МЕТОДЫ И СИСТЕМЫ ЗАЩИТЫ ИНФОРМАЦИИ, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ / METHODS AND SYSTEMS OF INFORMATION PROTECTION, INFORMATION SECURITY

DOI: https://doi.org/10.60797/IRJ.2024.145.174

ШИФРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ В ПРИЛОЖЕНИИ НА БАЗЕ ОС ANDROID

Научная статья

Протченко К.О.[1, *], Титов П.Л.[2]
[1] ORCID : 0009-0005-9900-761X;
[2] ORCID : 0000-0002-3408-5757;
[1] Scanny, Владивосток, Российская Федерация
[1, 2] Дальневосточный федеральный университет, Владивосток, Российская Федерация

* Корреспондирующий автор (kirill.protchenko[at]mail.ru)

Аннотация

В данной статье реализуется шифрование пользовательских данных в приложении мессенджера на базе операционной системы Android и интеграция этого приложения с облачным сервисом Firebase. В рамках работы была проведена работа с анализом криптографических алгоритмов, а также другими практиками защиты.

Результатом работы является разработанное приложение мессенджера для операционной системы Android и интеграцией его с облачными сервисами Firebase. Разработка велась с использованием языков программирования Java и Kotlin, в качестве основного криптографического алгоритма был взят алгоритм RSA.

Данная статья будет полезна как тем, кто изучает защиту данных в программных сервисах, так и тем, кто занимается разработкой приложений по Android и работой с Firebase.

Ключевые слова: криптографический алгоритм RSA, операционная система Android, Firebase, языки программирования Kotlin и Java.

ENCRYPTING USER DATA IN AN ANDROID APPLICATION

Research article

Protchenko K.O.[1, *], Titov P.L.[2]
[1] ORCID : 0009-0005-9900-761X;
[2] ORCID : 0000-0002-3408-5757;
[1] Scanny, Vladivostok, Russian Federation
[1, 2] Far Eastern Federal University, Vladivostok, Russian Federation

* Corresponding author (kirill.protchenko[at]mail.ru)

Abstract

This article implements encryption of user data in a messenger application based on Android operating system and integration of this application with Firebase cloud service. The work involved analysing cryptographic algorithms as well as other security practices.

The result of the work is a developed messenger application for Android operating system and its integration with Firebase cloud services. The development was performed using Java and Kotlin programming languages, RSA algorithm was used as the main cryptographic algorithm.

This article will be useful both for those who study data protection in software services and for those who develop Android applications and work with Firebase.

Keywords: RSA cryptographic algorithm, Android operating system, Firebase, Kotlin and Java programming languages.

Введение

В наше время не удивить мессенджерами и другими средствами обмена информацией, ведь это основной источник обмена информацией в сети Интернет. Этими средствами пользуются миллиарды людей с разных уголков планеты, и каждый имеет право на конфиденциальность личной переписки. Мало кому понравится, если его данные утекут в сети интернет. Особенно сейчас, когда много различных пользовательских данных становятся доступны широкой публике. К тому же компании несут ответственность за разглашение конфиденциальной информации пользователей. Данные факты предъявляют к средствам обмена информацией высочайшие требования к безопасности.

Огромное подавляющее население планеты имеют и ежедневно пользуются смартфонами под управлением различных операционных систем. По статистике за 2021 год, доля системы Android среди всех занимает долю чуть более 80% на рынке, а системе IOS отводится чуть более 18%.

Приложения под Android [1] пишутся на двух языках – это Java и Kotlin. Язык программирования Kotlin пришел на замену Java, он работает также на Java Virtual Machine (сокр. JVM), но отличается более активной поддержкой, удобством и мощью. Именно поэтому Java уже устаревает и практически не используется в новых проектах.

Что касается облачного хранилища, то Firebase, наиболее популярное решение для решения множества задач [2], начиная с хранения изображений и других файлов, заканчивая аналитикой и базами данных. Это связано с удобным интерфейсом взаимодействия с данным решением и отличная интеграция со множеством специализированных средств для разработки. В рамках данной работы будут использоваться хранилище файлов и база данных.

Отличительная особенность хранения данных в Firebase является то, что данные хранятся в нереляционной базе данных, тогда как для хранения большинства информации лучше всего подходят реляционные базы.

В данной работе за основу шифрования данных был взят асимметричный алгоритм RSA [3]. Но данный алгоритм будет несколько изменен для возможности использования с учетом выбранных технологий, но основные принципы изменениям не подвергались. На практике одним лишь криптографическим алгоритмом не обойтись, поэтому будут применяться и другие концепции, призванные защищать систему от взлома.

Обзор

Согласно исследованию 2019 года компании BI.ZONE [4] по разновидностям уязвимостей на мобильных приложениях, 28% приходится на хранение данных, 21% приходится на аутентификацию и управление сессиями, 16% на серверную часть, 11% на взаимодействие с платформой, 8% на сетевое взаимодействие и 16% на другое.

Таким образом основными слабыми местами на мобильных устройствах является хранение данных и токенов авторизации.

По умолчанию операционная система Android имеет базовые системы защиты хранилища данных, одни из них: шифрование всех данных в хранилище пока телефон находится в режиме сна, четкое ограничение доступа к локальным хранилищам приложений и другие.

В статье по основным способам шифрования на мобильных устройствах [5] приводятся следующие принципы касаемо защиты данных:

1. Шифрование диска для защиты данных на мобильном устройстве, но как уже говорилось выше современные операционные системы предлагают уже встроенную защиту хранилища.
2. Шифрование файлов с целью защиты отдельных файлов, что может быть полезно для защиты конфиденциальной информации, такой как документы.
3. Шифрование сообщений для защиты переписки между пользователями.
4. Шифрование VPN с целью защиты интернет-соединения.

Если говорить о применении алгоритмов шифрования на мобильных устройствах, то в статье [6] описан процесс разработки кроссплатформенного приложения для обмена данным с информационной системой.

Однако, автором была выбрана не лучшая технология для разработки мобильного приложения, а именно Xamarin от компании Microsoft. Как описывает и сам автор, недостатками данного решения является: недостаточная поддержка данного продукта со стороны компании, ограниченный выбор сторонних библиотек, ограниченная функциональность данного решения, что ставит под вопрос дальнейшее развитие и поддержку разрабатываемого мобильного приложения.

В качестве основного алгоритма шифрования используется Rijndael, данный алгоритм обладает следующими преимуществами: высокое быстродействие, гибкость модификации данного алгоритма, надежность и криптостойкость, однако без минусов не обойтись: сложность обратного дешифрования сообщений ввиду различных параметров и разным порядком применения функций, что сказывается на эффективности при реализации шифра.

Дополнительно хотелось бы отметить то, что автор не привел реализацию защиты данных в разрабатываемом решении.

Актуальность и постановка целей и задач

Защита информации издавна была есть и еще долгое время будет оставаться важной темой. В наше время издается много законов обязывающих компании соблюдать защиту пользовательских данных, постоянно улучшаются методы защиты. Данная тема представляет интерес как для государств, так и для компаний и пользователей, так как утечка важной и конфиденциальной информации будет иметь существенные последствия.

В данной статье будет представлено мобильное приложение с использованием актуальных подходов и популярных технологий, таких как: нативная разработка под Android с использованием широко используемого языка программирования Kotlin, использование облачных решений от Firebase и другие, так как это позволяет эффективно разрабатывать и впоследствии поддерживать программное обеспечение.

Особое внимание будет уделено защите данных, где в качестве основного алгоритма шифрования будет использоваться асимметричный алгоритм RSA преимуществами которого, является возможность модификации при необходимости, простота реализации, асимметричность, что позволяет использовать его по незащищенным каналам связи, из недостатков можно выделить скорость работы. Данный алгоритм широко используется для защиты программного обеспечения и в схемах цифровой подписи.

С учетом используемых технологий данный алгоритм будет модифицирован для возможности использования его с облачным хранилищем Firebase. А также будет представлена реализация на языке программирования Kotlin.

Дополнительно, что хочется отметить, данное приложение не будет хранить никакой чувствительной информации на устройстве, что снижает риск кражи конфиденциальных данных с устройства.

Основные результаты

Для начала необходимо организовать структуру хранения данных в облачном сервисе Firebase. Наиболее подходящим вариантом будет использование структуры, напоминающей реляционную БД.

Исходя из этого, данные будут поделены на 3 корневых файла:

1) Chats – в котором будет храниться информация об уникальном чате между пользователями;
2) Messages – в котором будут храниться все сообщения пользователей;
3) Users – в котором будут храниться данные для авторизации пользователей.

Далее необходимо определить, какого рода информацию необходимо хранить в облаке. Пойдем по порядку, файл Users будет содержать информацию о зарегистрированных пользователях. Описание полей для хранения информации в файле Users представлено в таблице 1.

Таблица 1 - Описание полей для файла Users

DOI: https://doi.org/10.60797/IRJ.2024.145.174.1

Название	Тип данных	Обозначение	Пример	Примечание
userId	Int	Уникальный идентификатор пользователя	14234	–
password	String	Пароль, которым пользователь будет защищать свой аккаунт	353285702362754 43623	Хранится в виде хэш-функции
username	String	Логин или имя пользователя, по которому он будет авторизовыватьс я и по которому другие пользователи смогут его найти	Иван	–

Касаемо пароля стоит уточнить, что пароль в чистом виде никогда не хранится. Вместо этого пароль проводят через хэш-функцию и получают некоторый хэш-код. Впоследствии при сопоставлении двух паролей сравнивают их хэш-функции. Делается это в целях безопасности.

Визуальная структура пакетов файла Users представлена на рисунке 1.

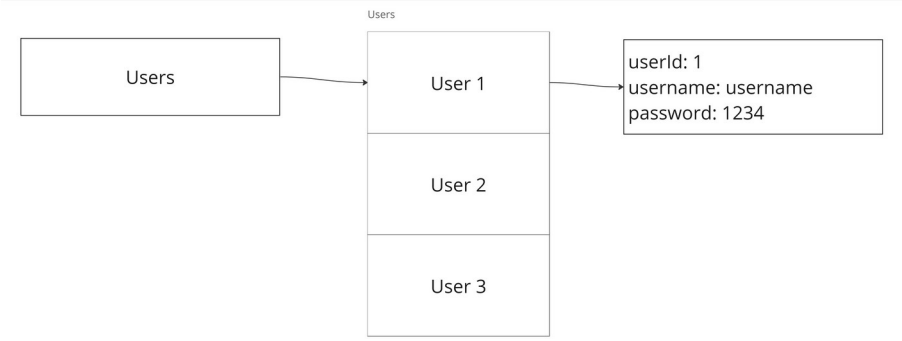

Рисунок 1 - Структура данных файла Users

DOI: https://doi.org/10.60797/IRJ.2024.145.174.2

Далее перейдем к файлу Chats, который будет содержать в себе информацию с кем определенный пользователь ведет общение.

Внутри файла Chats будут файлы с именами пользователей, внутри них будут содержаться файлы с именами пользователей, с которыми данный человек ведет переписку. Ниже, в таблице 2 описаны поля, которые будут содержаться в данном файле.

Таблица 2 - Описание полей для файла Chats

DOI: https://doi.org/10.60797/IRJ.2024.145.174.3

Название	Тип данных	Обозначение	Пример	Примечание
chatId	Int	Уникальный id чата, по которому можно будет найти переписку	14234	–
lastMessage	String	Последнее сообщение	3532857|0236275| 443623	Хранится в зашифрованном виде

lineKeys	String	Ключи, используемые для шифрования	353257236| 357925|239755	Хранится в зашифрованном виде
timeMessage	String	Время, в которое было отправлено сообщение	3462753| 32956236|312657	Хранится в зашифрованном виде
usernameToWhom	String	Имя пользователя, с которым ведется общение	Имя	–

Визуальная структура пакетов файла Chats представлена на рисунке 2.

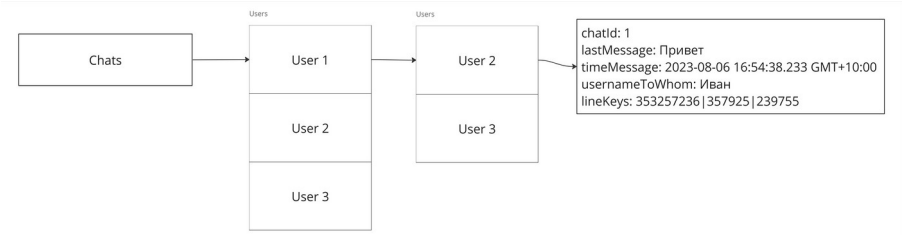

Рисунок 2 - Структура данных файла Chats
DOI: https://doi.org/10.60797/IRJ.2024.145.174.4

И наконец файл Messages, который будет содержать сообщения. Структура будет следующая: внутри файла Messages хранится список файлов, которые делятся по id чатов, внутри этих файлов хранится список со всеми сообщениями этого чата. Поля, которые необходимы для хранения сообщений, представлены в таблице 3.

Таблица 3 - Описание полей для файла Messages

DOI: https://doi.org/10.60797/IRJ.2024.145.174.5

Название	Тип данных	Обозначение	Пример	Примечание
timeMessage	String	Время, в которое было написано сообщение	5235223|2368352| 1485635	Хранится в зашифрованном виде
message	String	Сообщение	3532857|0236275| 443623	Хранится в зашифрованном виде
urlImage	String/null	Ссылка на изображение, если оно есть	353257236| 357925|239755	Хранится в зашифрованном виде
usernameFrom	String	Кто написал это сообщение	3462753| 32956236|312657	Хранится в зашифрованном виде

Визуальная структура пакетов файла Messages представлена на рисунке 3.

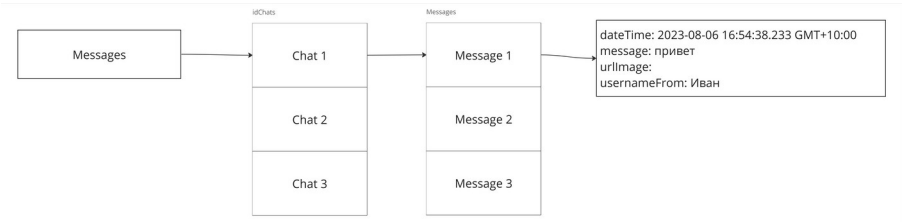

Рисунок 3 - Структура данных файла Messages
DOI: https://doi.org/10.60797/IRJ.2024.145.174.6

Теперь разберем, как работает сам алгоритм создания ключей на базе криптографического алгоритма RSA.

Большинство криптографических алгоритмов строятся по принципу «В одну сторону легко, в другую - невероятно сложно» [7], [8], [9]. Для примера возьмем выражение (1), где произведение двух чисел 21489237 и 903285327 образуют число 19410912470626499.

$$21489237 \times 903285327 = 19410912470626499 \quad (1)$$

Зная результат выражения 19410912470626499, нам будет очень сложно подобрать числа 21489237 и 903285327.

Теперь перейдем к математической стороне создания открытого и закрытого ключа. Алгоритм следующий:

1) выбираем 2 больших простых числа p и q;

2) вычисляем n как произведение p и q в соответствии с формулой (2);

$$n = p \times q \quad (2)$$

3) после, вычисляем функцию Эйлера φ(n) согласно формуле (3);

$$\varphi(n) = (p-1) \times (q-1) \quad (3)$$

4) затем выбираем произвольное простое число e в диапазоне: 0 < e < n. Пара чисел (e, n) будет открытым ключом;

5) вычисляем целое число d из соотношения по формуле (4). Пара чисел (d, n) будет закрытым ключом.

$$(d \times e) mod \varphi(n) = 1 \quad (4)$$

Шифрование сообщения происходит посимвольно по формуле (5). Для этого используем пару (e, n) открытого ключа:

$$C_i = T_i^e mod n \quad (5)$$

Чтобы расшифровать или восстановить сообщение, используем пару (d, n) закрытого ключа в соответствии с формулой (6):

$$T_i = C_i^d mod n \quad (6)$$

где: T – код символа; C – закодированный код символа.

Реализация криптографического алгоритма на языке программирования [5] Kotlin представлена на рисунках 4 – 5.

```kotlin
fun generateKeypair(): Pair<PublicKeys, PrivateKeys> {
    // Генерация публичного и приватного ключей
    val p = generateLargePrime()
    val q = generateLargePrime()
    val n = p * q
    val phi = (p - BigInteger.ONE) * (q - BigInteger.ONE)
    var e: BigInteger
    while (true) {
        e = BigInteger(phi.bitLength(), Random())
        if (e > BigInteger.valueOf( val: 2) && e < phi && gcd(e, phi) == BigInteger.ONE) {
            break
        }
    }
    val (_, d, _) = extendedGcd(e, phi)
    val privateKey = (d % phi + phi) % phi
    return Pair(PublicKeys(e, n), PrivateKeys(privateKey, n))
}
```

Рисунок 4 - Метод для формирования публичного и приватного ключа

DOI: https://doi.org/10.60797/IRJ.2024.145.174.7

```kotlin
private fun generateLargePrime(): BigInteger {
    // Генерация большого простого числа
    while (true) {
        val p = BigInteger( numBits: 512, Random())
        if (p.isProbablePrime( certainty: 10)) {
            return p
        }
    }
}

kirlozavr
private fun gcd(a: BigInteger, b: BigInteger): BigInteger {
    // Нахождение наибольшего общего делителя
    return if (b == BigInteger.ZERO) a else gcd(b,  b: a % b)
}

kirlozavr
private fun extendedGcd(
    a: BigInteger,
    b: BigInteger
): Triple<BigInteger, BigInteger, BigInteger> {
    // Расширенный алгоритм Евклида
    return if (a == BigInteger.ZERO) Triple(b, BigInteger.ZERO, BigInteger.ONE)
    else {
        val (gcd, x1, y1) = extendedGcd( a: b % a, a)
        val x = y1 - (b / a) * x1
        val y = x1
        Triple(gcd, x, y)
    }
}
```

Рисунок 5 - Вспомогательные методы для формирования публичных и приватных ключей

DOI: https://doi.org/10.60797/IRJ.2024.145.174.8

Теперь перейдем к формированию ключей шифрования для пароля. В основе будет лежать описанный выше криптографический алгоритм RSA. Как мы помним, для начала формируются 2 больших простых числа p и q, а после формируем число e.

Для определения seed-значения, от которого мы будем отталкиваться при формировании большого простого числа, обычно используется генератор случайных чисел. Но в данном случае, чтобы для одинакового набора информации всегда получать одни и те же ключи, а впоследствии и хэш-функцию, seed-значение мы будем формировать на основе введенного набора информации.

Алгоритм формирования семени для нахождения большого простого числа p:

1) получаем id и username пользователя;
2) умножаем строку username саму на себя 8 раз и прибавляем к получившейся строке id;
3) умножаем строку «FNJ#J!N2*SFN#N» саму на себя 6 раз;
4) находим сумму кодов всех символов в обеих строках;
5) затем преобразуем данное число в BigInteger с количеством бит равным 512;
6) проверяем, является ли данное число простым, если да, то число p найдено, в противном случае прибавляем 1 и повторяем условие в шаге №6.

Реализация нахождения большого простого числа p на языке Kotlin представлена на рисунке 6.

```
var count = 0L

var p: BigInteger
while (true) {
    val preP = getNumberFromTwoValues(username.repeat( n: 8) + id, "FNJ#J!N2*SFN#N".repeat( n: 6)) + count

    p = BigInteger.valueOf(preP)
        .add(BigInteger.valueOf( val: 352351))
        .setBit(512)
    count += 1
    if (p.isProbablePrime( certainty: 10)) {
        break
    }
}
```

Рисунок 6 - Нахождение большого простого числа p
DOI: https://doi.org/10.60797/IRJ.2024.145.174.9

Алгоритм формирования семени для нахождения большого простого числа q:
1) получаем id и username пользователя;
2) умножаем строку username саму на себя 24 раза и прибавляем к получившейся строке id;
3) умножаем строку «=BSFUIA(#)$)(%@#DJNJ» саму на себя 7 раз;
4) находим сумму кодов всех символов в обеих строках;
5) затем преобразуем данное число в BigInteger с количеством бит равным 512;
6) проверяем, является ли данное число простым, если да, то число q найдено, в противном случае прибавляем 1 и повторяем шаг №6.

Реализация нахождения большого простого числа q на языке Kotlin представлена на рисунке 7.

```
count = 0L
var q: BigInteger
while (true) {
    val preQ = getNumberFromTwoValues(username.repeat( n: 24) + id, "=BSFUIA(#)$)(%@#DJNJ".repeat( n: 7)) + count

    q = BigInteger.valueOf(preQ)
        .add(BigInteger.valueOf( val: 892487))
        .setBit(512)
    count += 1
    if (q.isProbablePrime( certainty: 10)) {
        break
    }
}
```

Рисунок 7 - Нахождение большого простого числа q
DOI: https://doi.org/10.60797/IRJ.2024.145.174.10

Алгоритм формирования семени для нахождения большого простого числа e:
1) получаем id и username пользователя;
2) умножаем строку username саму на себя 36 раз, прибавляем к получившейся строке id и массив байт значения φ(n);
3) умножаем строку «UH6WFUNX_$@XRMJ#@» саму на себя 3 раза;
4) находим сумму кодов всех символов в обеих строках;
5) затем преобразуем данное число в BigInteger с количеством бит равным 512;
6) проверяем, является ли данное число простым и попадает в диапазон 0 < e < n, если да, то число e найдено, в противном случае прибавляем 1 и повторяем условие в шаге №6.

Реализация нахождения большого простого числа e на языке Kotlin представлена на рисунке 8.

```
val n = p * q
val phi = (p - BigInteger.ONE) * (q - BigInteger.ONE)

count = 0
var e: BigInteger
while (true) {
    val preE = getNumberFromTwoValues(username.repeat( n: 36) + id + phi.toByteArray().contentToString(), "UH6WFUNX_$@XRMJ#@".repeat( n: 3)) + count

    e = BigInteger.valueOf(preE)
        .add(BigInteger.valueOf( val: 322761))
        .setBit(512)
    count += 1
    if (e > BigInteger.valueOf( val: 2) && e < phi && gcd(e, phi) == BigInteger.ONE) {
        break
    }
}
```

Рисунок 8 - Нахождение большого простого числа e
DOI: https://doi.org/10.60797/IRJ.2024.145.174.11

После того как получились открытые и закрытые ключи для работы с паролем, можно производить операцию шифрования. Кодирование происходит посимвольно по формуле (5) и для получения итогового хэш-кода мы складываем в строку результат шифрования каждого символа, пример можно увидеть в выражении (7).

$$111 + 222 + 333 = 111222333 \quad (7)$$

Как уже говорилось выше, хранить мы будем только хэш-функцию и все дальнейшие операции будем производить только с ней. Сам пароль и ключи шифрования для пароля никуда не передается.

Далее переходим к визуальному интерфейсу. Для начала необходим экран авторизации и регистрации, а также возможность запомнить данные для входа. Экран авторизации и регистрации представлен на рисунке 9.

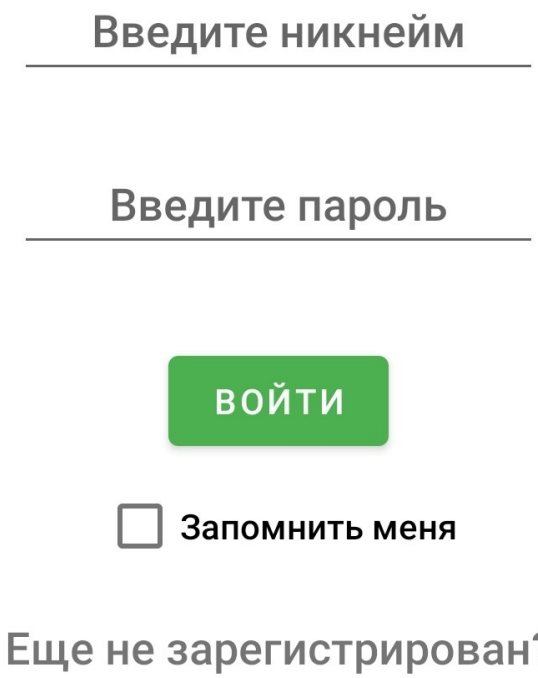

Рисунок 9 - Экран авторизации и регистрации
DOI: https://doi.org/10.60797/IRJ.2024.145.174.12

В зависимости от намерений пользователя после ввода данных они либо записываются в базу данных Firebase при регистрации, либо проверяется наличие такого пользователя в базе данных, и в соответствии с этим либо разрешается, либо запрещается доступ к аккаунту. Если пользователь неправильно ввел данные, ему выводится сообщение, что

неправильно введен логин или пароль. Сделано это для безопасности, чтобы злоумышленник не мог понять, какое из полей в форме было заполнено неправильно.

Следующим экраном будет отображение списка чатов, где будет видна вся переписка с другими пользователями. В элементах списка отображается информация о логине пользователя, с которым у нас ведется общение, последнее написанное сообщение и время отправки этого сообщения. Также в шапке присутствует приветствие пользователя и 2 кнопки: кнопка слева для поиска пользователей и кнопка справа для выхода из аккаунта. Внешний вид интерфейса данного экрана можно наблюдать на рисунке 10.

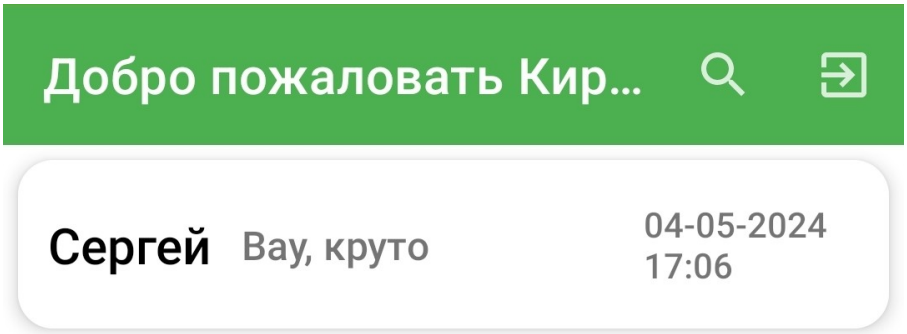

Рисунок 10 - Основной экран со списком чатов
DOI: https://doi.org/10.60797/IRJ.2024.145.174.13

При клике на элемент списка происходит переход на экран общения с данным пользователем. На данном экране есть возможность просматривать всю историю сообщений, писать и отправлять сообщение своему собеседнику, а также возможность отправлять изображения. Внешний вид экрана для общения пользователей представлен на рисунке 11.

Рисунок 11 - Экран для общения двух пользователей
DOI: https://doi.org/10.60797/IRJ.2024.145.174.14

И последний экран, который упоминался ранее, — это поиск других пользователей. Задумка проста: если пользователь хочет начать с кем-то общение, он его находит через поиск, а дальше начинает общение. При клике на

элемент списка открывается экран чата с этим пользователем. Интерфейс поиска среди пользователей представлен на рисунке 12.

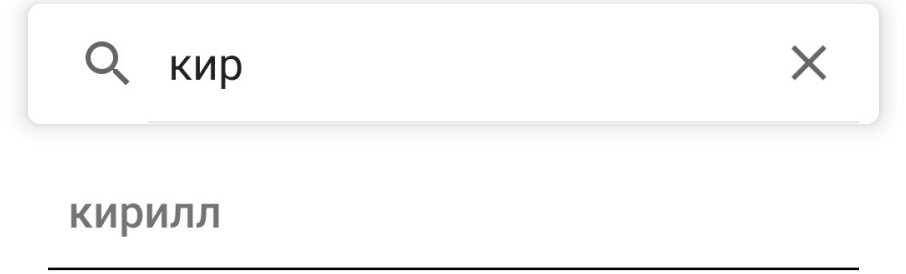

Рисунок 12 - Экран поиска других пользователей

DOI: https://doi.org/10.60797/IRJ.2024.145.174.15

Для более детального ознакомления исходный код, используемый в работе, приведен по ссылке [10].

Заключение

При разработке приложения использовались современные технологии, которые являются на данный момент актуальны при создании мобильных приложений, что дает возможность использовать приведенные результаты на практике в рабочих проектах и поддерживать их в дальнейшем.

Помимо используемых технологий была представлена реализация асимметричного алгоритма шифрования RSA и последующая его модификация с целью использования. Что в силу ограниченности информации по прикладному применению алгоритмов шифрования при разработке программного обеспечения будет полезным руководством.

Предложенные методы хранения и шифрования данных могут быть полезны разработчикам приложений, которые заботятся о безопасности пользовательской информации.

В целом, данная статья предлагает руководство и примеры практической реализации для специалистов, занимающихся разработкой приложений на Android и интересующихся аспектами криптографии и безопасности данных в облачной среде Firebase.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы / References

1. Android Documentation. — 2024 — URL: https://developer.android.com/docs (accessed: 10.05.2024)
2. Firebase Documentation. — 2024 — URL: https://firebase.google.com/docs (accessed: 10.05.2024)
3. Чичикин Г.Я. Криптосистема RSA / Г.Я. Чичикин, Д.А. Семенов // Наука, образование и культура; — Иваново: Олимп, 2019.
4. Демичев М.С. Обзор основных уязвимостей мобильных приложений / М.С. Демичев, Р.Ф. Файзулин // Актуальные проблемы авиации и космонавтики; — Красноярск: Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, 2021.
5. Смирнов В.М. Основные способы шифрования информации в мобильных устройствах / В.М. Смирнов, Е.В. Гусеева // Московский университет МВД России имени В.Я. Кикотя. — 2023. — 1. — с. 210-212.
6. Газизов А.Р. Алгоритм применения кроссплатформенного мобильного приложения для защиты информации в облачном хранилище / А.Р. Газизов // Вестник ДГТУ. Технические науки. — 2022. — 49.
7. Ахметов Б.С. Прикладная криптология: методы шифрования: учебное пособие / Б.С. Ахметов, А.Г. Корченко, В.П. Сиденко. — Алматы: КазНИТУ имени К.И. Сатпаева, 2015. — 496 с.
8. Smart N. Cryptography: An Introduction / N. Smart — Москва: ТЕХНОСФЕРА, 2005. — 529 с.
9. Москвин А.Д. Анализ современных алгоритмов шифрования данных / А.Д. Москвин, Л.Э. Петросян // Инженерный вестник Дона. — 2023. — 4.
10. GitHub. Source code of the developed system. — 2023 — URL: https://github.com/kirlozavr/SendMessages (accessed: 10.05.2024)

Список литературы на английском языке / References in English

1. Android Documentation. — 2024 — URL: https://developer.android.com/docs (accessed: 10.05.2024)
2. Firebase Documentation. — 2024 — URL: https://firebase.google.com/docs (accessed: 10.05.2024)
3. Chichikin G.Ja. Kriptosistema RSA [RSA Cryptosystem] / G.Ja. Chichikin, D.A. Semenov // Science, Education, and Culture; — Ivanovo: Olimp, 2019. [in Russian]
4. Demichev M.S. Obzor osnovnyh ujazvimostej mobil'nyh prilozhenij [Review of the main vulnerabilities of mobile applications] / M.S. Demichev, R.F. Fajzulin // Current problems of aviation and astronautics; — Krasnojarsk: Siberian State University of Science and Technology named after Academician M.F. Reshetnev, 2021. [in Russian]
5. Smirnov V.M. Osnovnye sposoby shifrovanija informatsii v mobil'nyh ustrojstvah [The main ways to encrypt information in mobile devices] / V.M. Smirnov, E.V. Guseeva // Moscow University of the Ministry of Internal affairs of Russia named after V.Ya. Kikot'. — 2023. — 1. — p. 210-212. [in Russian]
6. Gazizov A.R. Algoritm primenenija krossplatformennogo mobil'nogo prilozhenija dlja zaschity informatsii v oblachnom hranilische [Algorithm for using a cross-platform mobile application to protect information in cloud storage] / A.R. Gazizov // Bulletin of DSTU. Technical science. — 2022. — 49. [in Russian]
7. Ahmetov B.S. Prikladnaja kriptologija: metody shifrovanija: uchebnoe posobie [Applied Cryptography: Encryption Methods: a Study Guide] / B.S. Ahmetov, A.G. Korchenko, V.P. Sidenko. — Almaty: Kazntu named after K.I. Satpayev, 2015. — 496 p. [in Russian]
8. Smart N. Cryptography: An Introduction [Cryptography: An Introduction] / N. Smart — Moskva: TEHNOSFERA, 2005. — 529 p. [in Russian]
9. Moskvin A.D. Analiz sovremennyh algoritmov shifrovanija dannyh [Analysis of Modern Data Encryption Algorithms] / A.D. Moskvin, L.E. Petrosjan // Engineering Herald Don. — 2023. — 4. [in Russian]

10. GitHub. Source code of the developed system. — 2023 — URL: https://github.com/kirlozavr/SendMessages (accessed: 10.05.2024)