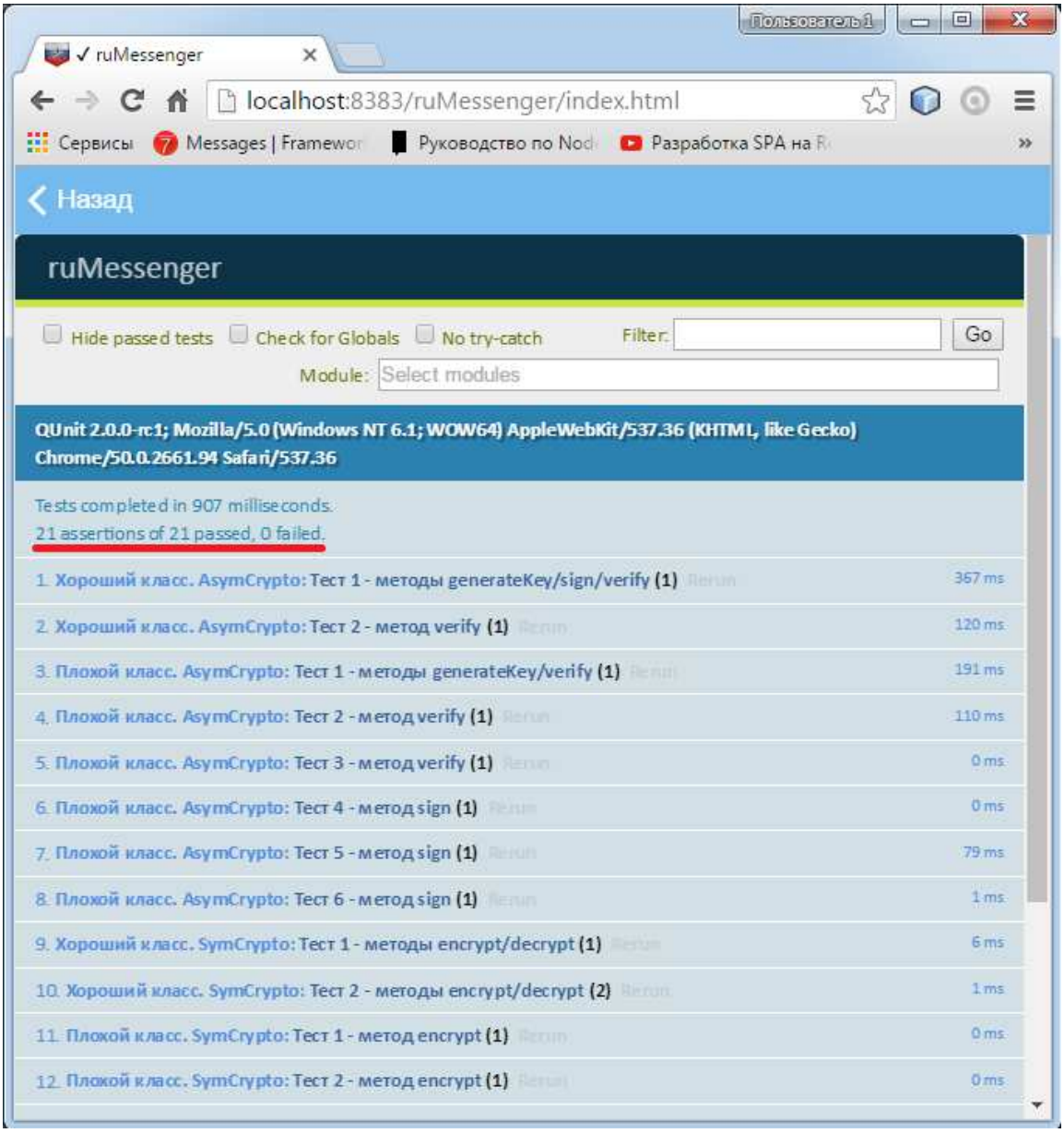

Рисунок 28 – Результаты запуска юнит-тестов, написанных для криптографического модуля

3.2.3 Функциональное тестирование

В качестве объекта тестирования был выбран сценарий приема и передачи сообщений между абонентами со стороны клиентской части программного комплекса, т.к. это основной и наиболее часто используемый функционала ИС. Данная функциональность реализуется на базе шаблона *MVC* и бизнес-логики отображения страниц *dialog*, она отвечает за работу с диалогом (чатом) между пользователями.

Целью тестирования является обнаружение ошибок следующего характера:

– ошибки в функциональности посредством интерфейса;
– необработанные исключения при взаимодействии с интерфейсом;

– потеря или искажение данных, передаваемых через элементы интерфейса;

– ошибки в интерфейсе.

3.2.3.1 Структура тестируемого модуля

Тестируемый модуль состоит из четырех файлов (компонентов), представленных на рисунке 29 с описанием в таблице 18, разделяющих зону ответственности между бизнес-логикой и пользовательским интерфейсом.

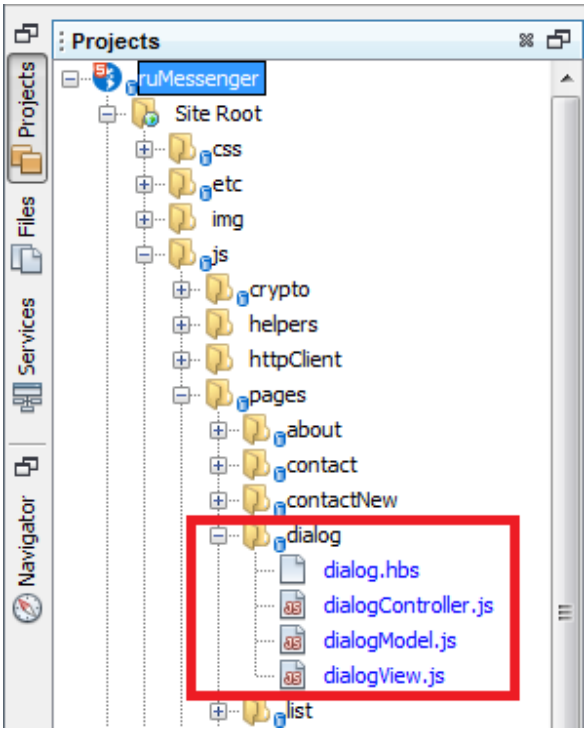

Рисунок 29 – Структура модуля *dialog*

Таблица 18 – Описание файлов модуля *dialog*

Модуль	Файл	Описание
dialog	*dialog.hbs*	Описывает представление, т.е. графические компоненты пользовательского интерфейса на языке *HTML*.
	dialogModel.js	Файл модели, содержит класс *Dialog*. Этот класс описывает сущность “Диалог”.
	dialogView.js	Инициализирует описанное представление (файл с расширением *hbs*) на *HTML* странице. Также связывает события компонентов с их обработчиками.
	dialogController.js	Промежуточное, связывающее, звено между моделью (*dialogModel.js*) и представлением (*dialogView.js*), определяет обработчики событий графических элементов управления.

3.2.3.2 Используемые компоненты

Для создания автоматизированных тестов применяется инструмент *Selenium* [49], в частности *Selenium IDE*. Инструмент *Selenium* поддерживает взаимодействие с различными браузерами, и написания сложных тестов, что достигается за счет части *Selenium Webdriver*. Однако в данной работе используется плагин *Selenium IDE* к браузеру *Mozilla Firefox*, который позволяет записывать и воспроизводить действия пользователя в браузере. Выбор данной части обусловлен низким порогом изучения инструмента, удобством в эксплуатации, а также его компактностью.

Следует отметить, что при запуске тестов рекомендуется ставить низкую (*slowly*) скорость выполнения теста, пример показан на рисунке 30, либо в поле команд выбрать *ClickAndWait*. Потому что скорость выполнения теста и загрузки страницы варьируются, и это может стать источником провала тестовых сценариев.

3.2.3.3 Сценарии тестирования

Проверка пользовательского интерфейса осуществляется согласно тест-требованиям, приведенным в таблице 19, и сценарию тестирования (тест-плану), приведенному в таблице 20.

Таблица 19 – Тест-требования для сценария приема и передачи сообщений

ИД требования	Наименование	Описание
TR1	Интуитивность	Проверить, что система (модуль) имеет понятный пользовательский интерфейс. Критерии: – наличие подсказок; – отсутствие неоднозначных слов, терминов; – малая глубина (не более 3) вложенности меню. ***Способ проверки:*** ручной и автоматизированный.
TR2	Перекрытие	Проверить, что графические компоненты системы (модуля) не перекрываются. ***Способ проверки:*** ручной и автоматизированный.
TR3	Шрифты	Проверить, отсутствие неоднозначного понимания отображаемых текстовых символов алфавита. А также искажения, текстовой информации. ***Способ проверки:*** ручной и автоматизированный.
TR4	Размер сообщение	Проверить, что отправляемое сообщение имеет длину не более 1024 символов. ***Способ проверки:*** автоматизированный.
TR5	Тип сообщения	Проверить, как отображается отправленное и принятое сообщение. ***Способ проверки:*** автоматизированный.

Таблица 20 – Тест-план для сценария приема и передачи сообщений

ИД плана	**Описание**	**Сценарий тестирования**
TP1	***Группа тестов:*** Визуальный осмотр. ***Тест-требования:*** *TR1*, *TR2*, *TR3*. ***Назначение:*** Проверка того, что все элементы отображаются корректно, и отсутствуют визуальные дефекты: (прыгающие шрифты, отсутствие выравнивания, наложения, и т.д.).	***Шаги сценария:*** 1. Открыть приложение в *web*-браузере; 2. Нажать по кнопке меню, и выбрать пункт «Диалоги»; 3. Выбрать любой диалог, например, «Яна Иванова». ***Ожидаемый результат:*** Все компоненты логично структурированы и отображаются в соответствии с требованиями. ***Критерий прохождения теста:*** Все ожидаемые значения совпадают с реальными.
TP2	***Группа тестов:*** Работа с сообщениями. ***Тест-требования:*** *TR3*. ***Назначение:*** Проверка того, что длинна отправляемого сообщения проверяется на превышение максимально возможной длинны. В случае подтверждения отправки, отправка не происходит с соответствующим уведомлением пользователя.	***Шаги сценария:*** 1. Открыть приложение в *web*-браузере; 2. Нажать по кнопке меню, и выбрать пункт «Диалоги»; 3. Ввести сообщение, длинна которого больше 1024 символа; 4. Нажать на кнопку отправить. ***Ожидаемый результат:*** Должно появиться уведомление “Сообщение превышает допустимый размер. Пожалуйста, уменьшите число символов”, введенное сообщение не удаляется. ***Критерий прохождения теста:*** Все ожидаемые значения совпадают с реальными.
TP3	***Группа тестов:*** Работа с сообщениями. ***Тест-требования:*** *TR4*. ***Назначение:*** Проверка того, что сообщение отправляется и (или) принимается.	***Шаги сценария:*** 1. Открыть приложение в *web*-браузере; 2. Нажать по кнопке меню, и выбрать пункт «Диалоги»; 3. Ввести сообщение, длинна которого не превышает 1024 символа; 4. Нажать на кнопку отправить. ***Ожидаемый результат:*** В окне диалога должно появиться введенный текст отправленного сообщения на зеленом фоне, прижатым к правому краю. Также по левому краю должно появиться принятое сообщение с аналогичным текстом на сером фоне. ***Критерий прохождения теста:*** Все ожидаемые значения совпадают с реальными.

3.2.3.4 Организация тестового сценария

В листинге Б.2, приложения Б, демонстрируется реализация сценария тестирования (тест-план *TP3*) для тест-требования *TR5* на языке команд *Selenium*.

3.2.3.5 Результаты тестирования

Результаты, выполнения автоматизированных тестов, показаны на рисунке 30, а на рисунке 31 демонстрируется процесс автоматизированного функционального тестирования для тест-плана *TP2*. Как можно увидеть из рисунков 30-31, тесты выполнены успешно.

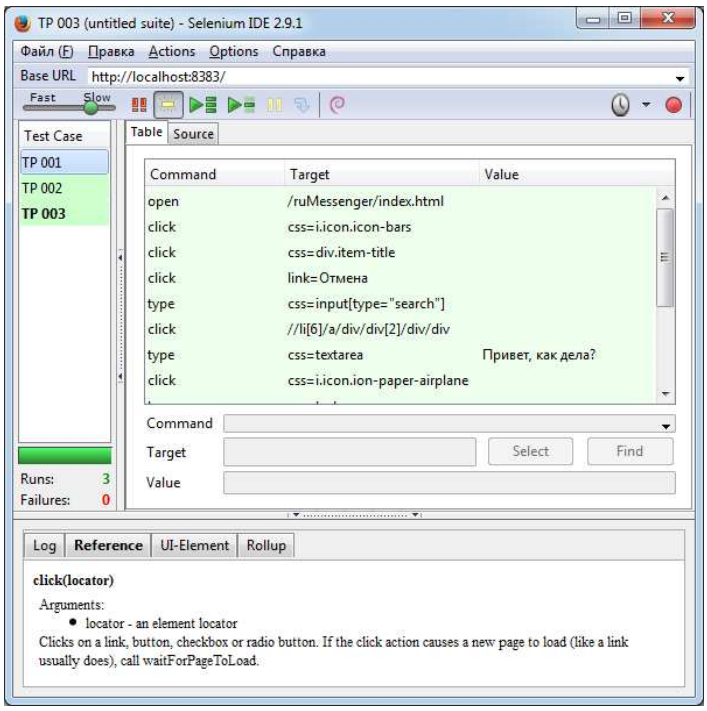

Рисунок 30 – Результаты выполненных тестов по тест-плану, описанному в таблице 20

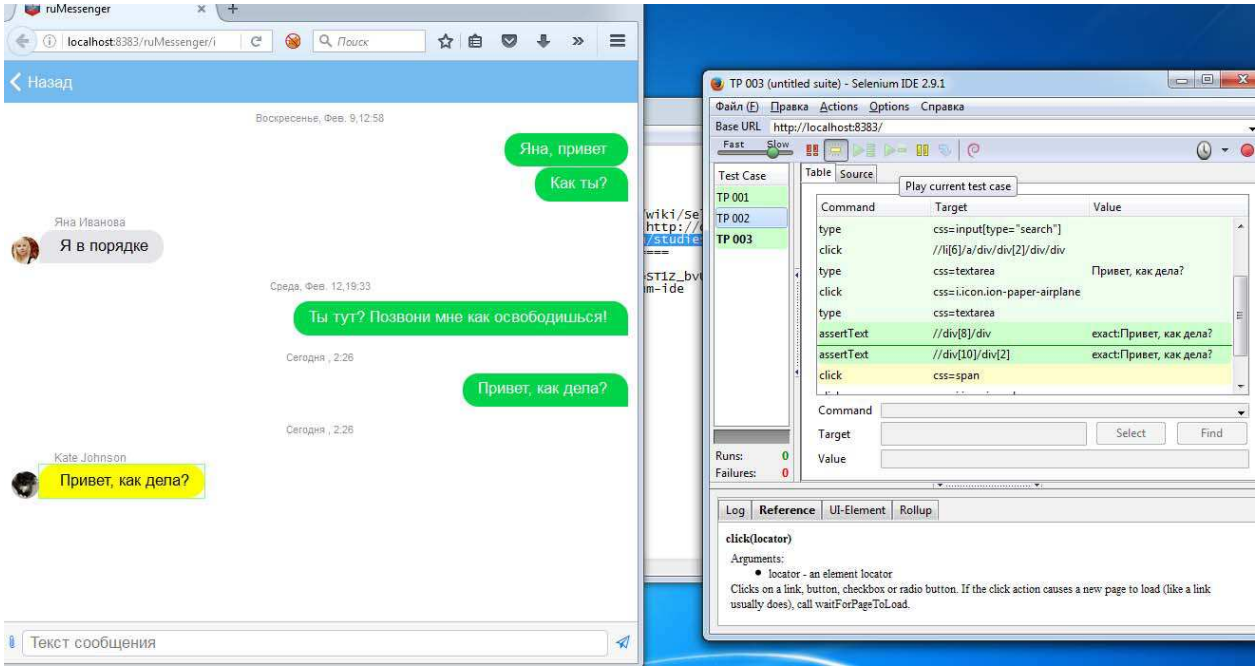

Рисунок 31 – Процесс автоматизированного функционального тестирования для тест-плана *TP2*

3.2.4 Тестирование производительности серверной части

Очевидно, что с ростом числа абонентов ИС возрастает нагрузка на её центральный узел (серверную часть), возрастает число обрабатываемых данных, увеличивается количество входящих и исходящих потоков данных и др. В связи с чем, возрастает время обслуживания запроса и увеличивается вероятность потенциальных сбоев в работе ИС. Для предотвращения данных последствий, необходимо оценить потенциальную нагрузочную способность серверной части – объект тестирования.

Целью нагрузочного тестирования являются:

– оценка поведения серверной части программного комплекса под ожидаемой нагрузкой;

– оценка времени ответа серверной части программного комплекса на запрос клиентской части с целью установления соответствия требованиям, предъявляемым к производительности серверной части.

Методика нагрузочного тестирования базировалась на варьировании количества экземпляров нагрузочного сценария с целью регистрации ключевых показателей производительности серверной части ИС [50].

3.2.4.1 Используемые компоненты

В качестве инструментария для проведения тестирования был использован программный продукт *Apache jMater* [51]. Серверная часть программного комплекса была развернута на персональном компьютере с двухъядерным

процессором *Intel Core i*3-4170, позволяющий реализовать четыре вычислительных потока, ОЗУ ёмкостью 8 Гб. В качестве операционной системы использовалась *Windows* 8.

3.2.4.2 Сценарии тестирования

Чтобы оценить потенциальную нагрузочную способность серверной части, разработанной ИС, было осуществлено её нагрузочное тестирование. Для этого, были определены следующие тестовые сценарии, описанные в таблице 21.

Таблица 21 – Сценарии нагрузочного тестирования

ИД сценария	Описание
PT1	Сценарий имитирует аутентификацию и авторизацию пользователя ИС, с получением 10 входящих сообщений. Экспериментальным путем было определено, что данный сценарий является наиболее вычислительно затратным так, как задействует основные ресурсоемкие операции: обращение к базе данных и криптографические преобразования.
PT2	Сценарий имитирует отправку сообщений в ИС авторизовавшемся пользователем. Это основной функционал ИС.
PT3	Сценарий имитирует поиск конкретного абонента в ИС по телефонному номеру, и отображения расширенных сведений о нём.
PT4	Сценарий содержит все выше описанные сценарии, которые запускаются одновременно и в равном соотношении.

3.2.4.3 Результаты тестирования

Результаты нагрузочного тестирования представлены на рисунке 32.

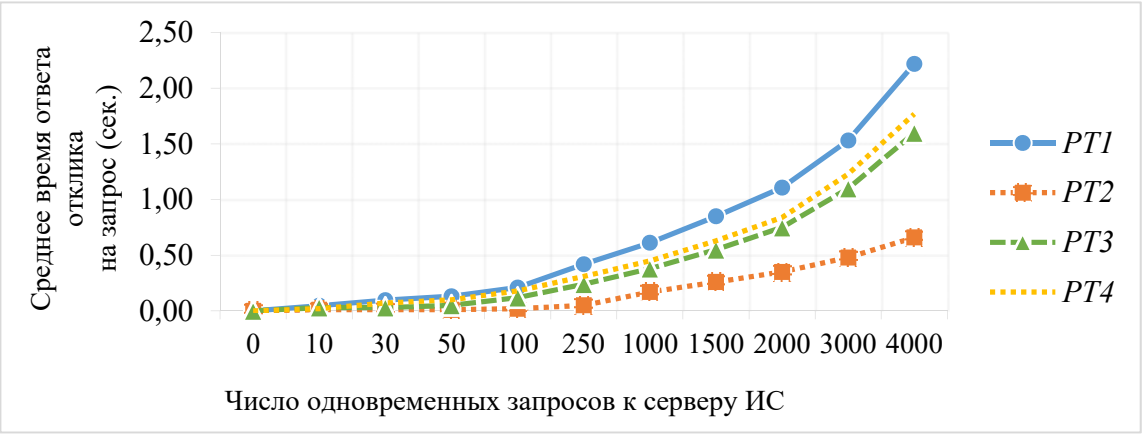

Рисунок 32 – Результаты нагрузочного тестирования

Как видно из представленных графиков, превышение отметки в ~ 2000 одновременно обрабатываемых запросов порождает время отклика системы более 1 сек., что можно принять за критическое. В реальных условиях эксплуатации ИС пороговое время отклика будет достигаться ещё меньшим значением одновременно

исполняемых запросов. В такой ситуации очевидным становится вопрос о потенциальной возможности масштабирования ИС.

3.3 Выводы по главе

Данная глава посвящена технологическим особенностям разработки информационной системы. В первом разделе главы приводится описание общей схемы взаимодействия всех программных частей и модулей в виде единого программного комплекса. Также, в данном разделе описывается используемый подход к разработке ИС, архитектура ИС, включая клиентскую и серверную стороны, архитектура БД, а также протоколы взаимодействия «абонент-сервер» и «абонент-абонента». Приводятся обоснования выбора используемых инструментов при реализации программного комплекса «*ruMessanger*» [45].

Во втором разделе главы были проведены различные виды тестирования, с целью повышения качества разработанного программного комплекса. В ходе подготовки к проведению тестирования, был составлен тест-план, определяющий средства и порядок испытаний, методы испытаний, и основные этапы тестирования. Затем, был проведен этап тестирования, который позволил обнаружить и устранить критические ошибки. Посредством программного комплекса «*ruMessanger*» были получены экспериментальные оценки производительности его серверной стороны. Отмечается, что ИС в монолитном исполнении серверной части не способна нести высокую нагрузку.

Глава 4. Предлагаемый подход масштабирования информационной системы

Как было отмечено в разделе 3.2.4 главы 3, что ИС в монолитном исполнении серверной части не способна нести высокую нагрузку, которая присуща системам мгновенного обмена информацией с огромным количеством пользователей. В силу чего, в данной главе предлагается подход к масштабированию архитектуры центрального узла, который в настоящий момент находится на стадии технологической реализации.

4.1 Проектирование архитектуры масштабирования системы

Конфиденциальная информация

4.2 Выводы по главе

В данной главе предлагается подход по масштабированию архитектуры серверной стороны программного комплекса, поскольку серверная сторона в монолитном исполнении не способна нести высокую нагрузку, которая характерна для систем мгновенного обмена информацией с огромным количеством пользователей. Также, отмечаются дальнейшие работы оптимизации архитектуры ИС в контексте задачи по её масштабированию.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была решена научно-техническая задача по организации безопасного обмена конфиденциальной информации в компьютерных сетях общего доступа, с учётом возможной компрометации центральной стороны информационной системы.

Основные результаты выполненной работы заключаются в следующем:

1. Проведенное изучение технической документации, в аспекте обеспечения защиты информации, современных и наиболее распространённых систем обмена информацией выявило потенциальную возможность компрометации её центрального узла владельцем ИС (или с согласия владельца) или злоумышленником. Такая потенциальная возможность существует из-за особенностей инфраструктуры ключей шифрования, предполагающая участие центрального узла во всех информационных обменах ключевой информации. Компрометация ключей шифрования может полностью нивелировать механизмы защиты передаваемой информации;
2. Разработан способ обмена ключевой информацией на основе разделении секрета и избыточности каналов связи, учитывающий возможную атаку «человек посередине» со стороны центральной части ИС. Предложенный способ

позволяет частично децентрализовать процесс распределения и выработки ключей шифрования;

3. Разработана модель информационной системы для безопасного обмена конфиденциальной информацией в компьютерных сетях общего доступа, а также сопутствующее алгоритмическое обеспечение;

4. Разработан программный комплекс на базе разработанной модели ИС, позволяющий осуществлять безопасный обмен конфиденциальной информацией в компьютерных сетях общего доступа между абонентами, а также заменить иностранные аналоги. Разработанный программный комплекс позволил успешно протестировать разработанную модель, что позволило экспериментально подтвердить целесообразность применения предложенного способа при условии возможной компрометации со стороны центрального узла. Получено свидетельство о государственной регистрации программы для ЭВМ – приложение Г;

5. Исследован разработанный программный комплекс на предмет надежности, были выявлены и исправлены ошибки в программных приложениях обеих частей комплекса, получены экспериментальные оценки производительности центрального узла, разработана архитектура масштабирования центрального узла.

Несмотря на схожий функционал с аналогичными системами, разработанный программный комплекс имеет отличительную особенность, заключающуюся в способе обмена ключевой информации между абонентами, который основан на избыточности каналов связи и разделении общего секрета. Избыточность в каналах связи создает практическую сложность в успешном проведении атаки «человек посередине», нежели использование одного канала связи и вычисление *QR*-кода из открытых ключей абонентов, как это реализуется в аналогичных системах.

Однако стоит подчеркнуть, что используемые авторские механизмы защиты информации не были в достаточном объёме проанализированы экспертным сообществом, в отличии от решений, применяемых в аналогичных системах.

Дальнейшие перспективы работы по данному направлению исследований заключаются в комплексной оценке безопасности, предлагаемой ИС, с учётом других потенциально возможных векторов атак, а также в рамках иной модели нарушителя. Кроме того, необходимо решить задачу технологической реализации предложенной масштабируемой архитектуры информационной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. About WhatsApp [Электронный ресурс] : офиц. сайт. – Режим доступа: https://www.whatsapp.com/about/?l=en (дата обращения: 06.06.2017);

2. Глинкин, А. Число пользователей WhatsApp превысило миллиард человек [Электронный ресурс] : Интернет-портал «Российской газеты». – 2016, 02 февраля. – Режим доступа: https://rg.ru/2016/02/02/chislo-polzovatelej-whatsapp-prevysilo-milliard-chelovek.html (дата обращения: 06.06.2017);

3. Белов, А. В. Синтез алгоритмов построения доверенных информационных систем с трехзвенной архитектурой / А. В. Белов, Д. В. Пашков // Вычислительные технологии в естественных науках. Методы суперкомпьютерного моделирования : сборник трудов, часть 2, 21-23 апреля 2015 г., Россия, г. Таруса / Российской акад. наук ; под ред. Р.Р. Назирова, Л.Н. Щура. – Москва : ИКИ РАН, 2015 – (Серия «Механика, управление и информатика»);

4. Schneier, B. Applied Cryptography: Protocols, Algorthms, and Source Code in C, 2nd edition, John Wiley & Sons, 1996.;

5. Bernstein D. J. Curve25519: new Diffie-Hellman speed records [Электронный ресурс] – Режим доступа: https://cr.yp.to/ecdh/curve25519-20060209.pdf (дата обращения: 06.06.2017);

6. WhatsApp Encryption Overview. Technical white paper [Электронный ресурс] : техн. информация // WhatsApp Inc. – Дата обновления: 17.11.2016. – Режим доступа: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (дата обращения: 06.06.2017);

7. Perrin, T. The Noise Protocol Framework [Электронный ресурс] : техн. информация, версия 32. – Дата обновления: 17.05.2017. – Режим доступа: http://noiseprotocol.org/noise.html (дата обращения: 06.06.2017);

8. McGrew, D. A. The Galois/Counter Mode of Operation (GCM) / McGrew D. A., Viega J. [Электронный ресурс]. – Дата публикации: май 2005. – Режим доступа: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf (дата обращения: 06.06.2017);

9. FIPS PUB 180-4. Secure Hash Standard (SHS). U.S Department of Commerce, National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899-8900. – Дата публикации: август 2015. – Режим доступа: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (дата обращения: 06.06.2017);

10. Cohn-Gordon, K. Formal Security Analysis of the Signal Messaging Protocol [Электронный ресурс] : аналит. отчет, версия 1.1 / K. Cohn-Gordon, C. Cremers, B.

Dowling, L. Garratt, and D. Stebila // A Cryptology ePrint Archive, 2016. – Дата обновления: октябрь 2016. – Режим доступа: https://eprint.iacr.org/2016/1013.pdf (дата обращения: 06.06.2017);

11. RFC 5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF) / H. Krawczyk, P. Eronen. // Internet Engineering Task Force (IETF). – Дата публикации: май 2010. – Режим доступа: https://tools.ietf.org/html/rfc5869 (дата обращения: 06.06.2017);

12. Фергюсон, Н. Практическая криптография : пер. с англ. / Н. Фергюсон, Б. Шнаер. – М.: Издат. дом “Вильямс”, 2004. – 432 с.: ил.;

13. Perrin, T. The Double Ratchet Algorithm [Электронный ресурс] : техн. информация, версия 1 / T.Perrin, M. Marlinspike. – Дата публикации: 20.11.2016. – Режим доступа: https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf (дата обращения: 06.06.2017);

14. Блэк, У. Интернет: протоколы безопасности. Учебный курс. / У. Блэк. – СПб.: Питер, 2001. – 288 с.: ил.;

15. Creating an Authorization Key [Электронный ресурс] : техн. информация // Telegram. – Режим доступа: https://core.telegram.org/mtproto/auth_key (дата обращения: 06.06.2017);

16. Gligor, V. D. On Message Integrity in Symmetric Encryption / V.D. Gligor, P. Donescu // In: Proc. 1st NIST Workshop on AES Modes of Operation. – Дата публикации: 10.11.2000. – Режим доступа: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ige/ige-spec.pdf (дата обращения: 01.04.2017);

17. Евдокимов, Д. Безопасность мобильного банкинга: возможность реализации атаки “MITM” [Электронный ресурс] : исследование // Digital Security. – 2014. – Режим доступа: http://dsec.ru/upload/medialibrary/56e/56e70f90cbcc8c092f036d8005351fd9.pdf (дата обращения: 06.06.2017);

18. MTProto Mobile Protocol [Электронный ресурс] : техн. информация // Telegram. – Режим доступа: https://core.telegram.org/mtproto (дата обращения: 06.06.2017);

19. Secret chats, end-to-end encryption [Электронный ресурс] : техн. информация // Telegram. – Режим доступа: https://core.telegram.org/api/end-to-end (дата обращения: 06.06.2017);

20. Key Visualization [Электронный ресурс] : техн. информация // Telegram. – Режим доступа: https://core.telegram.org/api/end-to-end/pfs#key-visualization (дата обращения: 06.06.2017);

21. About Viber [Электронный ресурс] : офиц. сайт. // Viber Media S.à r.l. – Режим доступа: https://www.viber.com/en/about (дата обращения: 06.06.2017);

22. Viber оценил число своих пользователей в России в 76 млн. [Электронный ресурс] : сайт // Сообщения и материалы информационного агентства «РБК». – Дата публикации: 21.12.2016. – Режим доступа: http://www.rbc.ru/rbcfreenews/585aa5c79a7947b25b2e2a94 (дата обращения: 21.12.2016);

23. Viber Encryption Overview [Электронный ресурс] : техн. информация // Viber Media S.à r.l. – Режим доступа: https://www.viber.com/en/security-overview (дата обращения: 06.06.2017);

24. Bernstein, D. J. Salsa20 specification [Электронный ресурс] : техн. информация / D. J. Bernstein. – Режим доступа: https://cr.yp.to/snuffle/spec.pdf (дата обращения: 06.06.2017);

25. Threema. The best-selling secure messenger [Электронный ресурс] : материалы для прессы // Threema GmbH. – Режим доступа: https://threema.ch/press-files/1_press_info/Press-Info_Threema_EN.pdf (дата обращения: 06.06.2017);

26. Threema Cryptography Whitepaper [Электронный ресурс] : техн. информация // Threema GmbH. – Дата обновления: 02.06.2017. – Режим доступа: https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf (дата обращения: 06.06.2017);

27. NaCl: Networking and Cryptography library [Электронный ресурс] : офиц. сайт. – Режим доступа: http://nacl.cr.yp.to (дата обращения: 06.06.2017);

28. Bernstein, D. J. Cryptography in NaCl [Электронный ресурс] : техн. информация / D. J. Bernstein. – Режим доступа: https://cr.yp.to/highspeed/naclcrypto-20090310.pdf (дата обращения: 06.06.2017);

29. Bernstein, D. J. Extending the Salsa20 nonce [Электронный ресурс] : техн. информация / D. J. Bernstein. – Режим доступа: https://cr.yp.to/snuffle/xsalsa-20081128.pdf (дата обращения: 06.06.2017);

30. Bernstein, D. J. The Poly1305-AES message-authentication code [Электронный ресурс] : техн. информация / D. J. Bernstein. – Режим доступа: http://cr.yp.to/mac/poly1305-20050329.pdf (дата обращения: 06.06.2017);

31. Wind, D. Man-in-the-middle attack on TextSecure [Электронный ресурс] : видео доклад с конференции IT-Security Community Xchange (ITSeCX), 2015. // University of Applied Science St. Pölten. – Режим доступа: https://www.youtube.com/watch?v=bSap-VI4oh8 (дата обращения: 06.06.2017);

32. Jakobsen, J. B. A practical cryptanalysis of the Telegram messaging protocol : master's thesis : computer science / Jakob Bjerre Jakobsen. – Aarhus, 2015. – 79 p. – Режим доступа: https://cs.au.dk/~jakjak/master-thesis.pdf (дата обращения: 06.06.2017);

33. WebCrypto GOST Library [Электронный ресурс] : офиц. сайт. – Режим доступа: http://gostcrypto.com (дата обращения: 06.06.2017);

34. Реализация WebCrypto генератора для алгоритмов ГОСТ [Электронный ресурс] : исход. код. – Режим доступа: https://github.com/rudonick/crypto/blob/master/gostRandom.js (дата обращения: 06.06.2017);

35. Watson, M. Web Cryptography API. W3C Recommendation [Электронный ресурс] : техн. информация / M. Watson. – Дата обновления: 26.01.2017. – Режим доступа: https://www.w3.org/TR/WebCryptoAPI/ (дата обращения: 06.06.2017);

36. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хэширования. – Взамен ГОСТ Р 34.11-94 ; введ. 01.01.2013. – Москва ; Стандартинформ, 2012. – 38 с.;

37. Turan, M.S. NIST SP 800-132 Recommendation for Password-Based Key Derivation. Part 1: Storage Applications [Электронный ресурс] / M. S. Turan, E. Barker, W. Burr, L. Chen // U.S Department of Commerce, National Institute of Standards and Technology (NIST). – Дата публикации: декабрь 2010. – Режим доступа: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf (дата обращения: 06.06.2017);

38. RFC 2898 PKCS #5: Password-Based Cryptography Specification. Version 2 / B. Kaliski // Internet Engineering Task Force (IETF). – Дата публикации: сентябрь 2000. – Режим доступа: https://tools.ietf.org/html/rfc2898 (дата обращения: 06.06.2017);

39. ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры. – Введ. 01.01.2016. – Москва ; Стандартинформ, 2015. – 25 с.;

40. ГОСТ Р 34.13-2015. Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров. – Введ. 01.01.2016. – Москва ; Стандартинформ, 2015. – 42 с.;

41. RFC 6455 The WebSocket Protocol / I. Fette, A.Melnikov // Internet Engineering Task Force (IETF). – Дата публикации: декабрь 2011. – Режим доступа: https://tools.ietf.org/html/rfc6455 (дата обращения: 06.06.2017);

42. Макконнелл, С. Совершенный код / С. Макконнелл. – 2-е изд., пер. с англ. – Москва : Издательство «Русская редакция», 2016. – 896 стр.: ил.;

43. Мацяшек, Лешек А. Анализ и проектирование информационных систем с помощью UML 2.0 / Лешек А. Мацяшек. – 2-е изд., пер. с англ. – Москва : ООО “И. Д. Вильямс”, 2008. – 816 с. : ил. – Парал. тит. англ.;

44. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб.: Питер, 2016. – 368 с.: ил. – (Серия «Библиотека программиста»);

45. Свидетельство № 2017610542 Российская Федерация. Программный комплекс для конфиденциального обмена информацией в компьютерных сетях общего доступа «ruMessanger» : свидетельство о государственной регистрации программы для ЭВМ / А. Н. Шниперов, А. П. Чистяков ; заявитель и правообладатель ФГАОУ ВО «Сибирский федеральный университет» ; заявл. 17.11.2016 ; зарегистр. 12.01.2017. – 1 с.;

46. MongoDB in Action: Covers MongoDB version 3.0 / K. Banker, P. Bakkum, S. Verch, D. Garrett, T. Hawkins. – 2nd edition. – Printed in black & white, March 2016. – 480 pages;

47. QUnit. A JavaScript Unit Testing framework. [Электронный ресурс] : офиц. сайт. – Режим доступа: https://qunitjs.com (дата обращения: 06.06.2017);

48. Myers, G. J. The art of software testing / G. J. Myers ; Revised and Updated by T. Badgett, T. M. Thomas, C. Sandler. – 2nd edition. – New York : Wiley, 2004. – 255 pages.;

49. Selenium [Электронный ресурс] : офиц. сайт. – Режим доступа: http://www.seleniumhq.org/ (дата обращения: 06.06.2017);

50. Хэа, Д. Тестирование и анализ производительности с помощью сервера приложений WebSphere [Электронный ресурс] / Д. Хэа. – Дата публикации: 13.06.2013. – Режим доступа: https://www.ibm.com/developerworks/ru/library/wes-1208_hare/ (дата обращения: 06.06.2017);

51. Apache JMeter [Электронный ресурс] : офиц. сайт. – Режим доступа: http://jmeter.apache.org (дата обращения: 06.06.2017);

52. Браун, И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / И. Браун. – СПб. : Питер, 2017. – 336 с.: ил. – (Серия «Бестселлеры O'Reilly»);

53. Nginx [Электронный ресурс] : офиц. сайт. – Режим доступа: https://www.nginx.com (дата обращения: 06.06.2017);

54. Using nginx as HTTP load balancer. Weighted load balancing [Электронный ресурс] : техн. информация. – Режим доступа: http://nginx.org/en/docs/http/load_balancing.html#nginx_weighted_load_balancing (дата обращения: 06.06.2017);

55. Redis [Электронный ресурс] : офиц. сайт. – Режим доступа: https://redis.io (дата обращения: 06.06.2017);

56. OCFS2 - Oracle Cluster File System for Linux [Электронный ресурс] : техн. информация // Oracle Corporation. – Режим доступа: http://www.oracle.com/us/technologies/linux/025995.htm (дата обращения: 06.06.2017).

ПРИЛОЖЕНИЕ А - Техническое задание

Данный документ составлен исходя из стандарта ГОСТ 19.201-78. Эта часть включает в себя технические требования на создание программного комплекса для конфиденциального обмена информацией в открытых компьютерных сетях. Программная система предназначена для безопасной коммуникации между пользователями, и создается для апробации модели основанной на предлагаемом способе, а также как отечественная альтернатива зарубежным системам.

А1 Основание для разработки

Данная работа выполняется в рамках задания к выпускной квалификационной работе «Модель и программный комплекс системы безопасного обмена информацией в компьютерных сетях общего доступа» для получения степени «магистр» по направлению 09.04.04 «Программная инженерия» в ФГАОУ ВО «Сибирский федеральный университет».

А2 Назначение разработки

А2.1 Функциональное назначение

Функциональным назначением программного комплекса является мгновенный информационный обмен между абонентами в информационно-телекоммуникационной сети Интернет.

А2.2 Эксплуатационное назначение

Эксплуатационным назначением комплекса является предоставление инструмента с повышенным уровнем безопасности для социального взаимодействия между людьми в сети Интернет. Взаимодействие между людьми может осуществляться в личных, коммерческих, и иных интересах.

А2.3 Состав функций

Функциональные возможности комплекса представлены на диаграмме вариантов использования (*use case diagram*) изображенной на рисунке А.1 с использованием языка *UML* (*Unified Modeling Language*), а описание основных вариантов использования приведено в таблице А.1.

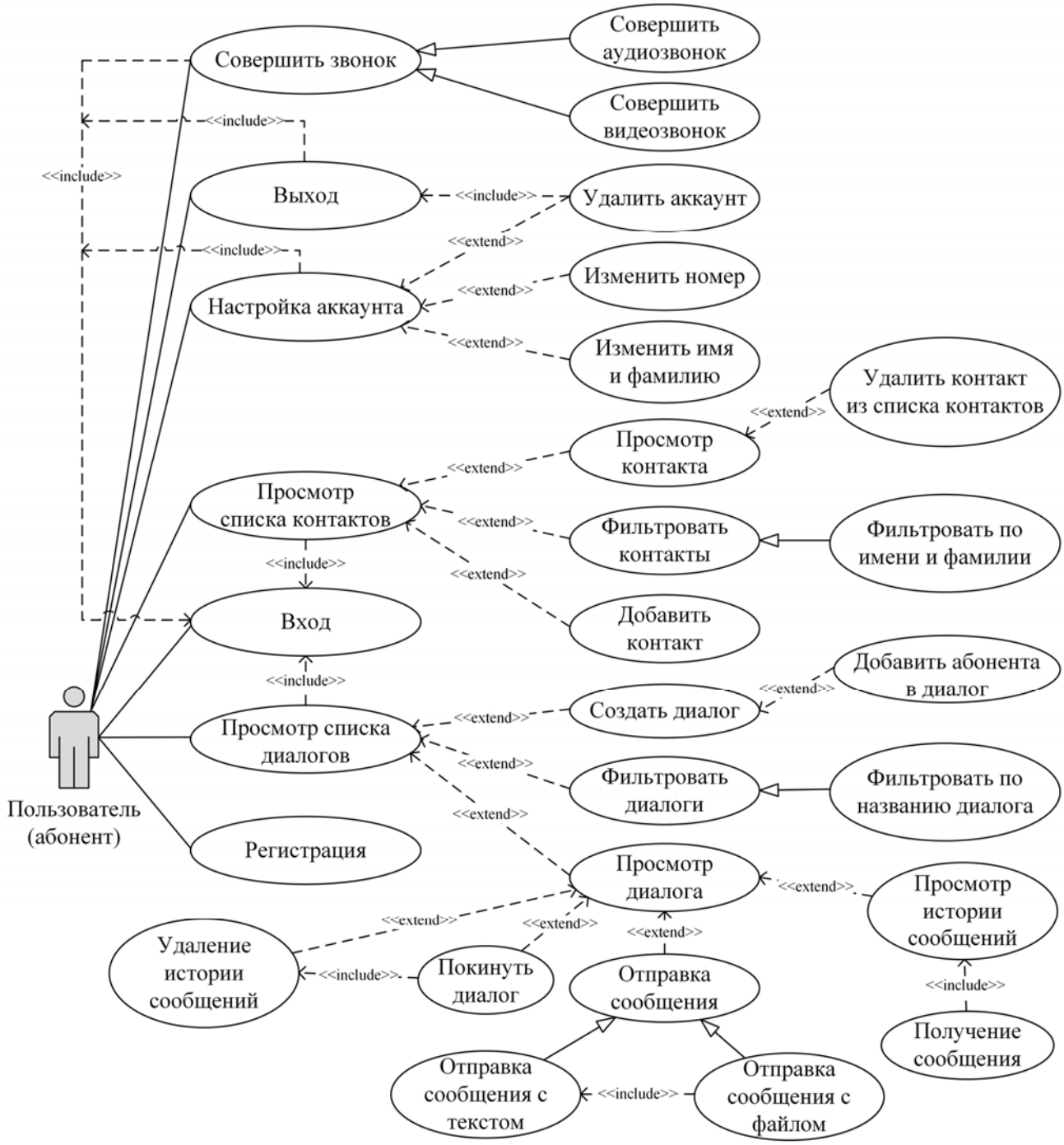

Рисунок А.1 – Диаграмма вариантов использования программного комплекса

Таблица А.1 – Детальный обзор основных сценариев использования

Сценарий использования	**Описание**
Регистрация	***Основной исполнитель:*** пользователь. ***Предусловия:*** пользователь успешно зашел в приложение (клиентская часть), и успешно установил безопасное соединение с сервером.

Продолжение таблицы А.1

Сценарий использования	Описание
Регистрация (продолжение)	***Результат:*** аккаунт успешно создан и сохранен, пользователь уведомлен об результате. ***Основной успешный сценарий:*** пользователь вводит данные, и передает их на проверку серверу, который создает аккаунт.
Вход	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе, успешно зашел в приложение, и установил безопасное соединение с сервером. ***Результат:*** пользователь аутентифицирован, авторизован, успешно переведен в статус «Онлайн». ***Основной успешный сценарий:*** пользователь вводит аутентификационные данные, передает их серверу на проверку, и получает от сервера токен (аутентификационный билет).
Настройка аккаунта	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе, успешно прошел аутентификацию и авторизацию. ***Результат:*** настройки успешно применены. ***Основной успешный сценарий:*** настройки успешно изменены и сохранены.
Изменить номер	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе, успешно прошел аутентификацию и авторизацию, а также подтвердил проводимую операцию вводом одноразового кода, полученный в смс-сообщении (высылается по новому номеру). ***Результат:*** номер телефона успешно изменен. ***Основной успешный сценарий:*** пользователь указывает новый номер телефона, на который высылается смс-сообщение с одноразовым кодом подтверждения выполняемой операции. После чего, пользователь водит полученный код из смс-сообщения. Затем передает его серверу на проверку, который привязывает новый номер телефона к аккаунту, и уведомляет пользователя.
Удаление аккаунта	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе, успешно прошел аутентификацию и авторизацию, и подтвердил проводимую операцию вводом одноразового кода, полученного в смс-сообщении. ***Результат:*** аккаунт, и все связанные с ним данные успешно удалены. ***Основной успешный сценарий:*** пользователь посылает запрос серверу на удаление аккаунта, и подтверждает свое действие путем ввода смс-кода, который отправил сервер.

Продолжение таблицы А.1

Сценарий использования	Описание
Удалить аккаунт (продолжение)	После успешной проверки, присланного пользователем смс-кода, сервер удаляет аккаунт, и все связанные с ним данные, а также уведомляет пользователя. Затем клиентская часть перенаправляет пользователя на страницу входа в аккаунт.
Просмотр списка диалогов	***Основной исполнитель:*** пользователь. ***Предусловия:*** Исполнитель создан в системе и успешно прошел аутентификацию и авторизацию. ***Результат:*** исполнителю отображен список его диалогов с другими абонентами системы. ***Основной успешный сценарий:*** список диалогов успешно получен от сервера, и отображен пользователю.
Создать диалог	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе и успешно прошел аутентификацию и авторизацию. ***Результат:*** создан диалог. ***Основной успешный сценарий:*** пользователь создал диалог.
Добавить абонента в диалог	***Основной исполнитель:*** пользователь. ***Предусловия:*** диалог создан в системе, исполнитель и добавляемый абонент созданы в системе и успешно прошли аутентификацию и авторизацию. ***Результат:*** выбранный абонент добавлен в диалог, и уведомлен об этом. ***Основной успешный сценарий:*** пользователь добавил абонента в диалог.
Отправить сообщение	***Основной исполнитель:*** пользователь. ***Предусловия:*** получатель создан в системе. Исполнитель создан в системе и успешно прошел аутентификацию и авторизацию, а также имеет открытый ключ получателя. ***Результат:*** получатель получил сообщение. ***Основной успешный сценарий:*** сообщение успешно передано и сохранено в истории переписки.
Совершить звонок	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе и успешно прошел аутентификацию и авторизацию. ***Результат:*** создана аудио/видео связь. ***Основной успешный сценарий:*** вызываемый пользователь получает уведомление об его вызове, и принимает решение.
Выход	***Основной исполнитель:*** пользователь. ***Предусловия:*** исполнитель создан в системе (имеет статус «Онлайн») и успешно прошел аутентификацию и авторизацию. ***Результат:*** исполнитель переведен в статус «Оффлайн», и приложение завершает свою работу. ***Основной успешный сценарий:*** резервная копия ключей шифрования успешно создана и передана на сервер, после

Окончание таблицы А.1

Сценарий использования	Описание
Выход (продолжение)	чего производиться завершение работы клиентского приложения.

А3 Требования к программному комплексу

А3.1 Требования к архитектуре программного комплекса

Система должна удовлетворять требованиям по архитектуре, представленные в таблице А.2.

Таблица А.2 – Требования архитектуры

Идентификатор	Описание требований
AR1	Система должна быть построена на архитектуре «клиент-сервер».
AR2	Система должна позволять дальнейшую модернизацию и масштабирование в части: – увеличения количества пользователей, работающих с системой; – увеличение объёма хранимой информации; – увеличения количества проводимых операций и событий; – подключения новых модулей; – увеличения количества интегрируемых систем и программно-аппаратных комплексов.

А3.2 Функциональные требования

Клиентская часть ИС должна представлять пользователю следующую функциональность, представленную в таблице А.3.

Таблица А.3 – Функциональные требования к клиентской части

Идентификатор	Описание требований
FR1_C	Возможность редактировать (добавить/удалить) список контактов.
FR2_C	Возможность редактировать личные данные.
FR3_C	Возможность поиска контакта из списка контактов и по базе данных ИС.
FR4_C	Возможность создавать групповые чаты (диалоги).
FR5_C	Возможность отправлять текстовые сообщения.
FR6_C	Возможность отправлять файлы.
FR7_C	Уведомлять о событиях, например, о входящем сообщении, смене ключей шифрования, и т.д.
FR8_C (необязательно)	Возможность совершать голосовой (аудио) звонок.
FR9_C (необязательно)	Возможность совершать видеозвонок.

Серверная часть ИС должна выполнять следующую функциональность, представленную в таблице А.4.

Таблица А.4 – Функциональные требования к серверной части

Идентификатор	Описание требований
FR1_S	Обрабатывать запросы, получаемые с клиентской части.
FR2_S	Хранить пользовательские данные.

А3.3 Требования к пользовательскому интерфейсу

Требования к интерфейсу серверной части отсутствуют, поскольку она представляет собой консольное приложение. Для клиентской части требования представлены в таблице А.5.

Таблица А.5 – Требования к пользовательскому интерфейсу клиентской части

Идентификатор	Описание требований
UI1_C	Интерфейс клиентской части должен быть простым, удобным, интуитивно понятным.
UI2_C	Дизайн страниц должен быть лаконичным и единообразным.
UI3_C	Поддержка различных разрешений экрана, т.е. автоматическое масштабирование страниц в зависимости от ширины экрана.
UI4_C	Расположение элементов управления должно быть логически сгруппированным.
UI5_C	Цветовая гамма: – основные цвета: оттенки серого, белый, синий; – фоновый цвет основного меню: серо-синий; – фоновый цвет навигационного бара: светло-синий; – цвет текстовых надписей, иконок, расположенных на синем фоне: белый; – фоновый цвет контентной части страниц и списков: белый; – цвет текстовых надписей на белом фоне: черный – для основной информации, и серый – дополнительный.

А3.4 Требования безопасности

Клиентская и серверная части должны отвечать требованиям безопасности, приведенным в таблице А.6.

Таблица А.6 – Общие требования безопасности

Идентификатор	Описание требований
SR1_CS	Все применяемые криптографические алгоритмы должны быть криптографически стойкими.
SR2_CS	Использовать криптографические стандарты РФ.
SR3_CS	Хранимая открытая информация не должна приводить к компрометации пользователя.
SR4_CS	Закрытые ключи могут храниться в открытом виде, только в оперативной памяти. В остальных случаях, закрытые ключи хранятся в защищенном (зашифрованном) виде.

Окончание таблицы А.6

Идентификатор	Описание требований
SR5_CS	Обеспечить конфиденциальность и целостность передаваемой информации по открытым каналам связи.
SR6_CS	Обеспечить целостность данных при их обработке.

Серверная часть должны отвечать требованиям безопасности, представленным в таблице А.7.

Таблица А.7 – Требования безопасности серверной части

Идентификатор	Описание требований
SR1_S	Серверная часть ИС должна обеспечивать регистрацию событий безопасности: – попытки неудачной аутентификации и авторизации пользователей; – попытки нарушения прав доступа.
SR2_S	Серверная часть должна обеспечить аутентификацию и авторизацию пользователей при выполнении всех запросов от пользователя.
SR3_S	Аутентификация должна быть многофакторной.
SR4_S	Серверная часть должна удалять неподдерживаемые сеансы связи с клиентской частью, превышающие по времени 1 часа.

В таблице А.8 приведены требования безопасности, касающиеся клиентской части системы.

Таблица А.8 – Требования безопасности клиентской части

Идентификатор	Описание требований
SR1_C	Удаление/изменение аккаунта пользователя производиться только после его аутентификации.
SR2_C	Никто не должен иметь возможности читать сообщения, данные, либо историю переписки пользователя, за исключением самого пользователя.
SR3_C	Возможность резервного восстановления данных: ключей шифрования, истории переписки.
SR4_C	Возможность входа в аккаунт из-под различных устройств.

А3.5 Требования к протоколу взаимодействия клиентского и серверного приложений

Требования к протоколу системы представлены в таблице А.9.

Таблица А.9 – Требования к протоколу взаимодействия

Идентификатор	Описание требований
PR1_AS	Возможность авторизованного входа пользователя в ИС.

Окончание таблицы А.9

Идентификатор	Описание требований
PR2_AS	Возможность регистрации пользователя в ИС.
PR3_AS	Возможность обмена сообщениями и файлами между пользователями.
PR4_AS	Возможность получать/удалять историю переписки.
PR5_AS	Возможность поиска пользователя.
PR6_AS	Возможность получения по запросу список контактов.
PR7_AS	Возможность получения/удаления/добавления резервных копий ключей шифрования.
PR8_AS	Возможность обмена ключевой информацией между пользователями.
PR9_AS	Возможность получения информации об пользователе.

А3.6 Требования к протоколу взаимодействия между клиентскими приложениями

Требования к протоколу ИС представлены в таблице А.10.

Таблица А.10 – Требования к протоколу взаимодействия

Идентификатор	Описание требований
PR1_AA	Возможность обмена ключевой информацией между пользователями.

А3.7 Другие требования

В таблице А.11 приведены прочие требования, касающиеся системы.

Таблица А.11 – Другие требования

Идентификатор	Описание требований
OR1	Используемые инструменты, фреймворки должны распространяться под свободной (*Open Source*) лицензией, а также быть бесплатными.
OR2	Использование кроссплатформенных технологий.
OR3	Система должна функционировать 24 часа в сутки, 7 дней в неделю при условии бесперебойной работы серверного и сетевого оборудования, общесистемного программного обеспечения.

А3.8 Требования к производительности системы

В таблице А.12 приведены требования к производительности системы.

Таблица А.12 – Требования к производительности системы

Идентификатор	Описание требований
PSR1	Среднее время отклика не должно превышать 1 секунду.

Окончание таблицы А.12

Идентификатор	Описание требований
PSR2	Предельное время отклика системы при выполнении любых функций не должно превышать 10 секунд.
PSR3	Предельное время доставки данных при взаимодействии с внешними информационными системами и программно-аппаратными комплексами в штатном режиме не должно превышать 5 секунд.
PSR4	Система должна быть рассчитана на одновременную работу не менее 1000 активных конкурентных (одновременно работающих) пользователей. При этом должны выполняться вышеуказанные требования по производительности.

А3.9 Требования к программной документации

В Требования к документированию работы приведены в таблице А.13.

Таблица А.13 – Требования к документированию

Идентификатор	Описание требований
DR1	Вся документация должна быть выполнена на русском языке.
DR2	Документы должны быть представлены на бумажном носителе (оригинал) и на электронном носителе (копия).
DR3	Содержание и оформление документов, разрабатываемых в рамках работы по созданию системы, должны отвечать требованиям стандарта организации СТО 4.2-07-2014 «Общие требования к построению и оформлению документов учебной деятельности» СФУ.
DR4	Для составления руководства по исходным кодам использовать автоматизированные системы сборки, например, *Doxygen*, *Javadoc*, *JSDoc*, и др.
DR5	Отчетные и иные документы, с которыми должен работать пользователь, должны быть ориентированы на *pdf*, либо *html* формат.

ПРИЛОЖЕНИЕ Б – Листинги

Листинг Б.1 – Содержимое файла *testSymCrypto.js*

```
/**
 * Модульные тесты для symCrypto, реализующий функции по работе с симметричной криптографией.
 * @module crypto
 */

// === Групповые тесты ===
 define(['js/crypto/symCrypto'], function(SymCrypto) {
     var symCrypto = new SymCrypto();

     QUnit.module("Хороший класс. SymCrypto", function() {

         QUnit.test("Тест 1 - методы encrypt/decrypt", function(assert) {
             var message = "Я к вам пишу — чего же боле?\n" +
                 "Что я могу еще сказать?\n" +
                 "Теперь, я знаю, в вашей воле\n" +
                 "Меня презреньем наказать.\n" +
                 "Но вы, к моей несчастной доле\n" +
                 "Хоть каплю жалости храня,\n" +
                 "Вы не оставите меня.\n" +
                 "Сначала я молчать хотела;...";

             var key =
"fa2471e706e695a7b765eff68c3f1a0db4b7335c7c6dc13757bcd272aa96b943759b909427184e9049043b69f24761c97253
5fd07eaa642f4cf7aa355c125c6e";
             var ciphertext = symCrypto.encrypt(key, message);
             var plaintext = symCrypto.decrypt(key, ciphertext);

             assert.ok(message === plaintext, "Passed!");
         });

         QUnit.test("Тест 2 - методы encrypt/decrypt", function(assert) {
             var message =
"1122334455667700ffeeddccbbaa99880011223344556778899aabbcceeff0a112233445566778899aabbcceeff0a002233
445566778899aabbcceeff0a0011";
             var output =
"oZMLL9fRaBA1xJKkBecULYssAPEkvf+gZU7q7GhE4Y1HsFXZW/T8CmWRaQXI4bckOjXGrGEFbJtp8YvIb/wBIMkvnyEiSMS56W3N
zHxE3JtyOA/qwIEJAo4us3BxPo403z53B4cnnvjHG/Zfi/pR8jYX9k10kJCAOASyCqOljEg=";

             var key = "8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef";
             var ciphertext = symCrypto.encrypt(key, message);
             var plaintext = symCrypto.decrypt(key, ciphertext);

             var ciphertext = ciphertext.replace(/\s/g, '');

             assert.ok(output === ciphertext, "Passed!");
             assert.ok(message === plaintext, "Passed!");
         });
     });

     QUnit.module("Плохой класс. SymCrypto", function() {

         QUnit.test("Тест 1 - метод encrypt", function(assert) {
             var message = "Я к вам пишу — чего же боле? Что я могу еще сказать?...";
             var key = "";

             var ciphertext = symCrypto.encrypt(key, message);
             assert.ok(ciphertext === null, "Passed!");
         });
         QUnit.test("Тест 2 - метод encrypt", function(assert) {
             var message = null;
             var key = null;

             var ciphertext = symCrypto.encrypt(key, message);
             assert.ok(ciphertext === null, "Passed!");
         });
     });
 });
```

Листинг Б.2 – Тест-план *TP3* на языке команд *Selenium*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://localhost:8383/" />
<title>New Test</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">New Test</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/ruMessenger/index.html</td>
        <td></td>
</tr>
<tr>
        <td>click</td>
        <td>css=i.icon.icon-bars</td>
        <td></td>
</tr>
<tr>
        <td>click</td>
        <td>css=div.item-title</td>
        <td></td>
</tr>
<tr>
        <td>click</td>
        <td>link=Отмена</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>css=input[type=&amp;quot;search&amp;quot;]</td>
        <td></td>
</tr>
<tr>
        <td>click</td>
        <td>//li[6]/a/div/div[2]/div/div</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>css=textarea</td>
        <td>Привет, как дела?</td>
</tr>
<tr>
        <td>click</td>
        <td>css=i.icon.ion-paper-airplane</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>css=textarea</td>
        <td></td>
</tr>
<tr>
        <td>assertText</td>
        <td>//div[8]/div</td>
        <td>exact:Привет, как дела?</td>
</tr>
<tr>
        <td>assertText</td>
        <td>//div[10]/div[2]</td>
        <td>exact:Привет, как дела?</td>
</tr>
<tr>
```

```
        <td>click</td>
        <td>css=span</td>
        <td></td>
</tr>
<tr>
        <td>click</td>
        <td>css=i.icon.icon-bars</td>
        <td></td>
</tr>

</tbody></table>
</body>
</html>
```

ПРИЛОЖЕНИЕ В – Апробация результатов

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Диплом

II степени

НАГРАЖДАЕТСЯ

Чистяков Алексей Павлович

за лучший доклад на секции 4.1 «Методы и системы защиты информации. Информационная безопасность»
Международной научно-технической конференции студентов, аспирантов и молодых ученых
«Научная сессия ТУСУР - 2016»

Ректор
д.т.н., профессор А.А. Шелупанов

Томск – 2016

СЕРТИФИКАТ

УЧАСТНИКА
VII МОЛОДЕЖНОЙ ШКОЛЫ-СЕМИНАРА
ПЕРСПЕКТИВА-2016

ВРУЧАЕТСЯ

ЧИСТЯКОВУ
АЛЕКСЕЮ ПАВЛОВИЧУ

Директор Института
компьютерных технологий и
информационной безопасности — Г.Е. Веселов

Институт компьютерных технологий и информационной безопасности

Таганрог, 2016 г.


ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
Диплом
III степени
награждается
А.П. Чистяков,
магистрант каф. информатики, СФУ
за лучший доклад на секции 3 «Информационные технологии и системы»
подсекция 3.2 «Распределенные информационные технологии и системы»
Международной научно-технической конференции студентов,
аспирантов и молодых ученых
«Научная сессия ТУСУР - 2017»
Председатель конференции,
Ректор
А.А. Шелупанов
Томск – 2017

ПРИЛОЖЕНИЕ Г – Свидетельство о государственной регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017610542

Программный комплекс для конфиденциального обмена информацией в компьютерных сетях общего доступа «ruMessanger»

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (RU)***

Авторы: ***Чистяков Алексей Павлович (RU),***
Шниперов Алексей Николаевич (RU)

Заявка № **2016662621**
Дата поступления **17 ноября 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***12 января 2017 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Информатика
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С.Кузнецов
подпись инициалы, фамилия
«13» 06 2017 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Модель и программный комплекс системы безопасного обмена
информацией в компьютерных сетях общего доступа
тема

09.04.04 «Программная инженерия»
код и наименования направления

09.04.04.01 «Программное обеспечение вычислительной техники
и автоматизированных систем»
код и наименования магистерской программы

Научный руководитель _____ рук. НУЛ ИБ, к.т.н А.Н.Шниперов
подпись, дата должность, ученная степень инициалы, фамилия

Выпускник _____ А.П.Чистяков
подпись, дата инициалы, фамилия

Рецензент 9.06.17 _____ рук. ИнТК СФУ, к.т.н И.А.Карлов
подпись, дата должность, ученная степень инициалы, фамилия

Красноярск 2017