Заведующему кафедрой
фундаментальной информатики
А. Г. Смольянову
студента 4 курса
очной формы обучения
(на (бес)платной основе)
направления подготовки
02.03.02 – Фундаментальная информатика
и информационные технологии
факультета математики и
информационных технологий
ФГБОУ ВО «МГУ им. Н.П. Огарёва»
Белякова Вадима Александровича

заявление

Прошу разместить мою выпускную квалификационную работу на тему «Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения» в электронной библиотечной системе университета в полном объеме.

09.06.2020
Дата

Беляков В. А.
Подпись

Заявление о самостоятельном характере выполнения работы

Я, Беляков Вадим Александрович, обучающийся 4 курса направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии, заявляю, что в моей работе на тему «Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения», представленной в Государственную экзаменационную комиссию для публичной защиты, не содержится элементов неправомерных заимствований.

Все прямые заимствования из печатных и электронных источников, а также ранее защищенных письменных работ, кандидатских и докторских диссертций имеют соответствующие ссылки.

Я ознакомлен с действующим в Университете Положением о проверке работ студентов, обучающихся в ФГБОУ ВО «МГУ им. Н.П. Огарёва», на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований является основанием для отрицательного отзыва руководителя работы.

17.06.2020 г.

Беляков Вадим Александрович

Подпись обучающегося Дата

Работа представлена для проверки в Системе «Антиплагиат.ВУЗ»

17.06.2020 г.

Дата представления работы *подпись руководителя*

ЛИСТ ОЗНАКОМЛЕНИЯ

(защита ВКР)

Я, Беляков Вадим Александрович, студент 4 курса очной формы обучения направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии (профиль «Информатика и компьютерные науки») факультета математики и информационных технологий , заявляю, что ознакомлен с пунктом 1.4 Положения о проведении государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры в ФГБОУ ВО «МГУ им. Н. П. Огарёва»:

пункт 1.4. Обучающимся и лицам, привлекаемым к ГИА, во время ее проведения запрещается иметь при себе и использовать средства связи.

09.06. 2020 г. _____

дата подпись студента

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Мордовский государственный
университет имени Н. П. Огарева

СПРАВКА

о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе Антиплагиат.ВУЗ

Автор работы	**Беляков Вадим Александрович**
Факультет, кафедра, номер группы	ФМиИТ, кафедра ФИ, 402 группа
Тип работы	Выпускная квалификационная работа
Название работы	Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения
Название файла	ВКР-анти-БеляковВА-ФИИТ(Б)-2020.docx
Процент заимствования	**12,18%**
Процент цитирования	**0,29%**
Процент оригинальности	**87,52%**
Дата проверки	**10:47:35 17 июня 2020г.**
Модули поиска	Кольцо Вузов; Модуль поиска общеупотребительных выражений; Коллекция Патенты; Модуль поиска перефразирований Интернет; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска "МГУ им. Н. П. Огарева"; Коллекция Медицина; Модуль поиска Интернет; Коллекция Гарант; Коллекция eLIBRARY.RU; Модуль поиска переводных заимствований по интернет (EnRu); Модуль поиска переводных заимствований по elibrary (EnRu); Переводные заимствования; Цитирования; Коллекция РГБ; Сводная коллекция ЭБС; Модуль выделения библиографических записей; Модуль поиска ИПС "Адилет"
Работу проверил	СМОЛЬЯНОВ АНДРЕЙ ГРИГОРЬЕВИЧ ФИО проверяющего
Дата подписи	17.06.2020 г.
	Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

О Т З Ы В

на выпускную квалификационную работу студента 4 курса
направления подготовки 02.03.02 Фундаментальная информатика и информационные технологии
Белякова Вадима Александровича
на тему «Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения»

Настоящая выпускная квалификационная работа посвящена вопросам организации безопасного обмена информацией между пользователями посредством удаленного общения через Интернет.

Структурно работа состоит из введения, трёх глав и заключения.

В работе рассмотрены способы обеспечения конфиденциальности обмена информацией на основе методов шифрования сообщений. Целью данной работы является рассмотрение одного из таких методов, а точнее метода RSA – симметричного криптографического алгоритма с открытым ключом. Данный алгоритм полезен в ситуациях, когда нет возможности безопасно и заблаговременно распространять ключи среди участников общения.

В своей работе автор подчеркивает актуальность систем обмена мгновенными сообщениями (Instant messaging, IM), как совокупности коммуникационных технологий, используемых для обмена текстовыми сообщениями между двумя или более участниками посредством Интернета или других типов сетей. Изучая RSA как способ шифрования, автор обращает внимание на использование этого метода совместно с другими схемами шифрования или для цифровых подписей, которые могут подтвердить подлинность и целостность всего сообщения.

Практическая часть настоящей работы посвящена разработке мобильного приложения-мессенджер под операционную систему Android с применением данного алгоритма. В связи с этим автором были рассмотрены средства разработки под данную операционную систему и вопросы взаимодействия с сервером. Предложенное автором программное решение показало свою эффективность и функциональность. Фрагменты исходного кода программного обеспечения представлены в Приложении к данной работе.

Представленная работа сделана грамотно. Автор достаточно глубоко изучил теоретические вопросы, необходимые для реализации практической части работы. Изложенный в работе материал имеет несомненную практическую значимость с точки зрения теории и практики программирования. Проделанное автором исследование характеризует его как программиста высокой квалификации, владеющего парадигмами, концепциями и методами современного программирования и способного к реализации своих знаний в реальных проектах, связанных с разработкой программного обеспечения.

Считаю, что настоящая работа удовлетворяет требованиям, предъявляемым к выпускным квалификационным работам, а её автор, Беляков В. А., заслуживает оценки «отлично».

Научный руководитель
зав. кафедрой фундаментальной информатики,
доцент А. Г. Смольянов

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЕВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук,
доц.
(подпись) А. Г. Смольянов

«18» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ RSA-ШИФРОВАНИЯ ДЛЯ ОБМЕНА СООБЩЕНИЯМИ ПОСРЕДСТВОМ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

Автор бакалаврской работы (подпись) 09.06.2020 (дата) В. А. Беляков

Обозначение бакалаврской работы БР–02069964–02.03.02–04–20

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. физ.-мат. наук, доц. (подпись) 09.06.2020 (дата) А. Г. Смольянов

Нормоконтролер
канд. техн. наук, доц. (подпись) 17.06.2020 (дата) С. В. Гарина

Саранск
2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЕВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук,
доц.
(подпись) А. Г. Смольянов

«25» декабря 2019 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

(в форме бакалаврской работы)

Студент В. А. Беляков

1 Тема «Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения»

Утверждена приказом № 10037-с от 25.12.2019

2 Срок представления работы к защите 03.07.20

3 Исходные данные для научного исследования:

4 Содержание бакалаврской работы:

4.1 Обзор компонентов реализуемого приложения

4.2 Проблема защиты данных в компьютерных сетях и некоторые методы шифрования

4.3 Проектирование и разработка приложения

5 Приложения: код программы

Руководитель работы _____ (подпись) 25.12.2019 (дата) А. Г. Смольянов

Задание принял к исполнению _____ (подпись) 25.12.2019 (дата) В. А. Беляков

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЕВА»**

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук,
доц.
_____ А. Г. Смольянов
(подпись)
«18» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ RSA-ШИФРОВАНИЯ ДЛЯ ОБМЕНА СООБЩЕНИЯМИ ПОСРЕДСТВОМ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

Автор бакалаврской работы _____ *09.06.2020* В. А. Беляков
(подпись) (дата)

Обозначение бакалаврской работы БР–02069964–02.03.02–04–20

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. физ.-мат. наук, доц. _____ *09.06.2020* А. Г. Смольянов
(подпись) (дата)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ *17.06.2020* В. А. Беляков
(подпись) (дата)

Саранск
2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. Н. П. ОГАРЕВА»

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук,
доц.
_____ А. Г. Смольянов
(подпись)
«25» декабря 2019 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

(в форме бакалаврской работы)

Студент В. А. Беляков

1 Тема «Программная реализация RSA-шифрования для обмена сообщениями посредством мобильного приложения»

Утверждена приказом № 10037-с от 25.12.2019

2 Срок представления работы к защите _____

3 Исходные данные для научного исследования:

4 Содержание бакалаврской работы:

4.1 Обзор компонентов реализуемого приложения

4.2 Проблема защиты данных в компьютерных сетях и некоторые методы шифрования

4.3 Проектирование и разработка приложения

5 Приложения: код программы

Руководитель работы _____ (подпись) _____ (дата) А. Г. Смольянов

Задание принял к исполнению _____ (подпись) _____ (дата) В. А. Беляков

РЕФЕРАТ

Бакалаврская работа содержит 69 страниц, 12 рисунков, 11 использованных источников, 1 приложение.

ШИФРОВАНИЕ, АЛГОРИТМ, МОБИЛЬНАЯ ПЛАТФОРМА, МЕССЕНДЖЕР, БЕЗОПАСНОСТЬ, RSA, КРИПТОГРАФИЯ, ANDROID.

Объектом изучения является создание мобильного приложения для OS Android с использованием криптографического алгоритма RSA для шифрования данных при передаче по сети.

Цель работы – это изучение проблемы безопасности при передаче данных по сети, а также создания приложения для ОС Android.

В процессе работы использовался опыт, приобретенный в процессе обучения на факультете, а также опыт коммерческой разработки на различных проектах.

В результате изучения были получены знания об основных проблемах безопасности при передаче данных по сети, а также было разобрано создание приложения для ОС Android.

Степень внедрения – частичная.

Область применения – в разработке приложений с возможностью обмена данными по сети.

Эффективность – надежный уровень безопасности при передаче данных по сети; хорошая оптимизированное приложение с проработанной архитектурой

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Благодаря развитию мобильного Интернета и тому, что он становится всё доступнее, сегодня практически у каждого есть смартфон, который из-за его огромного функционала и возможностей давно уже перестал быть простым средством общения по мобильной связи. Также, появляются и другие мобильные гаджеты, такие как, например, планшет, который уже не уступает по функциональности и производительности ноутбукам, или даже умные часы. Все это сегодня пользуется огромным спросом и многие уже не представляют другой жизни без этих устройств. Это не удивительно, так как в одном мобильном гаджете собрано всё: от новостной ленты до профессионального программного обеспечения в области графики или музыки. Но самой актуальной функцией смартфона сейчас, безусловно является общение, будь то социальные сети для разговоров с друзьями и близкими или корпоративные мессенджеры для простой и быстрой коммуникации сотрудников компании.

С давних времен существует проблема перехвата сообщений при передаче, которая актуальна по сегодняшний день. Еще до появления интернета люди научились шифровать свои послания, для того чтобы сохранить тайну переписки. Сначала это были примитивные методы, но со временем они только совершенствовались. На сегодняшний день существуют огромные вычислительные мощности, поэтому мы должны иметь очень надежные методы шифрования, которые будут гарантировать то, что в случае перехвата зашифрованного сообщения, его будет очень сложно расшифровать. Целью данной работы будет рассмотрение одного из таких методов, а точнее RSA – асимметричный криптографический алгоритм с открытым ключом. Также, будет реализовано android приложение – мессенджер с применением данного алгоритма для шифрования сообщений. В рамках проекта будет рассмотрена ОС Android, средства разработки под эту операционную систему и взаимодействие с сервером.

1 Обзор компонентов реализуемого приложения

1.1 Мобильная платформа

Мобильная платформа – это программная среда для ноутбуков, планшетов, смартфонов и других портативных устройств. Windows и Mac доминируют среди ноутбуков, а Apple и Google управляют вселенной смартфонов и планшетов. Портативные компьютеры используют процессоры с архитектурой x86, однако в подавляющем большинстве смартфонов и планшетов используются процессоры ARM., например, самые известные и наиболее распространенные кристаллы Apple, чипы Qualcomm Snapdragon и Samsung Exynos – все они основаны на чипах ARM.

С развитием мобильного Интернета и технологий стали появляться все более совершенные и функциональные мобильные устройства. Спрос на них растет огромными темпами. Сегодня портативные гаджеты используются не только в сфере развлечений. Мобильные приложения, существующие в современном информационном обществе условно можно разделить между собой на несколько направлений по функциям, выполняемым ими и сфере использования:

- Интернет магазины
- Игры
- Социальные сети
- Промо-приложения
- Контентные приложения
- Образовательные приложения
- Мессенджеры

Современные информационные технологии дают возможность адаптировать приложения для ряда мобильных устройств и сделать их достаточно удобными в использовании и простыми для понимания пользователей.

Самой популярной мобильной операционной системой на российском рынке сегодня является Android в силу своей доступности. На данной платформе работает огромное количество производителей гаджетов. Главными преимуществами этой системы является открытость, а также быстродействие и высокая производительность.

1.2 Система обмена мгновенными сообщениями

Система обмена мгновенными сообщениями (англ. Instant messaging, IM) – это совокупность коммуникационных технологий, используемых для обмена текстовыми сообщениями между двумя или более участниками посредством Интернета или других типов сетей, например, локальной. Процесс обмена сообщениями в рамках данной системы происходит в режиме реального времени. Важно отметить то, что онлайн-чат и обмен мгновенными сообщениями отличаются от других технологий, таких как электронная почта, из-за предполагаемого отсутствия синхронности сообщений пользователей.

Существуют системы, которые позволяют отправлять так называемые автономные сообщения неавторизованным пользователям, что устраняет некоторые различия между системой обмена мгновенными сообщениями и электронной почтой. IM обеспечивает эффективную и качественную связь, позволяя немедленно получать подтверждение отправки и получения или ответ. Однако, поддержка управления транзакциями не является обязательной и не всегда присутствует в мессенджерах. Зачастую в клиентских приложениях реализуют дополнительные функции, что говорит о том, что современные мессенджеры уже стали полноценными коммуникационными центрами. Например, собеседники могут иметь возможность видеть друг друга во время видео-звонка или общаться напрямую через интернет совершенно бесплатно. Также, многие приложения позволяют обмениваться файлами, хотя, как правило, на обмен накладываются определенные ограничения.

1.2 RSA как способ шифрования

RSA шифрование – это система, которая решает проблему, некогда считавшейся одной из самых больших в криптографии: как можно отправить закодированное сообщение, не имея возможности ранее поделиться ключом с получателем?

При шифровании алгоритмом RSA сообщения шифруются с помощью кода, называемого открытым ключом, которым можно делиться открыто. Из-за определенных математических свойств алгоритма, которые будут рассмотрены далее, зашифрованное сообщение может быть расшифровано только другим ключом, называемым закрытым. У каждого собеседника есть пара ключей, состоящая, соответственно, из открытого и закрытого. Как следует из названия, открытый ключ должен храниться в секрете, а перехват открытого ключа не позволит расшифровать сообщение.

Схемы шифрования с открытым ключом отличаются от шифрования с симметричным ключом, где в процессе шифрования и дешифрования сообщений используется один и тот же закрытый ключ. Эти различия делают алгоритм RSA полезным для связи в ситуациях, когда нет возможности безопасно распространять ключи заранее. Следует добавить, что симметричные алгоритмы имеют свои собственные приложения, такие как шифрование данных для личного использования или для случаев, когда существуют защищенные каналы, по которым могут передаваться закрытые ключи.

Безопасность данного метода шифрования обусловлена сложностью факторизации огромных целых чисел, являющихся произведением двух очень больших простых. Определение исходных чисел, составляющих это произведение, считается практически невозможным из-за времени, которое потребовалось бы даже на современных суперкомпьютерах. По мере увеличения вычислительных мощностей и обнаружения более эффективных алгоритмов факторинга, также повышается способность использовать еще большие числа в качестве ключей шифрования, что существенно усложняет процесс взлома.

Надежность шифрования напрямую зависит от размера ключа, а его удвоение способно обеспечить экспоненциальное увеличение криптографической стойкости, хотя и ухудшает производительность. Благодаря этому сегодня алгоритм RSA является одним из самых надежных и актуальных.

Алгоритм RSA часто используется в сочетании с другими схемами шифрования или для цифровых подписей, которые могут подтвердить подлинность и целостность сообщения. Обычно он не используется для шифрования целых сообщений. Данный метод используется в ряде различных систем, таких как OpenSSL, wolfCrypt, cryptlib и других распространенных криптографических библиотеках. Являясь одной из первых широко используемых схем шифрования с открытым ключом, RSA заложила основы безопасности для большинства сегодняшних способов коммуникации. Он по-прежнему используется в различных веб-браузерах, электронной почте, VPN, мессенджерах и других каналах связи. В таких протоколах, как OpenVPN и TLS могут использовать алгоритм RSA для обмена ключами и установки безопасного соединения между клиентами.

2 Проблема защиты данных в компьютерных сетях и некоторые методы шифрования

2.1 Проблема защиты данных при передаче по сети

Сегодня практически вся коммуникация и обмен данными происходит через Интернет, поэтому важно понимать, насколько важна защита при передаче информации по сети. Эта проблема касается не только крупных компаний, но и обычных пользователей. Совершенствование угроз защиты информации при передаче данных по открытым сетям диктует необходимость современных технологических решений, обеспечивающих защиту.

Шифрование личных данных во время их передачи с одного устройства на другое, например, через Интернет или по проводным, или беспроводным соединениям обеспечивает эффективную защиту от перехвата связи третьей стороной во время передачи данных. Также настоятельно рекомендуется использовать зашифрованную связь при передаче любых данных по беспроводной сети (например, Wi-Fi) или, когда данные будут передаваться по надежной сети.

Данные могут быть преобразованы в зашифрованный формат и переданы по незащищенному каналу связи, но при этом сохраняется возможность остаться защищенными. Примером может служить отправка надлежащим образом зашифрованного вложения по электронной почте. Однако использование безопасных методов связи, таких как безопасность транспортного уровня (TLS) или виртуальную частную сеть (VPN), обеспечит гарантию того, что содержание сообщения не сможет быть раскрыто, даже если его смогут перехватить, при условии, что метод реализован правильно.

Важно помнить, что без дополнительных методов шифрования данные будут зашифрованы только во время передачи и будут храниться в системе получателя в той же форме, в которой они хранятся в системе контроллера данных (т.е. в незашифрованном текстовом виде).

В качестве примера возьмем организацию, которая намерена использовать облачную службу хранения данных в качестве хранилища для архивирования данных. При передаче она использует ранее упомянутый безопасный метод связи TSL для шифрования данных во время передачи, для того, чтобы их нельзя было перехватить. В данном случае TSL будет обеспечивать надлежащую защиту только во время передачи данных. Как только облачный провайдер получает данные, они обычно существуют в расшифрованном состоянии, поэтому организация зашифровывает каждый файл в своей системе перед загрузкой. В этом случае облачный провайдер или другая сторонняя организация не сможет получить доступ к личным данным, пока они хранятся в облаке.

2.2 Криптографическая стойкость алгоритмов шифрования

Разные криптографические формы отличаются друг от друга. Некоторые системы легко взломать или обойти. Другие, наоборот, довольно устойчивы даже к очень серьезным атакам. Способностью криптографической системы противостоять угрозам взлома и защищать информацию, как раз, является криптографическая стойкость.

Криптографическая стойкость (англ. cryptographic strength) – это способность алгоритма шифрования противостоять процессу дешифровки (или криптографической атаки). Криптографическую стойкость нельзя доказать математически в большинстве случаев. Можно лишь сделать доказательства некоторых уязвимостей алгоритма либо попытаться свести задачу раскрытия алгоритма к такой задаче, вычислительная сложность которой будет очень высока, тем самым доказать, то что произвести вскрытие алгоритма не проще чем решить эту задачу. Алгоритм можно считать стойким в том случае, если успешная атака на него требует владения недостижимым на практике объемом вычислительных ресурсов или перехваченных зашифрованных и открытых сообщений, либо же для взлома требуется настолько больших временных затрат

на вскрытие, что к его моменту защищенная информация перестанет быть актуальной.

Криптостойкость зависит от многих факторов, в том числе:

- Секретность ключа
- Сложность угадывания ключа или перебора всех возможных вариантов (поиск ключа). Более длинные ключи, как правило, сложнее подобрать
- Сложность вскрытия алгоритма шифрования без знания ключа (взлом алгоритма шифрования)
- Наличие (или отсутствие) дополнительных способов или обходных путей, с помощью которых зашифрованные данные могут быть расшифрованы без знания ключа
- Свойство открытого текста и знание этих свойств злоумышленником. Например, криптографическая система может быть уязвимой для атаки, если все зашифрованные с ее помощью сообщения начинаются или заканчиваются известным фрагментом открытого текста

Целью криптографического проектирования является разработка алгоритма, вскрыть который будет на столько сложно, что ресурсы и усилия, потраченные на это, будут эквивалентны тем, что потребуются на подбор ключа, пробуя возможные решения один за другим. Это свойство должно сохраняться даже в том случае, если злоумышленник знает что-то о содержимом, зашифрованного с помощью алгоритма. В данном проектировании участвуют довольно сложные математические вычисления.

Демонстрация стойкости криптографической системы является довольно сложным делом, которое требует глубокого анализа. Таким образом, демонстрация лучше всего достигается большим количеством сотрудников. Планирование тестов, обмен, анализ и анализ результатов лучше всего проводить на публичном форуме.

2.3 Преобразование текстовой информации перед шифрованием

Так как алгоритмы шифрования работают только с числами, может возникнуть вопрос, каким образом можно зашифровать ключ типа «4b50c72b46966a811e4d00b8ec766b64» или сообщение «Hello, world!»? На самом деле реальность такова, что вся информация, которую обрабатывают наши компьютеры, хранится в двоичном формате, то есть состоит из нулей и единиц, а мы используем стандарты кодирования, такие как ASCII или Unicode для представления в виде букв, понятных людям.

Это означает, что приведенные выше в качестве примера ключ и сообщение уже существуют в виде чисел, которые можно легко вычислить, например, в алгоритме RSA. Числа, которыми они представлены, намного больше и сложнее для нас, поэтому мы предпочитаем иметь дело с буквенно-цифровыми символами, а не с беспорядочными двоичными числами.

К примеру, возьмем наше сообщение «Hello, world!» и попробуем представить его в виде некоторого числа. Для этого каждому символу сопоставим число, равное коду в таблице ASCII и переведем его в шестнадцатеричный формат: “H” = 48, e = 65, “l” = 6c, “o” = 6f, “w” = 77, “r” = 72, “d” = 64, “_” = 20, “!” = 21. Таким образом из этого сообщения мы получим число, равное 0x48656c6c6f2c20776f726c6421, с которым может работать алгоритм шифрования.

Приведенный выше пример – это достаточно примитивный способ преобразования текста в число и для злоумышленников не составит особого труда расшифровать сообщение. Поэтому на практике применяют различные модификации и схемы заполнения текста дополнительными, случайно сгенерированными данными, например, OAEP.

OAEP (англ. Optimal Asymmetric Encryption Padding, Оптимальное асимметричное шифрование с дополнением) – один из способов дополнения текста для защиты от дешифровки. Его сильные стороны в том, что он добавляет случайности процессу. С помощью этой схемы в зашифрованное сообщение

будет дополнено после выполнения операции с криптографической хеш-функцией:

1. X = (m + несколько нулей для фиксированной длины) ⊕ G (r)
2. Y = r ⊕ H (X), где G и H - криптографические хеш-функции.

Зашифрованный текст здесь будет X || Y

В процессе дешифрования это можно снова обратить с помощью тех же хеш-функций:

1. r = Y ⊕ H (X)
2. m + несколько нулей для фиксированной длины = X ⊕ G (r)

m можно получить после удаления нулей. Это усиливает механизм дополнения и хорошо работает против атак на шифр.

2.4 Обзор некоторых методов шифрования

2.4.1 Симметричное шифрование

Симметричное шифрование – это такой криптографический тип, в котором один и тот же ключ, называемый секретным (закрытым), используется как для шифрования, так и для дешифрования цифровой информации. Объекты, обменивающиеся данными при помощи симметричного шифрования, должны обмениваться ключом, чтобы его можно было использовать в процессе расшифровки.

Теперь подробнее рассмотрим, принцип работы данного типа шифрования. Симметричные методы основываются на использовании одного единственного ключа двумя или более сторонами. Тот же самый ключ применяется для кодирования и декодирования так называемого открытого текста, который представляет собой сообщение или часть кодируемых данных.

Этот метод шифрования отличается от асимметричного шифрования, в котором для шифрования и дешифрования сообщений используется пара ключей, где один из них является открытым, а другой закрытым. Данная

криптографическая схема широко используется, несмотря на свой уровень защиты, поскольку является достаточно легко реализуемой и производительной.

При условии, что схема кодирования достаточно надежна, использовать соответствующий ключ – единственный способ заполучить зашифрованную данным способом информацию. Безопасность симметричных схем шифрования основана на сложности того, что случайным образом угадать или подобрать методом простого перебора соответствующий ключ. По некоторым расчетам, с использованием мощности обычных компьютеров, для подбора ключа длиной 128 бит потребуются миллионы лет. В свою очередь, 256-битеные ключи считаются очень защищенными и теоретически устойчивыми к атакам методом метода перебора с использованием ресурсов квантового компьютера.

Существует два типа симметричных алгоритмов шифрования:

- Блочные алгоритмы. Согласно данной схеме, происходит группировка данных в блоки заранее определенного размер, при этом каждый блок шифруется с использованием соответствующего ключа и алгоритма шифрования. Система хранит данные в своей памяти, ожидая полных блоков.
- Потоковые алгоритмы. В этом случае данные открытого текста шифруются не блоками, а с шагом в 1 бит (1-битной открытый текст зашифровывается в 1-битный зашифрованный текст за одну итерацию)

Примеры алгоритмов симметричного шифрования:

- Advanced Encryption Standard (AES)
- Data Encryption Standard (DES)
- International Data Encryption Algorithm (IDEA)
- Blowfish (альтернатива DES и IDEA)
- Rivest Cipher 4 (RC4)
- Rivest Cipher 5 (RC5)
- Rivest Cipher 6 (RC6)

AES, DES, IDEA, Blowfish, RC5 и RC6 являются блочными шифрами, а RC4 - потоковый шифр.

DES. В «современных» вычислениях он был первым стандартизированным шифром для защиты электронных коммуникаций. Оригинальный DES больше не используется, так как считается слишком «слабым» из-за вычислительной мощности современных компьютеров.

AES – довольно актуальный на сегодня симметричный алгоритм, который изначально был известен как Rijndael. Это стандарт был установлен для шифрования цифровых данных в 2001 году Национальным институтом стандартов и технологий США. Он пришел DES, который использовался с 1977 года. AES обычно имеет размер блока 128 бит, но также может иметь другую длину: AES-192, AES-256.

Несмотря на то, что симметричное шифрование является более старым методом шифрования, оно быстрее и эффективнее, чем асимметричное. Это сказывается из-за проблем производительности и интенсивной загрузкой вычислительных мощностей. Благодаря более высокой скорости и производительности симметричного способа шифрования, он обычно используется для массового шифрования или шифрования больших объемов данных, например, для защиты баз данных.

Некоторые примеры использования симметричной криптографии:

- Платежные приложения, использующие транзакции с картами, где должна быть защищена личная информация для предотвращения кражи личных данных или мошеннических платежей
- Проверки, подтверждающие подлинность отправителя сообщений
- Генерация случайных чисел или хеширование

2.4.2 Асимметричное шифрование

Криптография с открытым ключом, представляет собой схему, которая использует пару связанных ключей - один из которых называется открытым, а

другой закрытым. Они используются для шифрования и дешифрования сообщения с целью его защиты от несанкционированного доступа. Открытый ключ может использоваться любым человеком для шифрования сообщения, таким образом, что его может расшифровать только владелец закрытого ключа, которому было адресовано послание. Закрытый ключ, также известный как секретный ключ, используется только инициатором ключа.

Когда кто-то хочет отправить зашифрованное сообщение, он может извлечь открытый ключ предполагаемого получателя из общего каталога и использовать его для шифрования сообщения перед его отправкой. Получатель сообщения может затем расшифровать сообщение, используя связанный с ним закрытый ключ. С другой стороны, если отправитель зашифровывает сообщение с использованием своего закрытого ключа, то сообщение может быть расшифровано только с использованием открытого ключа этого отправителя, что позволяет аутентифицировать отправителя.

Многие протоколы основаны на асимметричной криптографии, включая протоколы безопасности транспортного уровня (TLS) и уровня защищенных сокетов (SSL), которые делают HTTPS возможным. Процесс шифрования также используется в программах, таких как браузеры и мессенджеры, которым необходимо установить безопасное соединение через небезопасную сеть, такую, как Интернет, или проверить цифровую подпись.

Повышенная безопасность данных является основным преимуществом асимметричной криптографии. Это наиболее безопасный процесс шифрования, поскольку пользователям никогда не требуется раскрывать или делиться своими закрытыми ключами, что снижает шансы злоумышленника обнаружить личный ключ пользователя во время передачи.

Работу ассиметричной схемы криптографии можно описать следующим образом: двумя участниками рабочего процесса шифрования являются отправитель и получатель. У каждого участника имеется своя пара открытых и закрытых ключей. Сначала получатель отправляет свой открытый ключ. Затем открытый текст шифруется отправителем с использованием открытого ключа

получателя; таким образом получаем зашифрованный текст. Затем этот текст отправляется получателю, который его расшифровывает своим закрытым ключом и возвращает к исходному виду.

Очень часто асимметричная криптография применяется для аутентификации электронной информации с применением цифровой подписи. ***Цифровая подпись*** - это математический метод, используемый для проверки подлинности и целостности сообщения, программного обеспечения или цифрового документа. Это цифровой эквивалент рукописной подписи или печати с печатью. Основанные на асимметричной криптографии, цифровые подписи могут предоставлять гарантии подлинности авторства и статуса электронного документа, транзакции или сообщения, а также подтверждать информированное согласие подписывающего лица.

Асимметричная криптография также применяется в системах, в которых многим пользователям может понадобиться шифровать и дешифровать сообщения, в том числе:

- *Электронная почта, мессенджеры* - для шифрования сообщений
- *Криптографические протоколы SSL / TSL* - для установки зашифрованных связей между веб-сайтами и браузерами
- *Биткойн* и другие *криптовалютные системы* основаны на асимметричной криптографии для того, чтобы гарантировать, что средства могут потратить только законные владельцы

Преимущества асимметричной криптографии:

- Устранена проблема с распределением ключей, поскольку нет необходимости каждый раз обмениваться ключами.
- Повышается безопасность, так как закрытые ключи никогда не нужно никому передавать или открывать.
- Использование цифровых подписей, чтобы получатель мог проверить, что сообщение пришло от определенного отправителя.

Недостатки:

- Медленная работа в сравнении с симметричной криптографией
- Если человек теряет свой закрытый ключ, он не может расшифровать полученные сообщения.
- Если хакер получает закрытый ключ человека, то он может прочитать все его сообщения.

Примеры асимметричной криптографии:

- *Алгоритм RSA* - наиболее широко используемый асимметричный алгоритм - встроен в протоколы SSL / TSL. RSA считается достаточно безопасным благодаря сложности факторизации огромных целых чисел.
- *Криптография с эллиптической кривой (ECC)* становится популярной в области безопасности как альтернатива RSA для реализации криптографии с открытым ключом. ECC - это метод шифрования с открытым ключом, основанный на теории эллиптических кривых, который может создавать более быстрые, короткие и эффективные криптографические ключи. ECC генерирует ключи через свойства уравнения эллиптической кривой.

2.5 Теория алгоритма шифрования RSA

В данной главе будет рассмотрено:

- Как было разработано шифрование RSA
- Как оно работает
- Что за ним стоит
- Для чего оно используется
- также, о уязвимостях в безопасности, которые присутствуют в данном методе

Разобравшись в этом мы получим базовые знания о способах защиты цифровой информации в сети Интернет.

Предположим, что нам необходимо рассказать секрет своему приятелю, находясь при этом далеко друг от друга. Мы можем воспользоваться приложением на смартфоне или отослать письмо по почте, но любой из данных каналов связи небезопасен и человек, обладающий сильной мотивацией, сможет перехватить сообщение практически без труда. Безусловно, не стоит рисковать в случае, если этот секрет достаточно важен, потому что сегодня существует огромное количество кибермошенников.

Одним из решений, позволяющих защищать содержимое сообщения от злоумышленников, является шифрование. Оно заключается в том, чтобы преобразовать исходный текст в нечитаемый набор символов. При должной реализации, получить исходное содержимое будет возможно только тому, кто обладает ключом для дешифровки.

Перехватив ключи, злоумышленники получали возможность прочитать абсолютно любое сообщение из переписки, в которой они использовались, что ставило общение под угрозу. Как было упомянуто раннее, из-за отсутствия возможности безопасного обмена ключами было очень сложно обеспечить защищенную связь до появления криптографии с открытым ключом.

Один из первых и наиболее широко используемых алгоритмов шифрования с открытым ключом RSA был разработан в 1977 году учеными из Массачусетского технологического института в честь которых он и был назван. Это были Рональд Ривест, Ади Шамир и Леонард Адлеман. В 1983 эта идея была запатентована и получила большую популярность. Главным элементом идеи была односторонняя функция, которую было бы очень трудно инвертировать.

Далее, при подробном рассмотрении работы алгоритма для упрощения в примерах будут участвовать числа, гораздо меньшие чем используются в действительности.

Для начала необходимо разобраться с концепцией так называемой *односторонней функцией с потайным входом*. RSA шифрование работает за счет условия, что функция легко вычисляется в одном направлении, но почти не вычисляется в обратном. Например, если бы нам было известно, что 701111 – это

произведение двух простых чисел, то мы бы вряд ли смогли выяснить, что это за числа, не прибегая к методу перебора. На самом деле это произведение двух чисел 907 и 773. Это показывает, что некоторые уравнения можно легко вычислить одним способом, но очень сложно обратным. Поскольку связь между этими числами довольно простая для вычисления в одном направлении, но невероятно сложная в обратном, это уравнение известно, как *односторонняя функция*. Стоит обратить внимание на то, что хоть и для человека решение приведенной задачи очень сложно, компьютеры, в свою очередь, могут выдать ответ за очень короткий промежуток времени. Именно поэтому RSA использует гораздо большие числа. Размер простых чисел в реальной реализации алгоритма варьируется, но существуют ключи размером 2048 бит – это число длинной 617 цифр.

Упомянутая выше *односторонняя функция* образует основу для работы схем шифрования с открытым и закрытыми ключами. Ее свойства позволяют раскрывать общие ключи, не подвергая опасности сообщение и не рассекречивая закрытый ключ. Они также позволяют шифровать данные одним ключом таким образом, что дешифровку возможно произвести только с использованием другого ключа из пары.

Первым шагом шифрования сообщения с помощью RSA является генерация ключей. Для этого нам понадобятся два простых числа (p и q), которые выбраны с помощью теста на простоту. Тест на простоту – это алгоритм, который эффективно находит простые числа, например, тест Рабина-Миллера. Простые числа в RSA должны быть очень большими, а также относительно далеко друг от друга. В противном случае шифр будет намного легче взломать.

Допустим, тест на простоту выдал нам простые числа, которые мы рассматривали выше: 907 и 773. Следующим шагом будет нахождение числа n, называемым модулем и равным произведению этих чисел. Таким образом, $n = p \times q$, где $p = 907$ и $q = 773$, следовательно, $n = 907 \times 773 = 701111$.

Как только мы получили n, используем функцию Кармайкла:

$$\lambda\ (n) = \text{НОК}\ (p - 1, q - 1)$$

Здесь $\lambda\ (n)$ представляет собой коэффициент Кармайкла для n, в то время как НОК означает наименьшее общее кратное, которое представляет собой наименьшее число, на которое можно разделить как p, так и q. Теперь подставим наши числа в уравнение:

1. λ (701111) = НОК (907 - 1, 773 - 1)
2. λ (701111) = НОК (906, 772)
3. λ (701,111) = 349716

Сейчас, когда у нас число Кармайкла из наших простых чисел, пришло время выяснить, каков наш открытый ключ. Согласно RSA открытые ключи состоят из простого числа e, а также n. Число e может быть любым между 1 и значением для $\lambda\ (n)$, которое в нашем примере составляет 349716. Поскольку открытый ключ передается открыто, для e не так важно быть случайным числом. На практике значение e обычно составляет 65537, поскольку при случайном выборе гораздо больших чисел шифрование становится намного менее эффективным. В нашем примере мы будем брать небольшие цифры, чтобы сделать вычисления простыми. Возьмем e = 11. Наши зашифрованные данные называются зашифрованным текстом (c). Мы получаем его из нашего открытого текста сообщения (m), применяя открытый ключ по следующей формуле:

$$c = m\text{^}e \bmod n,$$

где c – зашифрованный текст;
m – открытый текст сообщения;
e – произвольное простое число;
n – модуль, равный произведению p и q

Мы уже определились со значением *e*, вычислили *n* и осталось разобраться операцией *mod*. Этот оператор обозначает вычисление остатка от деления, например, 10 mod 3 = 1.

Теперь вернемся к нашему уравнению. Дабы не усложнять вычисления, предположим, что сообщение (*m*), которое мы хотим зашифровать – это всего лишь число 4. Тогда:

1. $c = m^e \bmod n$
2. $c = 4^{11} \bmod 701\,111$
3. $c = 4\,194\,304 \bmod 701\,111$
4. $c = 688\,749$

Следовательно, после применения RSA алгоритма для шифрования нашего сообщения «4» с помощью нашего открытого ключа, это дает нам зашифрованный текст «688749». Теперь мы можем отправить этот текст владельцу пары ключей. Расшифровать его сможет только он, потому что это можно сделать только с помощью закрытого ключа из той же пары ключей. Закрытые ключи состоят из *d* и *n*, где *d* вычисляется по формуле:

$$d = 1/e \bmod \lambda(n)$$

Таким образом, 1/11 mod 349716 = 254 339 – будет нашим закрытым ключом, с помощью которого мы можем расшифровать полученное сообщение, используя следующую формулу:

$$m = c^d \bmod n$$

Подставим значения, которые получили раннее, в данное уравнение и получим 688749^254339 mod 70111 = 4 – наше исходное сообщение.

3 Проектирование и разработка приложения

3.1 Аспекты разработки мобильных приложений под ОС Android

Как видно из рисунка 1, большинство мобильных устройств работает на операционной системе Android. С одной стороны, преимущество разработки приложения под эту ОС заключается в том, что оно будет доступно для огромного количество пользователей. Но с другой – разработчиком приходится учитывать множество различных нюансов, таких как всевозможные размеры дисплеев и разрешений гаджетов, их аппаратные характеристики, разные версии операционной системы и прошивок для того, чтобы приложение выглядело и функционировало одинаково хорошо на все возможных конфигурациях устройств. К примеру, если разработчик не уделит должного внимания к созданию гибкого и универсального для различных экранов интерфейса, пользователь может столкнуться с тем, что у него будут слишком маленькие или наоборот очень большие элементы навигации и текст, что может негативно повлиять на восприятие и сформировать отрицательное впечатление о сервисе.

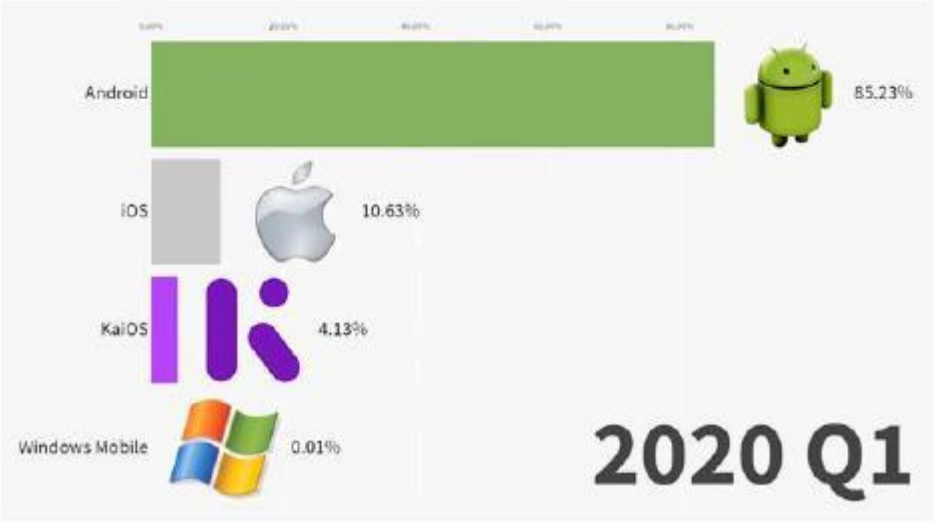

Рисунок 1 – Рейтинг популярности мобильных ОС

Также, такое многообразие гаджетов, работающих на ОС Android порождает так называемую проблему обратной совместимости, которая состоит в том, что устройства, на которых установлена старая версия операционной системы, могут не поддерживать некоторые возможности, доступные на более новых версиях. Существует несколько решений данной проблемы:

- Использование старых компонентов, которые поддерживаются большим количеством версий ОС.
- Написание библиотек обратной совместимости, позволяющих использовать новые компоненты на более ранних версиях ОС.
- Ограничение поддержки более старых версий ОС.

Следующее, что должен знать каждый Android разработчик – это жизненный цикл activity, представленный на рисунке 2. Activity (активность) – это основной компонент приложения, который служит для создания визуального интерфейса. Часто activity ассоциируется с отдельным экраном или окном приложения. Нередко приложение состоит из несколько таких активностей, которые переключаются между собой в процессе использования.

Для навигации между этапами жизненного цикла активности класс *Activity* предоставляет основной набор из следующих методов обратного вызова (от англ. callback):

- *onCreate()* – вызывается при первом создании activity. Этот метод обязательно должен быть определен в классе activity. В нем производится первоначальная настройка activity. В частности, создаются объекты визуального интерфейса.
- *onStart()* – вызывается, когда activity становится видимой для пользователя. В этом методе осуществляется подготовка к выводу activity на экран устройства. Как правило, этот метод не требует переопределения, а всю работу производит встроенный код.
- *onResume()* – вызывается, когда activity становится доступной для взаимодействия с пользователем. Activity остается в этом состоянии,

пока она не потеряет фокус, например, вследствие переключения на другую активность или при выключении экрана устройства.

- *onStop()* – вызывается, когда activity больше не видна пользователю
- *onRestart()* – вызывается после того, как activity пересоздается в следствие остановки. За ним всегда следует вызов метода *onStart()*.
- *onDestroy()* – вызывается перед уничтожением activity.

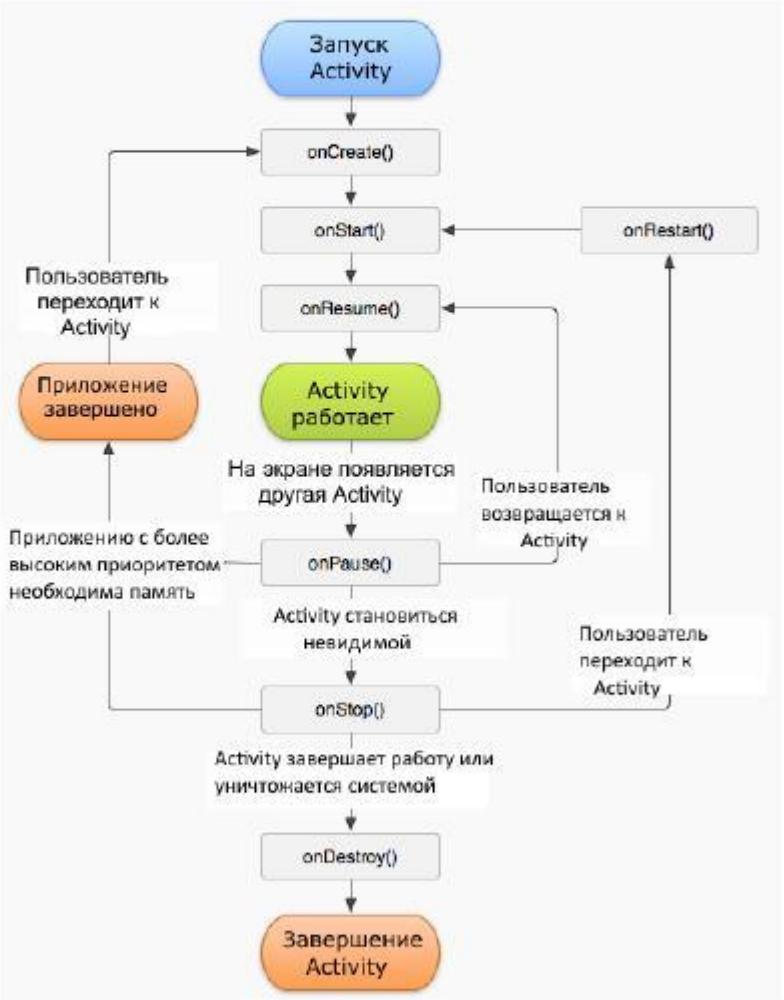

Рисунок 2 – Жизненный цикл Activity

Система вызывает каждый из этих методов обратного вызова, когда activity переходит в новое состояние. Правильная обработка этих вызовов повышает надежность и производительность приложения. Качественная реализация может помочь избежать некоторые проблемы, возникающие в процессе пользования приложением, например:

- Сбой, во время того, как пользователь получает телефонный звонок или переключается на другое приложение.
- Потребление ценных системных ресурсов, когда пользователь в них не нуждается.
- Потеря прогресса пользователя, после того, как он покидает приложение и возвращается через некоторое время.
- Сбой или потеря прогресса при изменении конфигурации, например, поворот устройства или смена языка.

Одним из неочевидных нюансов, с которым сталкивались практически все, когда когда-либо создавал приложения для Android, является пересоздание activity при изменении конфигурации. Если пользователь изменил язык системы или просто перевел устройство в альбомный режим экрана, то весь пользовательский интерфейс должен обновиться для того, чтобы соответствовать текущей конфигурации. Произойдет цепочка вызовов: onPause(), onStop(), onDestroy, а затем onCreate(), onStart(), onResume() уже в новом объекте activity, который был сгенерирован предыдущим экземпляром. Перед уничтожением активности, следует сохранить состояние всех элементов интерфейса для того, чтобы пользователь не потерял прогресс, если он, к примеру, набирал текст.

Поведение activity при изменении конфигурации можно изменить с помощью атрибута android: configChanges, объявленного в манифесте приложения. Тогда вместо перезапуска активности у нас будет выполняться вызов метода onConfigurationChanged(), в котором мы обрабатываем данную ситуацию. Однако, нередка практика, при которой в приложении, например,

блокируют смену ориентации экрана, тем самым минуют проблему пересоздания activity при повороте устройства.

Еще один важный аспект в разработке android приложений – это так называемые *разрешения*. Они необходимы для получения доступа к определенным ресурсам или выполнения определенных операций, например, доступ к геолокации пользователя или к Интернету. Если нам необходимо какое-либо разрешение, оно должно быть указано в манифест-файле приложения с использованием тега <uses-permission>. Для обладателей устройств на ОС Android версии 5.1.1 и ниже при скачивании приложения отображается экран, изображенное на рисунке 3, с информацией о том, какие требуются разрешения и в случае согласия пользователя при установке, система предоставляет доступ ко всем запрашиваемым ресурсам.

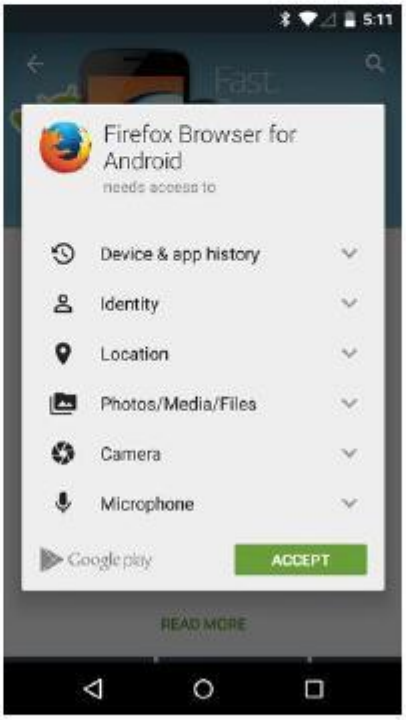

Рисунок 3 - Экран разрешений при установке приложения на ОС Android версии 5.1.1 и ниже

На смартфонах с более новой версией Android дела обстоят несколько иначе. А именно: в версии 6.0 и выше разрешения могут запрашиваться только во время выполнения приложения и только по отдельности, как показано на рисунке 4. Рекомендуется спрашивать пользователя о предоставлении соответствующих доступов по мере их необходимости, а не все вместе при старте приложения.

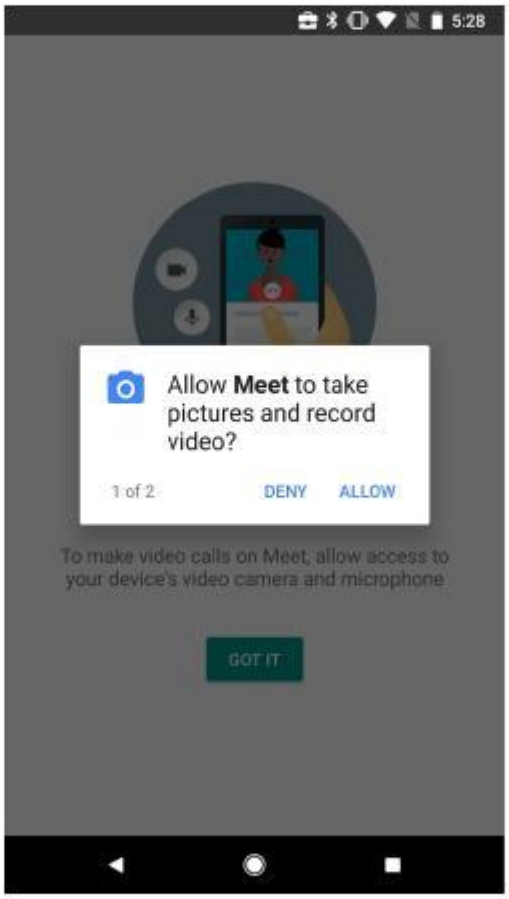

Рисунок 4 - Экран разрешений во время исполнения приложения на ОС Android версии 6.0 и выше

3.2 Обзор используемых инструментов разработки

Основным инструментом любого andoid-разработчика является так называемая *интегрированная среда разработки* (англ. Integrated development environment — IDE). Она представляет из себя программный пакет, включающий в себя основные инструменты, необходимые для написания и тестирования программного обеспечения. Среда IDE предназначена для упрощения разработки ПО, а также помогает выявлять и минимизировать ошибки в коде.

Если раньше приходилось выбирать среду для разработки android приложений, то в настоящее время существует официальная IDE, разработанная компанией IntelliJ Idea и поддерживаемая корпорацией Google – **Android Studio**. Одним из её главных преимуществ, по поему мнению, является возможность предпросмотра экрана верстки на этапе написания кода. Это позволяет видеть то, как будет выглядеть экран на смартфоне без необходимости делать сборку целого проекта, после каждого изменения, при чем можно просмотреть результат для дисплеев разных размеров и с различными разрешениями.

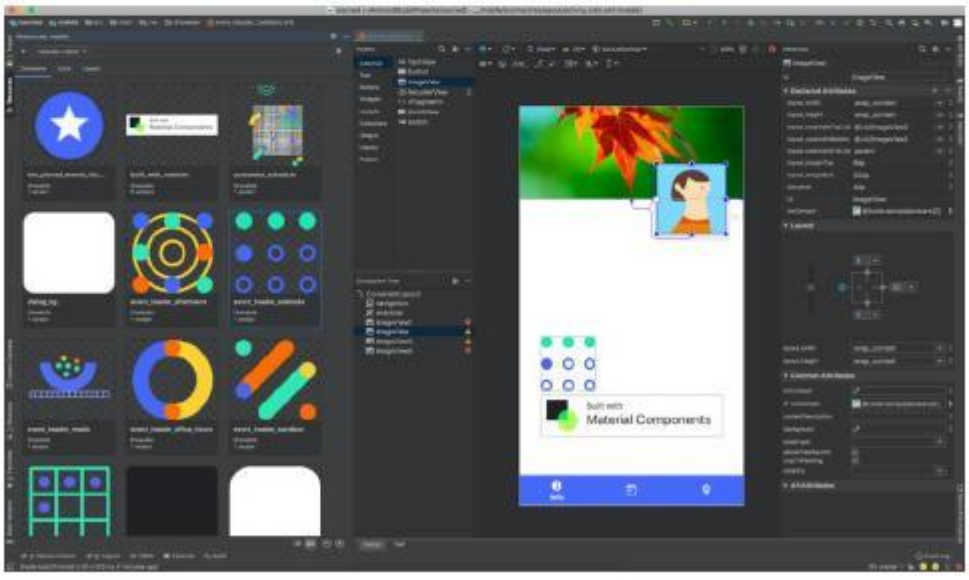

Рисунок 5 - Верстка экрана приложения в Android Studio

Как наглядно продемонстрировано на рисунке 5, также, имеется возможность размещать на экране различные компоненты и элементы интерфейса с помощью панели инструментов, что зачастую значительно упрощает разработку и экономит большое количество времени на написание кода.

Еще одно важное преимущество заключается в том, что Android Studio генерирует большое количество шаблонного кода в начале создания приложения. Как показано на рисунке 6, при старте нового проекта, среда предлагает нам выбрать платформу, например, телефон, планшет или TV. Далее мы указываем начальный тип проекта, зачастую это *Basic Activity*. После этого данная IDE создаст все необходимые файлы, и мы сразу же сможем запустить это приложение. Также, Android Studio генерирует код при создании любых компонентов – от обычных классов и интерфейсов до новых экранов. Это экономит большое количество времени и позволяет сконцентрироваться непосредственно на реализации логики.

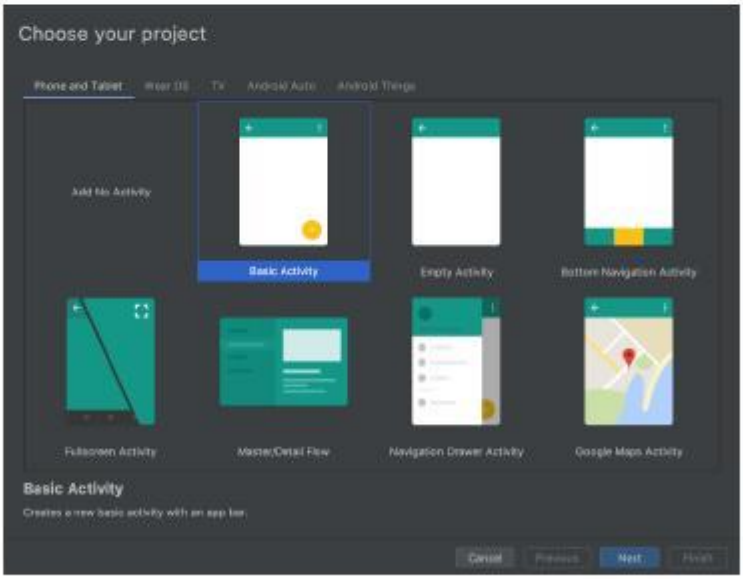

Рисунок 6 - Выбор шаблона проекта в Android Studio

Данная среда обладает еще огромным количеством достоинств, таких, как:

- Мониторинг производительности и потребления ресурсов
- Высокое качество синтаксического анализа кода
- Встроенный эмулятор android-смартфона (AVD)
- Встроенные инструменты для тестирования
- Встроенная интеграция с GitHub
- Позволяет разрабатывать приложения для различных типов устройств (смартфоны, планшеты, телевизоры, смарт часы)

Ко всему прочему, Android Studio обладает великолепной поддержкой со стороны Google и часто получает крупные обновления с новыми функциями и инструментами, а также с улучшением уже существующих. Благодаря этому, данная среда разработки быстро развивается и становится все более удобным средством разработки мобильных приложений.

Что касается языка программирования, то изначально Java – был единственным для разработки под ОС Android. Но позже Google разработала **Kotlin** – новый язык, направленный именно на мобильную разработку. На конференции Google I/O 2019 было анонсировано, что дальнейшая разработка Android будет ориентирована на Kotlin. Документация так же стала дополняться примерами кода на этом языке.

Помимо того, что Google активно развивает Kotlin для разработки под ОС Android, у этого языка есть еще ряд преимуществ:

- Обладает null-безопасными конструкциями, что позволяет избежать частых проверок на null и крашей приложения из-за нулевых ссылок
- Код языка очень лаконичный и писать на нем действительно быстрее и приятнее
- Множество новых функций и конструкций языка
- Возможность сократить код за счет глубокой интеграции в Android Studio
- Более простая работа с асинхронным кодом

Также, одним из главных достоинств языка является полная совместимость с Java. За счет того, что код обоих этих языков компилируется в так называемый байткод, в разработке на Kotlin можно использовать код, написанный на Java и наоборот. Это дает возможность поддерживать старые проекты, разрабатывая на новом языке, а также, использовать огромное количество библиотек, написанных на Java.

3.3 Архитектура Android-приложения и её особенности

По умолчанию в Android не применяется никаких архитектурных шаблонов. Но со временем, при увеличении масштаба приложения, разработка включает в себя все более сложные жизненные циклы и добавляется много логики, что без должной обработки может привести к хаосу. Поэтому разработчики мобильных приложений полагаются на различные архитектурные шаблоны, такие как MVC, MVP и MVVM.

В качестве архитектуры для разрабатываемого приложения был выбран **MVP** (англ. Model-View-Presenter). Это довольно известный и распространенный архитектурный шаблон проектирования среди Android разработчиков. Он позволяет отделить бизнес-логику (Model) от логики представления (Activity/Fragment) добавляя посредника, называемого Presenter.

Как видно из названия, Model-View-Presenter разделен на три разных слоя:

- **Model** – слой данных. Отвечает за управление бизнес-логикой и взаимодействие с сетью и уровнями базы данных
- **View** – слой пользовательского интерфейса. Отображает данные и уведомляет Presenter о действиях пользователя
- **Presenter** – извлекает данные из Model, применяет логику пользовательского интерфейса и управляет состоянием представления

Схема взаимодействия вышеперечисленных слоев представлена на рисунке 7.

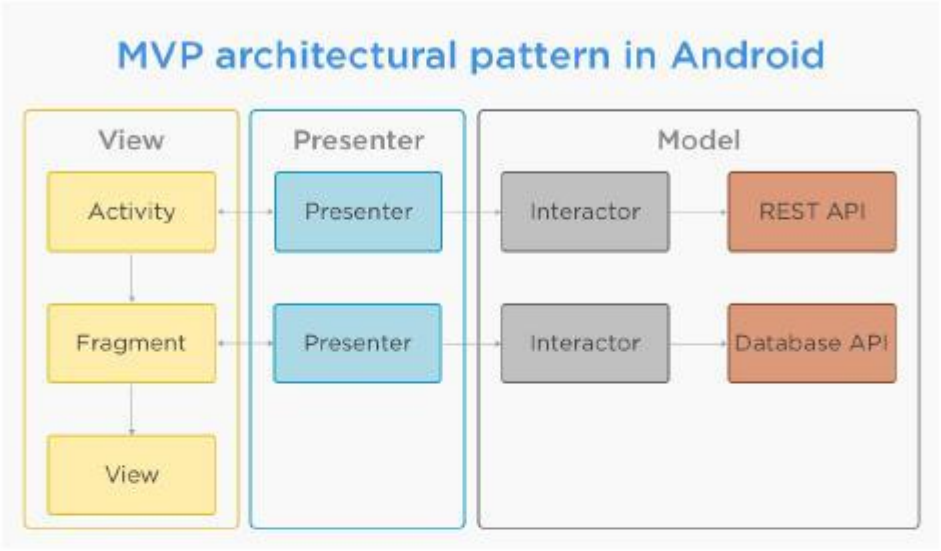

Рисунок 7 - Схема взаимодействия слоев MVP архитектуры Android приложения

Как уже упоминалось выше, Model – это место, где хранятся бизнес-логика и данные приложения. В Android роль модели обычно играет слой доступа к данным, такой как API базы данных или REST API. В нашем случае за это отвечает класс *MessagesProvider*, в котором происходит получение и отправка сообщений на сервер. Model, также, может состоять из компонентов, которые несут ответственность за генерацию и выборку данных. В общем случае, все эти действия должны совершаться в фоновом режиме, поскольку они могут блокировать поток пользовательского интерфейса.

Presenter, выступающий в роли посредника между View и Model, несет ответственность за все, что связанно с логикой представления в приложении. В общем, Presenter выполняет запросы к Model, обновляя View и отвечая на взаимодействия с пользователем.

Основные преимущества данной архитектуры:

- **Упрощается отладка приложения** – MVP применяет три различных уровня абстракций, что облегчает отладку. Более того, поскольку бизнес-логика полностью отделена от View, становится проще выполнять модульное тестирование при разработке.
- **Обеспечивает разделение ответственностей** – MVP отлично справляется с разделением бизнес-логики и логики между классами Activity и Fragment, что, в свою очередь, обеспечивает лучшее разделение задач.
- **Повторное использование** кода – в MVP активно переиспользуется код, поскольку одной View может управлять несколько Presenter'ов. Это, также, способствует упрощению кода.

Таким образом, архитектура MVP делает код приложения легко поддерживаемым и масштабируемым, что способствует более быстрому и эффективному процессу разработки.

3.4 Реализация клиентской части приложения

Реализацию приложения следует начать с создания макета экрана. Несмотря на то, что компоненты интерфейса можно добавлять в коде, который выполняется непосредственно во время работы приложения, верстка в Android разработке традиционно описывается в xml файле с помощью соответствующих тегов. Благодаря такому подходу мы можем наблюдать результат визуальной составляющей в Android Studio, как показано на рисунке 8, без необходимости запускать приложение на устройстве или эмуляторе. Также это позволяет оптимизировать отрисовку интерфейса за счет того, что загрузка ресурсов будет происходить из заранее подготовленного набора, а не созданного динамически во время исполнения.

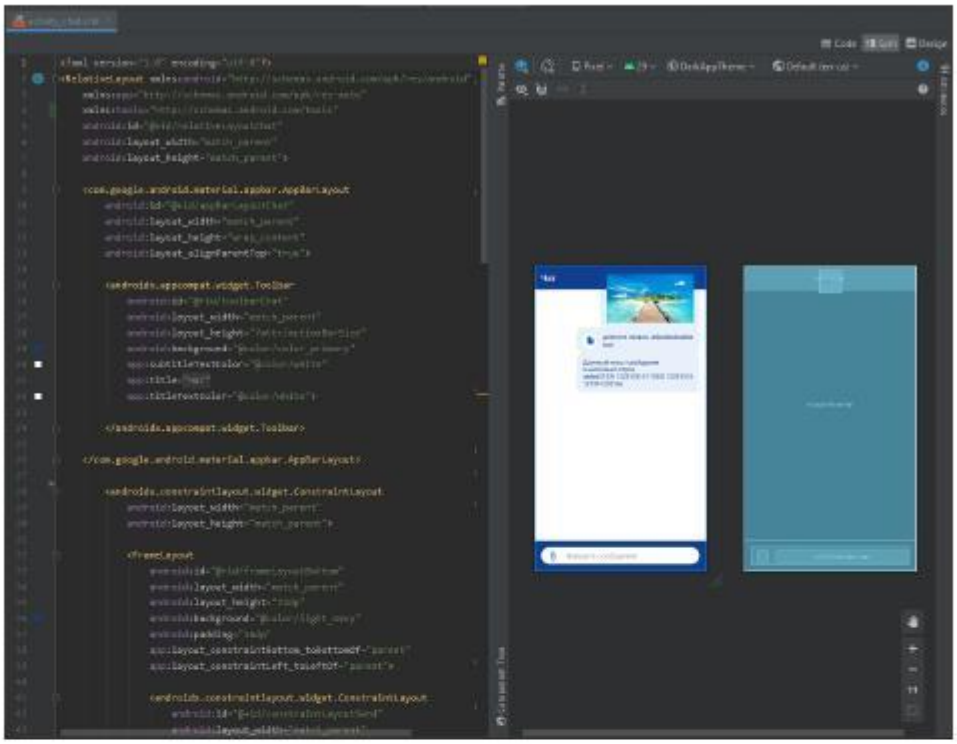

Рисунок 8 - Код верстки и её предпросмотр в Android Studio

Итак, наш экран состоит из следующих компонентов:

- **Toolbar** – Самая верхняя часть экрана, на которой обычно находится заголовок, кнопка возврата на предыдущий экран и иногда всплывающее меню в правой части. В нашем случае там располагается кнопка закрытия чата и имя собеседника.
- **AppCompatEditText** – Текстовое поле для ввода сообщений, а также для отображения уже отправленных или полученных.
- **AppCompatImageView -** Выступает в роли контейнера для изображений. В нашем случае служит для отображения иконок на кнопках отправки сообщений и прикрепления файлов.

- **StateViewFlipper –** Контейнер, который отображает один из вложенных компонентов в зависимости от установленного состояния (данные/загрузка/ошибка).
- **ProgressBar –** Индикатор загрузки.
- **RecyclerView –** Ключевой коннонент в Android разработке для отображения списков элементов. В нашем случае в нем будут отображаться текстовые сообщения, прикрепленные файлы и изображения.
- **FrameLayout, LinearLayout, RelativeLayout, ConstraintLayout** – Контейнеры для группировки компонентов и расположения их на экране. Могут включать в себя другие контейнеры.

Код верстки содержится в файлах, представленных в приложении 1:

- ***activity_chat.xml*** – макет экрана чата
- ***item_message_sent.xml*** – макет для отправленных сообщений
- ***item_message_received.xml*** – макет для полученных сообщений

Для вывода сложных объектов в *RecyclerView* необходимо определить так называемый адаптер, который наследуется от абстрактного класса *RecyclerView.Adapter*. В нашем приложении это класс **MessagesAdapter,** код которого представлен в приложении 1. В этом классе мы должны переопределить следующие методы:

- **onCreateViewHolder()** – В этом методе мы определяем тип сообщения, которое необходимо отобразить и возвращаем соответствующий *ViewHolder* .
- **onBindViewHolder()** – Принимает на вход позицию элемента списка в адаптере и объект ViewHolder, определенный в вышеописанном методе. В нем мы должны передать *вьюхолдеру* наше сообщение, взяв его из списка адаптера по индексу, который соответствует позиции.
- **getItemCount()** – Метод должен возвращать количество элементов, поэтому здесь указывает размер списка.

Далее создаем *выюхолдеры* для каждого типа сообщений: ***FileMessageHolder***, ***ReceivedMessageHolder***, ***SentMessageHolder***. Реализация этих классов представлена в приложении 1. В них мы указываем файл верстки, соответствующий типу сообщения; задаем отображение компонентов и привязываем на некоторые из них действия, которые должны выполняться при нажатии. Например, как продемонстрировано на рисунке 9, отправленные сообщения отображаются справа, а полученные - слева, при чем фоны для текста отличаются друг от друга.

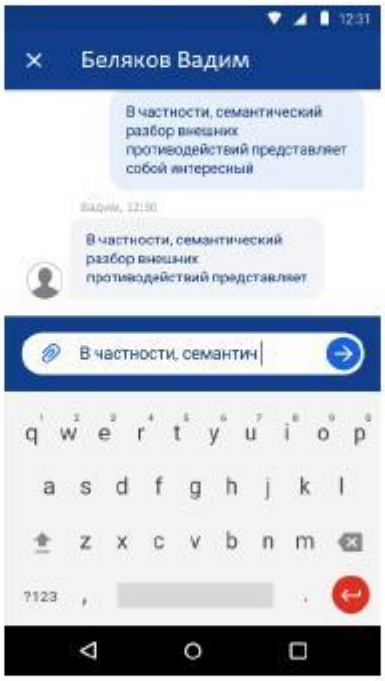

Рисунок 9 - Скриншот обмена сообщениями в чате

Также, если в сообщении было отправлено изображение, то на экране чата должна быть отображена соответствующая миниатюрная картинка. Если сообщение содержит документ, то должно быть указано его название и размер, также должна присутствовать иконка. Скриншот результатов представлен на рисунке 11.

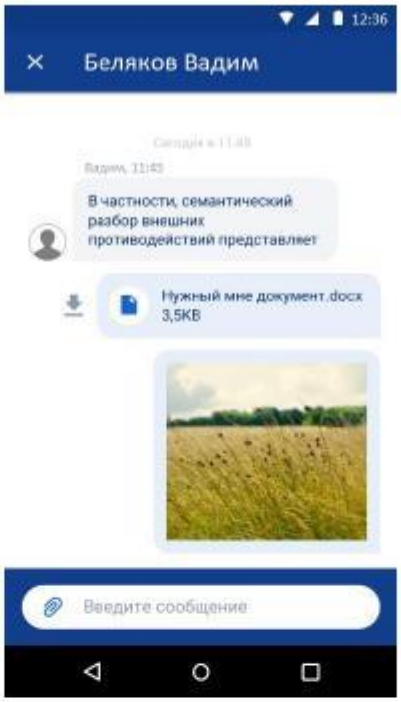

Рисунок 10 - Скриншот сообщений, содержащих документ и изображение

Далее, приступим к классу **ChatActivity**. Этот класс является ключевым в приложении, так как он отвечает за создание экрана с чатом, а также в нем содержится логика взаимодействия всех компонентов приложения. Согласно архитектуре *MVP* он реализует интерфейс ***ChatMvpView***, с помощью которого класс ***ChatPresenter***, выступающий в роли презентера, вызывает методы активити. В методе *onCreate()* мы инициализируем компоненты пользовательского интерфейса, в частности, как продемонстрировано на рисунке 11, добавляем меню прикрепления файла при нажатии на иконку скрепки рядом с текстовым полем для ввода сообщения. Здесь же с помощью *BroadcastReceiver* подписываемся на системные события, связанные с подключением устройства к Интернету. Благодаря этому, в случае отсутствия подключения, мы можем отобразить пользователю соответствующее состояние, как показано на изображении 12.

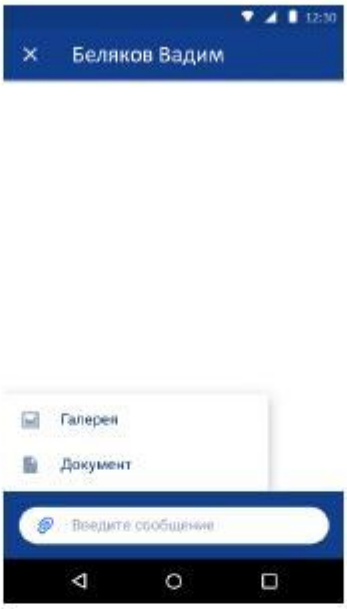

Рисунок 11 - Скриншот меню прикрепления файлов для отправки

Рисунок 12 - Скриншот состояния, когда отсутствует подключение к интернету

Класс **ChatPresenter** является некоторым посредником между пользовательским интерфейсом и слоем данных. В нем собраны методы, которые связанны с действиями пользователя, такими как, например, нажатие на кнопку отправки сообщения. Презентер, в свою очередь, при вызове соответствующего метода, принимает на вход текст сообщения. Далее полученный текст шифруется алгоритмом RSA и передается в класс **ChatProvider**, который отвечает за получение и отправку данных. Также презентер сообщает активити, как поступить с интерфейсом, чтобы отобразить правильное состояние в случае успешной или неудачной отправки сообщения. Аналогичные действия происходят и при получении сообщений.

С помощью **ChatProvider** мы обмениваемся данными с классом **ApiService,** который отправляет запросы и получает ответы от удаленного сервера посредством библиотеки *Retrofit*. Эта библиотека известна каждому Android разработчику, когда-либо работавшему с REST API. **REST** – это общие принципы организации взаимодействия приложения/сайта с сервером посредством протокола HTTP. С помощью *Retrofit* мы можем:

- получать данные от сервера (обычно в формате JSON) с помощью метода **GET**
- отправлять данные на сервер (так же используя JSON формат) с помощью метода **POST**
- изменять некоторые данные на сервере так же с помощью **POST**
- удалять данные на сервере с помощью метода **DELETE**

К примеру, запрос на получение списка сообщений пользователя будет выглядеть следующим образом:

```
@GET("chat/{userID}/messages")
fun getMessages(
    @Path("userID") userId: String,
    @Query("limit") limit: Int = 0
): Single<List<Message>>
```

ЗАКЛЮЧЕНИЕ

В связи с тем, что сегодня наше общение в основном проходит в Интернете посредством браузера или различных приложений-мессенджеров, сейчас очень актуальна проблема защиты передаваемой информации по сети от злоумышленников, которые могут перехватывать сообщения. Поэтому в данной работе было важно рассмотреть способы шифрования, и мы познакомились с некоторыми из них, в частности с RSA – одним из самых надежных и актуальных криптографических алгоритмов на сегодня.

Было разработано мобильное приложение-мессенджер под операционную систему Android с применением данного алгоритма. Также, рассмотрены особенности разработки под данную ОС, основные инструменты, и взаимодействие с сервером. Подробно изучены аспекты Android разработки, которые имею практическую значимость, так как они актуальны на сегодняшний день. Были разобраны основные компоненты Android приложения и принципы работы с ними.

Знания, полученные благодаря изложенной информации об основах шифрования данных при передаче по сети, а также о создании мобильных Android приложений, дают не только понимание того, как это устроено и за счет чего работает, но и способность участвовать в реальных проектах по разработке таких приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Мгновенные сообщения [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://en.wikipedia.org/wiki/Instant_messaging – Загл. с экрана.

2 Как работает RSA шифрование [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://www.comparitech.com/blog/information-security/rsa-encryption – Загл. с экрана.

3 Криптографическая стойкость [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://ru.wikipedia.org/wiki/Криптографическая_стойкость – Загл. с экрана.

4 Криптографическая стойкость [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://no1bc.com/more/articles/cryptographic-strength – Загл. с экрана.

5 Симметричная криптография [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://academy.binance.com/security/what-is-symmetric-key-cryptography – Загл. с экрана.

6 Асимметричная криптография [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://searchsecurity.techtarget.com/definition/asymmetric-cryptography – Загл. с экрана.

7 Android Studio [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://developer.android.com/studio/intro – Загл. с экрана.

8 Kotlin для Android [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://developer.android.com/kotlin/first – Загл. с экрана.

9 Жизненный цикл Activity [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://developer.android.com/guide/components/activities/activity-lifecycle – Загл. с экрана.

10 Архитектурные паттерны Android [Электронный ресурс]. – [Б. м. : Б. и.], [М., 201-]. – Режим доступа: https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b – Загл. с экрана.

11 Разрешения Android [Электронный ресурс]. – [М., 201-]. – Режим доступа: https://developer.android.com/guide/topics/permissions/overview – Загл. с экрана.

ПРИЛОЖЕНИЕ А Код программы

(Обязательное)

Листинг файла **activity_chat.xml**

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/relativeLayoutChat"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBarLayoutChat"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbarChat"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="@color/color_primary"
            app:subtitleTextColor="@color/white"
            app:title="@string/chat"
            app:titleTextColor="@color/white">

        </androidx.appcompat.widget.Toolbar>

    </com.google.android.material.appbar.AppBarLayout>

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <FrameLayout
                android:id="@+id/frameLayoutBottom"
                android:layout_width="match_parent"
                android:layout_height="72dp"
                android:background="@color/light_navy"
                android:padding="16dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintLeft_toLeftOf="parent">

                <androidx.constraintlayout.widget.ConstraintLayout
                    android:id="@+id/constraintLayoutSend"
                    android:layout_width="match_parent"
                    android:layout_height="40dp"
                    android:background="@drawable/bg_round_white_2"
                    android:gravity="center_vertical"
                    android:orientation="horizontal">

                    <FrameLayout
                        android:id="@+id/frameLayoutSend"
                        android:layout_width="32dp"
                        android:layout_height="32dp"
                        android:layout_marginEnd="4dp"
                        android:layout_marginRight="4dp"
```

```
                android:background="@drawable/bg_azul_oval"
                android:visibility="gone"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintRight_toRightOf="parent"
                app:layout_constraintTop_toTopOf="parent">

                <androidx.appcompat.widget.AppCompatImageView
                    android:id="@+id/imageViewSendMessage"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    app:srcCompat="@drawable/ic_arrow_right" />

            </FrameLayout>

            <androidx.appcompat.widget.AppCompatEditText
                android:id="@+id/editTextInputMessage"
                fontPath="fonts/Roboto-Regular.ttf"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginStart="20dp"
                android:background="@android:color/transparent"
                android:ems="10"
                android:hint="@string/input_message"
                android:inputType="textMultiLine|textCapSentences"
                android:maxLength="2000"
                android:maxLines="4"
                android:minLines="1"
                android:nextFocusLeft="@id/editTextInputMessage"
                android:nextFocusUp="@id/editTextInputMessage"
                android:scrollHorizontally="false"
                android:singleLine="false"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintLeft_toRightOf="@id/imageViewAttachMenu"
                app:layout_constraintRight_toLeftOf="@id/frameLayoutSend"
                app:layout_constraintTop_toTopOf="parent">

            </androidx.appcompat.widget.AppCompatEditText>

            <androidx.appcompat.widget.AppCompatImageView
                android:id="@+id/imageViewAttachMenu"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="16dp"
                android:layout_marginStart="16dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:srcCompat="@drawable/ic_attach" />

        </androidx.constraintlayout.widget.ConstraintLayout>

    </FrameLayout>

    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"
                android:layout_gravity="center_horizontal"
                android:layout_marginTop="10dp"
                android:padding="3dp"
                android:visibility="visible"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

            <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recyclerViewChat"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_marginBottom="72dp"
                android:clipToPadding="false"
                android:paddingBottom="5dp"
                android:scrollbars="none"
                tools:itemCount="1"
                tools:listitem="@layout/item_message_sent"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="1.0">

            </androidx.recyclerview.widget.RecyclerView>

        </androidx.constraintlayout.widget.ConstraintLayout>

</RelativeLayout>
```

Листинг файла **item_message_sent.xml**

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tool="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="100dp"
    android:layout_marginLeft="100dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp">

    <FrameLayout
        android:id="@+id/frameLayoutAttachedImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/bg_sent_message"
        android:visibility="visible"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.appcompat.widget.AppCompatImageView
            android:id="@+id/imageViewAttachedImage"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:adjustViewBounds="true"
```

Продолжение ПРИЛОЖЕНИЯ А

```
            android:scaleType="fitXY"
            android:maxWidth="208dp"
            tool:src="@drawable/test_2" />

    </FrameLayout>

    <androidx.appcompat.widget.LinearLayoutCompat
        android:id="@+id/linearLayoutMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/bg_sent_message"
        android:orientation="vertical"
        android:visibility="visible"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/relativeLayoutAttachedFile">

        <androidx.appcompat.widget.AppCompatTextView
            android:id="@+id/textViewMessageText"
            fontPath="fonts/Roboto-Regular.ttf"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textColor="@color/marine_blue"
            tool:text="Длинный текст сообщения в несколько строк
sadad31231132312311132311323131312133123321as" />

    </androidx.appcompat.widget.LinearLayoutCompat>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/relativeLayoutAttachedFile"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/bg_sent_message"
        android:visibility="visible"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/frameLayoutAttachedImage">

        <androidx.appcompat.widget.LinearLayoutCompat
            android:id="@+id/linearLayoutFile"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toRightOf="@id/constraintLayoutImage"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent">

            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/textViewFileName"
                fontPath="fonts/Roboto-Regular.ttf"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginStart="12dp"
                android:layout_marginLeft="12dp"
                android:ellipsize="middle"
                android:singleLine="true"
                android:textColor="@color/marine_blue"
                tool:text="длинное название документа
dsadadasdasdasdsaasddsaadsdasdsadsadsadsadas" />
```

```
            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/textViewFileSize"
                fontPath="fonts/Roboto-Regular.ttf"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="12dp"
                android:layout_marginLeft="12dp"
                android:ellipsize="middle"
                android:singleLine="true"
                android:textColor="@color/marine_blue"
                tool:text="text" />
        </androidx.appcompat.widget.LinearLayoutCompat>

        <androidx.constraintlayout.widget.ConstraintLayout
            android:id="@+id/constraintLayoutImage"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="5dp"
            android:layout_marginRight="5dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toTopOf="parent">

            <ProgressBar
                android:id="@+id/progressBarFileUpload"
                style="?android:attr/progressBarStyleHorizontal"
                android:layout_width="36dp"
                android:layout_height="36dp"
                android:background="@drawable/bg_circle_shape"
                android:indeterminate="false"
                android:max="100"
                android:progress="0"
                android:progressDrawable="@drawable/circular_progress_bar"
                android:scaleType="centerCrop"
                android:visibility="visible"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

            <androidx.appcompat.widget.AppCompatImageView
                android:id="@+id/imageViewFile"
                android:layout_width="20dp"
                android:layout_height="20dp"
                android:adjustViewBounds="false"
                android:scaleType="centerCrop"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintDimensionRatio="1:1"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:srcCompat="@drawable/ic_do" />

        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Продолжение ПРИЛОЖЕНИЯ А

Листинг файла **item_message_received.xml**

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="54dp"
    android:layout_marginRight="54dp">

    <FrameLayout
        android:id="@+id/frameLayoutAttachedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="62dp"
        android:layout_marginLeft="62dp"
        android:background="@drawable/bg_receive_message"
        android:visibility="gone"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textViewOperatorName">

        <androidx.appcompat.widget.AppCompatImageView
            android:id="@+id/imageViewAttachedImage"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:adjustViewBounds="true"
            android:maxWidth="208dp"
            tools:src="@drawable/test_2" />

    </FrameLayout>

    <RelativeLayout
        android:id="@+id/relativeLayoutAttachedFile"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="62dp"
        android:layout_marginLeft="62dp"
        android:background="@drawable/bg_receive_message"
        android:visibility="visible"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textViewOperatorName">

        <androidx.appcompat.widget.LinearLayoutCompat
            android:id="@+id/linearLayoutFile"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="12dp"
            android:layout_marginLeft="12dp"
            android:layout_toEndOf="@id/constraintLayoutImage"
            android:layout_toRightOf="@id/constraintLayoutImage"
            android:orientation="vertical">

            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/textViewFileName"
                fontPath="fonts/Roboto-Regular.ttf"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
```

```
            android:ellipsize="middle"
            android:singleLine="true"
            android:textColor="@color/marine_blue"
            tools:text="kappa" />

        <androidx.appcompat.widget.AppCompatTextView
            android:id="@+id/textViewFileSize"
            fontPath="fonts/Roboto-Regular.ttf"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ellipsize="middle"
            android:singleLine="true"
            android:textColor="@color/marine_blue"
            tools:text="text" />
    </androidx.appcompat.widget.LinearLayoutCompat>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/constraintLayoutImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="5dp"
        android:layout_marginRight="5dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <ProgressBar
            android:id="@+id/progressBarFileUpload"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="36dp"
            android:layout_height="36dp"
            android:background="@drawable/bg_circle_shape"
            android:indeterminate="false"
            android:max="100"
            android:progress="0"
            android:progressDrawable="@drawable/circular_progress_bar"

            android:scaleType="centerCrop"
            android:visibility="visible"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <androidx.appcompat.widget.AppCompatImageView
            android:id="@+id/imageViewFile"
            android:layout_width="20dp"
            android:layout_height="20dp"
            android:adjustViewBounds="false"
            android:scaleType="centerCrop"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintDimensionRatio="1:1"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:srcCompat="@drawable/ic_do" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</RelativeLayout>
```

```
    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/textViewOperatorName"
        fontPath="fonts/Roboto-Regular.ttf"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="62dp"
        android:layout_marginLeft="62dp"
        android:paddingBottom="2dp"
        android:visibility="visible"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Сергей, старший ">

    </androidx.appcompat.widget.AppCompatTextView>

    <androidx.appcompat.widget.LinearLayoutCompat
        android:id="@+id/linearLayoutMessage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="62dp"
        android:layout_marginLeft="62dp"
        android:orientation="vertical"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textViewOperatorName">

        <androidx.appcompat.widget.LinearLayoutCompat
            android:id="@+id/message_body"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@drawable/bg_receive_message"
            android:minWidth="55dp"
            android:orientation="vertical">

            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/textViewMessageText"
                fontPath="fonts/Roboto-Regular.ttf"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textColor="@color/marine_blue"

tools:text="Sadad31223113232113232133313211231323123123131231221312231132132312321132231312321as" />

        </androidx.appcompat.widget.LinearLayoutCompat>

        <androidx.cardview.widget.CardView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            app:cardCornerRadius="20dp">

            <androidx.appcompat.widget.LinearLayoutCompat
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical">

                <androidx.appcompat.widget.AppCompatImageView
                    android:id="@+id/attached_image"
```

```
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:scaleType="fitXY"
                    android:visibility="visible"
                    tools:ignore="ContentDescription" />

            </androidx.appcompat.widget.LinearLayoutCompat>
        </androidx.cardview.widget.CardView>
    </androidx.appcompat.widget.LinearLayoutCompat>

    <de.hdodenhof.circleimageview.CircleImageView
        android:id="@+id/imageViewAvatar"
        android:layout_width="42dp"
        android:layout_height="42dp"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Листинг файла **ChatMvpView.kt**

```
interface ChatMvpView : MvpView {
    fun setStateLoading()
    fun setStateData()
    fun setStateInternetError()
    fun initViews()
}
```

Листинг файла **ChatActivity.kt**

```
class ChatActivity : BaseDaggerActivity(), ChatMvpView {

    companion object {
        private const val PICK_FILE = 0
        private const val PICK_PICTURE = 1

        fun createStartIntent(context: Context, userId: String) =
                Intent(context, ChatActivity::class.java)
    }

    @Inject
    lateinit var chatPresenter: ChatPresenter

    private var listController: ListController? = null

    private lateinit var connectionStateReceiver: BroadcastReceiver

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chat)
        activityComponent().inject(this)
        chatPresenter.attachView(this)
```

```
        connectionStateReceiver = object : BroadcastReceiver() {
            override fun onReceive(context: Context?, intent: Intent?) {
                val connectivityManager =
getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
                val activeNetwork: NetworkInfo? = connectivityManager.activeNetworkInfo
                val isConnected: Boolean = activeNetwork?.isConnectedOrConnecting ==
true
                chatPresenter.onConnectionChanged(isConnected)
            }
        }
        chatPresenter.onCreateView(this)

        val intentFilter = IntentFilter()
        intentFilter.addAction(ConnectivityManager.CONNECTIVITY_ACTION)
        registerReceiver(connectionStateReceiver, intentFilter)
    }

    override fun onDestroy() {
        chatPresenter.destroySession()
        super.onDestroy()
    }

    override fun initViews() {
        toolbarChat.setNavigationIcon(R.drawable.ic_back_white_2)
        toolbarChat.setNavigationOnClickListener { onBackPressed() }
        initChatView()
        initEditText()
        initSendButton()
        initChatMenu()
        setStateData()
    }

    private fun selectFromGallery() {
        val intent = Intent(Intent.ACTION_GET_CONTENT)
        intent.type = "image/*"
        intent.action = Intent.ACTION_GET_CONTENT

        try {
            startActivityForResult(
                    Intent.createChooser(
                            intent,
                            getString(R.string.chat_file_chooser_title)
                    ),
                    PICK_PICTURE
            )
        } catch (e: android.content.ActivityNotFoundException) {
            Toast.makeText(
                    this,
                    getString(R.string.chat_file_chooser_not_found),
                    Toast.LENGTH_SHORT
            ).show()
        }
    }

    private fun selectFromDocuments() {
        val intent = Intent(Intent.ACTION_GET_CONTENT)
        intent.type = "*/*"
```

```
        intent.addCategory(Intent.CATEGORY_OPENABLE)

        try {
            startActivityForResult(
                    Intent.createChooser(
                            intent,
                            getString(R.string.chat_file_chooser_title)
                    ),
                    PICK_PICTURE
            )
        } catch (e: android.content.ActivityNotFoundException) {
            Toast.makeText(
                    this,
                    getString(R.string.chat_file_chooser_not_found),
                    Toast.LENGTH_SHORT
            ).show()
        }
    }

    private fun initChatView() {
        progressBar.gone()
        recyclerViewChat.visible()
        recyclerViewChat.itemAnimator = null
        listController =
                ListController.install(
                        this, recyclerViewChat, progressBar,
                        chatPresenter.getChatMessagesStream(args.userId)
                )
    }

    private fun initEditText() {
        editTextInputMessage.addTextChangedListener(object : TextWatcher {
            override fun onTextChanged(charSequence: CharSequence?, start: Int, before:
Int, count: Int) {
                if (charSequence?.isEmpty() != false) {
                    frameLayoutSend.gone()
                } else {
                    frameLayoutSend.visible()
                }
            }

            override fun afterTextChanged(s: Editable?) {}
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int,
after: Int) {}
        })
    }

    private fun initSendButton() {
        imageViewSendMessage.setOnClickListener {
            val message = editTextInputMessage.text.toString()
            editTextInputMessage.text?.clear()
            if (message.isNotEmpty()) {
                chatPresenter.sendMessage(message)
            }
        }
    }

    private fun initChatMenu() {
```

```
        val popupView = layoutInflater.inflate(R.layout.item_popup_menu_chat, null)
        val popupWindow = PopupWindow(popupView, 280.dpToPx(),
WindowManager.LayoutParams.WRAP_CONTENT)
        popupWindow.inputMethodMode = PopupWindow.INPUT_METHOD_NEEDED
        popupWindow.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

        popupView.constraintLayoutGallery.setOnClickListener {
            selectFromGallery()
            popupWindow.dismiss()
        }

        popupView.constraintLayoutDocument.setOnClickListener {
            selectFromDocuments()
            popupWindow.dismiss()
        }

        imageViewAttachMenu.setOnSingleClickListener {
            if (popupWindow.isShowing) {
                popupWindow.dismiss()
            } else {
                popupWindow.isFocusable = true
                popupWindow.isTouchable = true
                popupWindow.isOutsideTouchable = true
                popupWindow.showAtLocation(
                        popupView, Gravity.NO_GRAVITY,
                        0,
                        frameLayoutBottom.y.toInt() -
                                frameLayoutBottom.measuredHeight -

resources.getDimensionPixelSize(R.dimen.chat_edit_text_padding) - 14.dpToPx())
            }
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        if (requestCode == PICK_FILE || requestCode == PICK_PICTURE) {
            if (resultCode == Activity.RESULT_OK) {
                val uri = data!!.data
                if (uri != null) {
                    val mime = contentResolver.getType(uri)
                    val extension = if (mime == null) null else
MimeTypeMap.getSingleton().getExtensionFromMimeType(mime)
                    var name = if (extension == null) null else uri.lastPathSegment +
"." + extension
                    var file: File? = null
                    try {
                        val inputStream = contentResolver.openInputStream(uri)
                        if (inputStream != null) {
                            file = File.createTempFile(
                                    "webim",
                                    extension, cacheDir
                            )
                            writeToFile(file, inputStream)
                            val cursor = contentResolver.query(
                                    uri,
                                    null,
                                    null,
                                    null,
```

```
                                null
                            )
                            if (cursor != null && cursor.moveToFirst()) {
                                name = cursor.getString(
cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME)
                                )
                                cursor.close()
                            }
                        }
                    } catch (e: IOException) {
                        if (file != null) {
                            file.delete()
                            file = null
                        }
                    }
                    if (file != null && name != null && mime != null) {
                        val fileToUpload: File = file
                        chatPresenter.sendFile(
                                Message(fileToUpload),
                                object : SendFileCallback {
                                    override fun onProgress(id: Message.Id, sentBytes:
Long) {}
                                    override fun onSuccess(id: Message.Id) {
                                        fileToUpload.delete()
                                    }

                                    override fun onFailure(
                                        id: Message.Id,
                                        error: WebimError<SendFileError>
                                    ) {
                                        fileToUpload.delete()
                                        val message: String = when (error.errorType) {
                                            SendFileError.FILE_TYPE_NOT_ALLOWED ->
getString(
                                                    R.string.file_upload_failed_type
                                            )
                                            SendFileError.FILE_SIZE_EXCEEDED ->
getString(
                                                    R.string.file_upload_failed_size
                                            )
                                            SendFileError.FILE_NAME_INCORRECT ->
getString(
                                                    R.string.file_upload_failed_name
                                            )
                                            SendFileError.CHAT_NOT_STARTED ->
getString(
                                                    R.string.file_upload_failed_no_chat
                                            )
                                            SendFileError.UPLOADED_FILE_NOT_FOUND ->
getString(
                                                    R.string.file_upload_failed_unknown
                                            )
                                            else -> getString(
                                                    R.string.file_upload_failed_unknown
                                            )
                                        }
                                        Toast.makeText(
```

```
                                        this@ChatActivity,
                                        message,
                                        Toast.LENGTH_SHORT
                                    ).show()
                                }
                            }
                        )
                        return
                    }
                }
            }
            if (resultCode != Activity.RESULT_CANCELED) {
                Toast.makeText(
                        this,
                        getString(R.string.file_selection_failed),
                        Toast.LENGTH_SHORT
                ).show()
            }
            return
        }
        super.onActivityResult(requestCode, resultCode, data)
    }

    private fun writeToFile(to: File, from: InputStream) {
        val buffer = ByteArray(4096)
        var out: OutputStream? = null
        try {
            out = FileOutputStream(to)
            var read = from.read(buffer)
            while (read != -1) {
                out.write(buffer, 0, read)
                read = from.read(buffer)
            }
        } finally {
            from.close()
            out?.close()
        }
    }

    private class ListController private constructor(
        chatActivity: ChatActivity,
        private val recyclerView: RecyclerView,
        private val progressBar: ProgressBar,
        messageStream: MessageStream
    ) : MessageListener {

        companion object {
            const val MESSAGES_PER_REQUEST = 25
            fun install(
                chatActivity: ChatActivity,
                recyclerView: RecyclerView,
                progressBar: ProgressBar,
                messageStream: MessageStream
            ): ListController {
                return ListController(chatActivity, recyclerView, progressBar,
messageStream)
            }
        }
```

```
        private val adapter: MessagesAdapter
        private val tracker: MessageTracker
        private val layoutManager: LinearLayoutManager = LinearLayoutManager(
                chatActivity, LinearLayoutManager.VERTICAL, true
        )
        private val scroolListener: EndlessScrollListener

        private var requestingMessages: Boolean = false

        init {
            this.layoutManager.stackFromEnd = false
            this.adapter = MessagesAdapter()
            this.recyclerView.layoutManager = layoutManager
            this.recyclerView.adapter = this.adapter
            this.tracker = messageStream.newMessageTracker(this)
            this.scroolListener = object : EndlessScrollListener(10) {
                override fun onLoadMore(totalItemsCount: Int) {
                    requestMore()
                }
            }
            scroolListener.setLoading(true)
            scroolListener.setAdapter(adapter)
            recyclerView.addOnScrollListener(scroolListener)
            requestMore(true)
        }

        private fun requestMore(flag: Boolean = false) {
            requestingMessages = true
            progressBar.visible()
            if (flag) {
                recyclerView.gone()
            }
            tracker.getNextMessages(MESSAGES_PER_REQUEST) { received ->
                requestingMessages = false
                progressBar.gone()
                recyclerView.visible()

                if (received.size != 0) {
                    adapter.addAll(0, received)
                    adapter.notifyItemRangeInserted(adapter.itemCount - 1,
received.size)
                    if (flag) {
                        recyclerView.postDelayed({
                            recyclerView.smoothScrollToPosition(0)
                            val itemCount = layoutManager.itemCount
                            val lastItemVisible =
layoutManager.findLastVisibleItemPosition() + 1
                            if (itemCount == lastItemVisible) {
                                requestMore()
                            }
                        }, 100)
                    }
                    scroolListener.setLoading(false)
                }
            }
        }
```

```
        override fun messageAdded(before: Message?, message: Message) {
            val ind = if (before == null) 0 else adapter.indexOf(before)
            if (ind <= 0) {
                adapter.addMessage(message)
                adapter.notifyItemInserted(0)
                recyclerView.smoothScrollToPosition(0)
            } else {
                adapter.addMessage(ind, message)
                adapter.notifyItemInserted(adapter.itemCount - ind - 1)
            }
        }

        override fun messageRemoved(message: Message) {
            val pos = adapter.indexOf(message)
            if (pos != -1) {
                adapter.removeMessage(pos)
                adapter.notifyItemRemoved(adapter.itemCount - pos)
            }
        }

        override fun messageChanged(from: Message, to: Message) {
            val ind = adapter.lastIndexOf(from)
            if (ind != -1) {
                adapter.setMessage(ind, to)
                adapter.notifyItemChanged(adapter.itemCount - ind - 1, 42)
                recyclerView.itemAnimator = null
            }
        }

        override fun allMessagesRemoved() {
            val size = adapter.itemCount
            adapter.removeAll()
            adapter.notifyItemRangeRemoved(0, size)
            if (!requestingMessages) {
                requestMore()
            }
        }
    }

    override fun setStateLoading() {
        stateViewFlipperChat.setStateLoading()
    }

    override fun setStateData() {
        stateViewFlipperChat.setStateData()
    }

    override fun setStateInternetError() {
        stateViewFlipperChat.setStateInternetError {}
        hideSoftKeyboard()
    }
}
```

Листинг файла **MessagesAdapter.kt**

```
class MessagesAdapter : RecyclerView.Adapter<MessagesAdapter.MessageHolder>() {
```

```
    private val messageList = mutableListOf<Message>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MessageHolder {
        val messageType = Message.Type.values()[viewType]
        val view = LayoutInflater.from(parent.context).inflate(
                getLayout(messageType),
                parent, false
        )
        return when (messageType) {
            Message.Type.RECIEVED -> ReceivedMessageHolder(view)
            Message.Type.SENT -> SentMessageHolder(view)
            else -> MessageHolder(view)
        }
    }

    override fun getItemCount(): Int {
        return messageList.size
    }

    override fun onBindViewHolder(holder: MessageHolder, position: Int) {
        val message = messageList[messageList.size - position - 1]
        holder.bind(message)
    }

    override fun getItemViewType(position: Int): Int {
        val message = messageList[messageList.size - position - 1]
        if (message.sendStatus == Message.SendStatus.SENDING) {
            return Message.Type.SENT.ordinal
        }

        return when (message.type) {
            Message.Type.RECIEVED -> Message.Type.RECIEVED.ordinal
            Message.Type.SENT -> Message.Type.SENT.ordinal
        }
    }

    private fun getLayout(viewType: Message.Type): Int {
        return when (viewType) {
            Message.Type.RECIEVED -> R.layout.item_message_received
            Message.Type.SENT -> R.layout.item_message_sent
        }
    }

    fun addMessage(message: Message) {
        messageList.add(message)
    }

    fun addMessage(i: Int, message: Message) {
        messageList.add(i, message)
    }

    fun addAll(i: Int, messages: Collection<Message>): Boolean {
```

```
        return messageList.addAll(i, messages)
    }

    fun setMessage(i: Int, message: Message): Message {
        return messageList.set(i, message)
    }

    fun removeMessage(i: Int): Message {
        return messageList.removeAt(i)
    }

    fun removeAll() {
        messageList.clear()
    }

    fun indexOf(message: Message): Int {
        return messageList.indexOf(message)
    }

    fun lastIndexOf(message: Message): Int {
        return messageList.lastIndexOf(message)
    }

    open inner class MessageHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {

        val messageText =
itemView.findViewById<TextView>(R.id.textViewMessageText)
        val messageBody = itemView.findViewById<View>(R.id.linearLayoutMessage)

        open fun bind(message: Message) {
            messageBody.gone()
            if (messageText != null &&
                    message.type != Message.Type.RECIEVED &&
                    message.type != Message.Type.SENT
            ) {
                messageBody.visible()
                messageText.text = message.text
                messageText.visible()
            }
        }
    }

    open inner class FileMessageHolder(itemView: View) :
MessageHolder(itemView) {

        val thumbView =
itemView.findViewById<View>(R.id.frameLayoutAttachedImage)
        val attachedImage =
itemView.findViewById<ImageView>(R.id.imageViewAttachedImage)
        val layoutAttachedFile =
itemView.findViewById<View>(R.id.relativeLayoutAttachedFile)
        val fileImage = itemView.findViewById<ImageView>(R.id.imageViewFile)
```

```
        val fileName = itemView.findViewById<TextView>(R.id.textViewFileName)
        val fileSize = itemView.findViewById<TextView>(R.id.textViewFileSize)
        val progressFileUpload =
itemView.findViewById<ProgressBar>(R.id.progressBarFileUpload)
        val context = itemView.context

        override fun bind(message: Message) {
            super.bind(message)

            thumbView.gone()
            layoutAttachedFile?.gone()

            val attachment = message.attachment
            if (attachment == null) {
                if (thumbView != null && message.type == Message.Type.SENT) {
                    thumbView.gone()
                    messageBody.visible()
                    val textMessage =
context.resources.getString(R.string.chat_uploading_file) + message.text
                    messageText.text = textMessage
                    messageText.visible()
                }
            } else {
                messageBody.gone()
                val imageInfo = attachment.fileInfo.imageInfo
                if (imageInfo == null) {
                    showAttachmentView(attachment)
                } else {
                    addImageInView(attachment.fileInfo, attachedImage,
thumbView, messageText)
                }
            }
        }

        private fun showAttachmentView(attachment: Message.Attachment) {
            layoutAttachedFile.visibility = View.VISIBLE
            fileName.text = attachment.fileInfo.fileName
            fileSize.visibility = View.GONE
            when (attachment.state) {
                AttachmentState.READY -> {
                    fileImage.visibility = View.VISIBLE
                    if (attachment.fileInfo.imageInfo == null) {
                        val fileName = attachment.fileInfo.fileName
                        val downloadedFile = File(

Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
                                fileName
                        )
                        if (downloadedFile.exists()) {
                            progressFileUpload.progress = 100
                            fileImage.setOnClickListener {
                                openFile(downloadedFile)
                            }
```

```
                        } else {
                            fileImage.setOnClickListener {
                                downloadFile(attachment.fileInfo)
                            }
                        }
                    }
                    fileSize.visibility = View.VISIBLE
                    val size: String =
humanReadableByteCountBin(attachment.fileInfo.size).toString()
                    fileSize.text = size
                }
                AttachmentState.UPLOAD -> {
                    if (progressFileUpload.visibility == View.GONE) {
                        fileImage.visibility = View.INVISIBLE
                        progressFileUpload.visibility = View.VISIBLE
                    }
                }
                else -> {
                }
            }
        }

        private fun openFile(file: File) {
            val map = MimeTypeMap.getSingleton()
            val ext = MimeTypeMap.getFileExtensionFromUrl(file.name)
            var type = map.getMimeTypeFromExtension(ext)

            if (type == null) {
                type = "*/*"
            }

            val intent = Intent(Intent.ACTION_VIEW)
            val data = FileProvider.getUriForFile(context,
context.applicationContext.packageName + ".fileprovider", file)
            intent.flags = Intent.FLAG_GRANT_READ_URI_PERMISSION
            intent.setDataAndType(data, type)
            context.startActivity(intent)
        }

        private fun humanReadableByteCountBin(bytes: Long): String? {
            val b = if (bytes == Long.MIN_VALUE) Long.MAX_VALUE else abs(bytes)
            return when {
                b < 1024L -> "$bytes B"
                b <= 0xfffccccccccccccL shr 40 ->
                    String.format(Locale.getDefault(), "%.0f kB", bytes /
1024.0)
                b <= 0xfffccccccccccccL shr 30 ->
                    String.format(Locale.getDefault(), "%.0f MB", bytes /
1048576.0)
                b <= 0xfffccccccccccccL shr 20 ->
                    String.format(Locale.getDefault(), "%.0f GB", bytes /
1.073741824E9)
                b <= 0xfffccccccccccccL shr 10 ->
```

Продолжение ПРИЛОЖЕНИЯ А

```
                    String.format(Locale.getDefault(), "%.0f TB", bytes /
1.099511627776E12)
                b <= 0xfffccccccccccccL ->
                    String.format(Locale.getDefault(), "%.0f PiB", (bytes shr
10) / 1.099511627776E12)
                else -> String.format(Locale.getDefault(), "%.0f EiB", (bytes
shr 20) / 1.099511627776E12)
            }
        }

        private fun addImageInView(
            attachment: Message.FileInfo,
            imageView: ImageView,
            thumbView: View,
            textView: TextView
        ) {
            val fileUrl = Uri.parse(attachment.url)
            Glide.with(context)
                    .load(fileUrl)
                    .diskCacheStrategy(DiskCacheStrategy.RESOURCE)
                    .into(imageView)
            textView.text = context.resources.getString(R.string.chat_image)
            textView.visibility = View.VISIBLE
            thumbView.visibility = View.VISIBLE
        }
    }

    inner class SentMessageHolder(itemView: View) : FileMessageHolder(itemView)

    inner class ReceivedMessageHolder(itemView: View) :
FileMessageHolder(itemView) {
        val nameTextForFile =
itemView.findViewById<TextView>(R.id.textViewOperatorName)
        val profileImage =
itemView.findViewById<ImageView>(R.id.imageViewAvatar)
        override fun bind(message: Message) {
            super.bind(message)
            nameTextForFile.text = message.senderName
            nameTextForFile.visible()
            Glide.with(context)
                    .load(message.senderAvatarUrl)
                    .into(profileImage as ImageView)
            profileImage.visible()
        }
    }
}
```

Листинг файла **ChatProvider**

```
class ChatProvider @Inject constructor(private val apiService: MyApiService) {
```

Продолжение ПРИЛОЖЕНИЯ А

```
    fun getPublicKey(userId: String): Single<String> {
        return apiService.getPublicKey(userId)
    }

    fun getChatMessagesStream(userId: String): Flowable<List<Message>> {
        return apiService.getMessages(userId)
    }

    fun getMessages(userId: String, limit: Int): Single<List<Message>> {
        return apiService.getMessages(userId, limit)
    }

    fun sendMessage(userId: String, message: Message): Completable {
        return apiService.sendMessage(userId, message)
    }

    fun sendFile(userId: String, file: Message): Completable {
        return apiService.sendFile(userId, file)
    }
}
```

Листинг файла **ChatPresenter.kt**

```
class ChatPresenter @Inject constructor(
    private val chatProvider: ChatProvider
) : BasePresenter<ChatMvpView>() {

    private lateinit var userId: String
    private lateinit var rsaCipher: CryptoUtils

    private var disposableChatStream: Disposable? = null

    fun onCreateView(context: Context, userId: String) {
        mvpView.initViews()
        this.userId = userId

        chatProvider.getPublicKey(userId)
                .doOnSuccess { userKey ->
                    rsaCipher = CryptoUtils.getInstance(context).apply {
                        publicKey = userKey
                    }
                }.async()
    }

    fun getChatMessagesStream(userId: String) {
        this.disposableChatStream = chatProvider.getChatMessagesStream(userId)
                .map { messagesList -> messagesList.decrypt() }
                .async()
    }

    fun requestMessages(limit: Int) {
        chatProvider.getMessages(this.userId, limit)
                .map { messagesList -> messagesList.decrypt() }
                .async()
    }
```

```
    fun sendMessage(message: Message) {
        message.apply { data = rsaCipher.encrypt(data) }
        chatProvider.sendMessage(this.userId, message).async()
    }

    fun sendFile(file: Message, sendFileCallback: MessageStream.SendFileCallback) {
        file.apply { data = rsaCipher.encrypt(data) }
        chatProvider.sendFile(this.userId, file)
                .doOnSubscribe { sendFileCallback.onProgress(Message.Id) }
                .doOnComplete { sendFileCallback.onSuccess(Message.Id) }
                .doOnError { error -> sendFileCallback.onFailure(Message.Id, error) }
                .async()
    }

    fun destroySession() {
        disposableChatStream?.dispose()
    }

    fun onConnectionChanged(isConnected: Boolean) {
        if (isConnected) {
            mvpView.setStateData()
        } else {
            mvpView.setStateInternetError()
        }
    }

    private fun List<Message>.decrypt(): List<Message> {
        return this.map { message ->
            message.apply { data = rsaCipher.decrypt(data) }
        }
    }

    private fun <T> Flowable<T>.async(): Disposable = compose { flowable ->
        flowable
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
    }.subscribe()
}
```

Листинг файла **MyApiService.kt**

```
interface MyApiService {

    @GET("chat/{userID}/publicKey")
    fun getPublicKey(
        @Path("userID") userId: String
    ): Single<String>

    @GET("chat/{userID}/messages")
    fun getMessages(
        @Path("userID") userId: String
    ): Flowable<List<Message>>

    @GET("chat/{userID}/messages")
    fun getMessages(
        @Path("userID") userId: String,
        @Query("limit") limit: Int = 0
```

Окончание ПРИЛОЖЕНИЯ А

```
    ): Single<List<Message>>

    @POST("chat/{userID}/sendmessage")
    fun sendMessage(
        @Path("userID") userId: String,
        @Body message: Message
    ): Completable

    @POST("chat/{userID}/sendfile")
    fun sendFile(
        @Path("userID") userId: String,
        @Body file: Message
    ): Completable
}
```