Санкт-Петербургский государственный университет

Программная инженерия

Чернявский Олег Николаевич

Исследование шифрования данных приложения Wickr для различных платформ

Выпускная квалификационная работа

Научный руководитель:
доцент кафедры СП СПбГУ, к. т. н. Литвинов Ю. В.

Консультант:
архитектор ПО, ООО “Белкасофт” Тимофеев Н. М.

Рецензент:
ген.дир., ООО ”Кода-Технологии” Ханов А. Р.

Санкт-Петербург
2021

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering

Oleg Cherniavskii

The research of data encryption used by Wickr application on different platforms

Bachelor's Thesis

Scientific supervisor:
Docent, C.Sc. Yurii Litvinov

Consultant:
Software Architect, Belkasoft LLC Nikita Timofeev

Reviewer:
CEO, CODA Technology LLC Artur Khanov

Saint-Petersburg
2021

Оглавление

Введение

Широкое распространение смартфонов и повсеместный доступ к высокоскоростному мобильному Интернету привели к тому, что приложения-мессенджеры стали значимым способом общения между людьми по всему миру. Это касается как личных переписок, так и корпоративной связи.

Доверяя мобильным приложениям гигабайты конфиденциальной информации, пользователи привлекают внимание разного рода злоумышленников, крупных корпораций и государства, озабоченного проблемами национальной безопасности и сохранения порядка. Именно эта причина привела ко все возрастающей популярности мессенджеров с поддержкой шифрования данных.

Однако приложения с высокой степенью безопасности передаваемых и хранимых данных могут быть интересны не только пользователям, обеспокоенным приватностью конфиденциальной информации, но и преступникам, пытающимся скрыть следы своей незаконной деятельности от правоохранительных органов и нуждающихся в средствах коммуникации. Мессенджеры с поддержкой шифрования данных регулярно используются в планировании и координации правонарушений, связанных с торговлей наркотиками, насилием над детьми и терроризмом [16][18][8]. Все это способствовало возникновению в среде криминалистов существенного спроса на программные решения для извлечения информации из подобных приложений.

Тем не менее, существуют мессенджеры, использующие нетипичные механизмы обеспечения безопасности данных пользователей и тем самым ставящие в тупик специалистов по цифровой криминалистике. Одним из таких приложений является Wickr Me, позиционирующийся его создателями как платформа-лидер по уровню безопасности и приватности данных [19]. Wickr Me поддерживает State-of-the-Art алгоритмы сквозного шифрования передаваемых по сети сообщений, возможность автоматического их удаления по истечению определенного срока (удаление не только логическое, но и физическое, с перезаписыванием обла-

сти памяти), полное шифрование данных приложения на самом устройстве пользователя и другие механизмы защиты информации [20].

На данный момент существуют программные решения, позволяющие извлекать данные из версий Wickr Me для платформ Android и iOS от таких компаний как Celebrite [5], MSAB [7], Oxygen Forensics [13] и Magnet Forensics [11]. Тем не менее, все они являются проприетарными, а принцип их работы остается неизвестен, так как исчерпывающие открытые исследования на эту тему отсутствуют. Именно этот пробел и призвано заполнить данное исследование — изучить механизм шифрования данных в приложении Wickr Me, описать его и предложить способ расшифровки хранимых данных. Данная работа будет выполнена в сотрудничестве с компанией Белкасофт, а ее результаты будут интегрированы в продукт компании Belkasoft X.

1. Постановка задачи

Цель данной работы — исследовать алгоритмы шифрования хранимых приложением Wickr Me данных (версии для iOS и Android) и реализовать механизм их расшифровки, используя методы реверс-инжиниринга.

Для достижения данной цели были поставлены следующие задачи:

- описать структуру файлов, хранимых приложением;
- определить используемые приложением алгоритмы шифрования и их параметры;
- описать способ извлечения ключей шифрования данных Wickr Me;
- описать механизм расшифровки данных приложения;
- интегрировать механизм расшифровки данных в Belkasoft X.

2. Обзор существующих решений

За последние годы сразу несколько компаний, специализирующихся на создании программного обеспечения в сфере цифровой криминалистики, представили решения для извлечения данных, хранимых приложением Wickr Me.

Одним из таких решений является Magnet AXIOM от компании Magnet Forensics [11]. Оно способно извлекать данные из версии приложения для устройств с операционной системой iOS, при этом ему не требуется имя пользователя и его пароль от аккаунта Wickr. Решение извлекает необходимые ключи шифрования из iOS Keychain [3] (централизованное защищенное хранилище для конфиденциальной информации), расшифровывает нужную информацию и показывает найденные сообщения, медиафайлы и геометки пользователю. Аналогичным образом данные из iOS версии-приложения способен извлекать продукт Cellebrite Responder от компании Cellebrite [5].

Целый ряд программных продуктов предоставляет возможность извлечения данных Wickr Me из полного образа файловой системы Android. Такими решениями являются Oxygen Forensic Detective от Oxygen Forensics [13], XRY от MSAB [7] и уже упомянутый Cellebrite Responder [5]. Oxygen Forensic Detective в дополнение к этому способен извлекать данные из облака Wickr посредством специального токена, который может быть извлечен из приложения на платформе Android [13].

Важно отметить, что все описанные выше продукты способны предоставлять доступ к сообщениям со сроком жизни не более 6 дней. Это связано с тем, что приложение не позволяет пользователям задать срок хранения данных больше указанного времени. После этого вся информация удаляется, а занимаемая ею область памяти перезаписывается, что делает извлечение старых сообщений невозможным даже теоретически [4].

3. Версия Wickr Me для iOS

В ходе исследования iOS-версии Wickr Me были извлечены хранимые приложением данные и исследована их структура. В процессе был использован инструмент Belkasoft X, предоставленный компанией Белкасофт, с помощью которого был снят образ мобильного устройства iPhone 5S с установленным на нем Wickr Me.

3.1. Структура хранимых приложением файлов

В данных приложения, хранимых в директории “private/var/mobile/Containers/Shared/AppGroup” был найден файл базы данных с именем “wickrLocal.sqlite”, оказавшийся не зашифрованным. Внутри него была обнаружена информация о пользователе, его собеседниках и их переписке. Особенный же интерес вызвала таблица ZWICKR_MESSAGE, содержащая все данные о входящих и исходящих сообщениях. Часть информации (например, время отправления сообщения и флаг о том, является ли сообщение входящим или исходящим) хранится в незашифрованном виде. В то же время текст сообщения и время его жизни зашифрованы и представлены в двоичном виде в колонках ZBODY и ZLIFESPAN.

Однако в ZWICKR_MESSAGE нет никакой информации о приложенных к сообщению файлах. Она хранится в таблице ZWICKR_FILE, где можно найти идентификаторы, по совместительству являющеся названиями файлов в файловой системе (колонка ZGUID, не зашифрована) и предположительно имена, которые они имели при отправке (колонка ZTITLE, зашифрована). Связь между сообщениями и приложенными к ним файлами хранится в таблице Z_11MSG, где ключу каждого файла из таблицы ZWICKR_FILE ставится в соответствие ключ сообщения из таблицы ZWICKR_MESSAGE, к которому этот файл относится.

Сами же файлы удалось найти среди данных приложения в директории “private/var/mobile/Containers/Data/Application”, а именно в поддиректории “Documents/downloads”. Их названием является иден-

тификатор из таблицы ZWICKR_FILE (см. рис. 1), у них отсутствует расширение и какие-либо заголовки, из чего был сделан вывод, что они зашифрованы.

Z_PK	Z_ENT	Z_OPT	ZSTATUS	ZGUID	ZMIMETYPE	ZTITLE
Фил...	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
2	11	1	0	16029147-C480-4657-9AA5-56417926E824	*NULL*	*BLOB*
3	11	1	0	85864A97-05CD-4F83-9DB0-8CCE39DBFCD3	*NULL*	*BLOB*
4	11	1	0	7C6CC1A4-2DF4-44CE-93EC-047C43CE068B	*NULL*	*BLOB*

Name	Date modified	Type	Size
7C6CC1A4-2DF4-44CE-93EC-047C43CE0...		File	4 499 KB
85864A97-05CD-4F83-9DB0-8CCE39DBF...		File	2 390 KB
16029147-C480-4657-9AA5-56417926E8...		File	2 390 KB

Рис. 1: Содержимое таблицы ZWICKR_FILE (сверху) и папки Documents/downloads (снизу)

3.2. Исполняемый файл и используемые библиотеки

Так как основная масса информации, представляющей интерес для криминалистов, оказалась зашифрована, было принято решение декомпилировать исполняемый файл приложения и некоторые из используемых им библиотек с целью выяснения алгоритма шифрования этих данных.

Исполняемые файлы iOS-приложений по умолчанию зашифрованы, поэтому был использован инструмент для специалистов по безопасности и реверс-инженеров под названием Frida [15]. С его помощью удалось извлечь из оперативной памяти устройства расшфрованный исполняемый файл WickrEnterprise и декомпилировать его, используя интерактивный дизассемблер IDA Pro [17].

Главным результатом декомпиляции стал псевдокод используемых приложением функций, их названия и используемые строковые константы. Также крайне полезной оказалась возможность перехода от адреса функции или константы к месту ее вызова или использования.

По итогам первичного исследования WickrEnterprise были определены представляющие наибольший интерес используемые приложением библиотеки:

- wickr_crypto_objc — содержит всю логику приложения, отвечающую за шифрование и расшифровку данных;
- wickr-crypto-c — библиотека с открытым исходным кодом [21], которая содержит базовые фунции шифрования, используемые wickr_crypto_objc;
- WickrMessagingProtocol — содержит описание формата protobuf-сообщений, используемых для общения между пользователями.

3.3. Извлечение ключа шифрования без пароля пользователя

В качестве отправной точки для определения механизма расшифровки данных приложения были выбраны методы attemptLoginWithPasswordCache и attemptLoginWithPassword класса AuthHelper из WickrEnterprise в силу их говорящего названия.

По цепочке вызовов в attemptLoginWithPasswordCache удалось обнаружить метод decodePasswordCache:devInfo класса WickrCryptoContext (находится в wickr_crypto_objc), в котором описан процесс расшифровки некоторого значения (листинг 1), необходимого для инициализации экземпляра класса WickrStorageKeys. Было сделано предположение, что это значение является ключом шифрования содержимого базы данных приложения.

Процесс расшифровки значения происходит в два этапа, на каждом из них вызывается метод decodeCipherTextGCM:WithKey класса AESHelper, принимающий два параметра — зашифрованное значение и ключ шифрования.

```
v8 = objc_msgSend(v5, "encLocalData");
v9 = objc_retainAutoreleasedReturnValue(v8);
v10 = objc_msgSend(v4, "systemSaltKey");
v11 = objc_retainAutoreleasedReturnValue(v10);
objc_release(v7);
v12 = (AESHelper_meta *, SEL, id, id))objc_msgSend)(
     (AESHelper_meta *)&OBJC_CLASS___AESHelper,
     "decodeCipherTextGCM:withKey:",
     v9,
     v11);
v13 = objc_retainAutoreleasedReturnValue(v12);
objc_release(v11);
objc_release(v9);
if ( v13 )
{
    v14 = objc_msgSend(v5, "cacheKey");
    v15 = objc_retainAutoreleasedReturnValue(v14);
    v16 = v15;
    v17 = (AESHelper_meta *, SEL, id, id))objc_msgSend)(
           (AESHelper_meta *)&OBJC_CLASS___AESHelper,
           "decodeCipherTextGCM:withKey:",
           v13,
           v15);
    v18 = (void *)objc_retainAutoreleasedReturnValue(v17);
    objc_release(v13);
    objc_release(v16);
}
```

Листинг 1: Фрагмент метода decodePasswordCache:devInfo, в котором видны два этапа расшифровки

Первый этап расшифровки. На первом этапе в качестве зашифрованных данных используется поле encLocalData класса WickrPasswordCache. Инициализацию этого поля удалось обнаружить в методе initWithEncData:andKey:, где ему присваивается значение (передаваемое из метода cachedStorageKeysInContext) из колонки zph таблицы ZSECEX_ACCOUNT базы данных приложения (листинг 2).

В роли ключа используется значение поля systemSaltKey класса

WickrDevInfo, которое получается из поля systemSalt того же класса путем добавления в начало нулевого байта.

```
v11 = (void *)objc_alloc(&OBJC_CLASS___WickrPasswordCache);
v12 = objc_msgSend(v3, "currUser");
v13 = (void *)objc_retainAutoreleasedReturnValue(v12);
v14 = v13;
v15 = objc_msgSend(v13, "ph");
v16 = objc_retainAutoreleasedReturnValue(v15);
v9 = objc_msgSend(v11, "initWithEncData:andKey:", v16, v6);
```

Листинг 2: Фрагмент метода cachedStorageKeysInContext, в котором значение zph (здесь ph) передается в initWithEncData:andKey:

Поле systemSalt в свою очередь инициализируется в функции wickr_dev_info_derive (библиотека wickr-crypto-c), куда передается поле uniqueDeviceSeed класса WCCoreRuntime, которое является конкатенацией значений fetchedDeviceID и bundleIdentifier (листинг 3). При этом:

- bundleIdentifier — это строка, уникальная для каждого iOS-приложения [6]. Для Wickr Me — это “com.mywickr.wickr”;
- fetchedDeviceID — это извлеченное из iOS Keychain значение, получаемое по ключу “!devid!”, от которого отсекли первые 32 байта;

```
v19 = objc_msgSend(v17, "bundleIdentifier");
v20 = objc_retainAutoreleasedReturnValue(v19);
objc_release(v18);

v21 = (void *)objc_alloc(&OBJC_CLASS___NSString);
v22 = (objc_msgSend)(v2, "fetchedDeviceID");
v23 = objc_retainAutoreleasedReturnValue(v22);
v24 = objc_msgSend(v21, "initWithData:encoding:", v23, 4LL);
objc_release(v23);

v25 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:",
    CFSTR("%@%@"), v24, v20);
```

Листинг 3: Фрагмент метода инициализации uniqueDeviceSeed, где происходит конкатенация bundleIdentifier и fetchedDeviceID

Наконец, в функции wickr_dev_info_derive по переданному значению uniqueDeviceSeed считается хэш SHA256 (листинг 4) и затем записывается в поле systemSalt структуры wickr_dev_info_t (по которой позже и инициализируется соответствующий экземпляр класса WickrDevInfo).

```
if (!crypto || !dev_salt || !system_id) {
    return NULL;
}

/* Convert the system_id into a salt by taking its hash, this
value remains private to the client */
wickr_buffer_t *system_salt =
    wickr_crypto_engine_digest(system_id, NULL, DIGEST_SHA_256);

if (!system_salt) {
    return NULL;
}
```

Листинг 4: Фрагмент исходного кода функции wickr_dev_info_t, в котором инициализируется значение systemSalt (system_id здесь — uniqueDeviceSeed)

Второй этап расшифровки. На втором этапе в качестве зашифрованных данных используется значение, полученное на первом этапе. В роли же ключа выступает значение поля cacheKey уже упомянутого класса WickrPasswordCache. В методе recordPasswordCache:inContext класса WCPasswordCacheHelper оно инициализируется как значение из iOS Keychain, получаемое по ключу “activeAccount”.

Алгоритм расшифровки. В метод decodeCipherText:withKey: класса AESHelper передаются два массива байт, соответствующих ключу шифрования и зашифрованным данным. Далее эти массивы передаются соответственно в функции wickr_cipher_key_from_buffer и wickr_cipher_result_from_buffer библиотеки wickr-crypto-c, где:

1. По первому байту аргументов decodeCipherText:withKey: опреде-

ляется тип шифрования (0 — AES-256 в режиме GCM[1], 1 — AES-256 в режиме CTR[2]). Далее будет приведено описание для случая AES-256 [2] в режиме GCM, так как именно он используется на обоих этапах расшифровки.

2. Ключ шифрования извлекается в функции wickr_cipher_key_from_buffer путем отсечения первого элемента соответствующего массива байт.

3. Остальные параметры, необходимые для расшифровки, извлекаются в функции wickr_cipher_result_from_buffer, куда передается массив, соответствующий зашифрованному значению. Байты со второго 2 по 13 трактуются как вектор инициализации, с 14 по 29 — как тег аутентификации, а остальные — как шифротекст.

4. Наконец, используя извлеченные параметры шифрования, происходит расшифровка данных.

Описанным выше образом происходит расшифровка значений и на первом, и на втором этапе. В итоге удалось получить набор байт, представляющий собой protobuf-сообщение, содержащее два значения (рис. 2). Описание формата этого сообщения удалось обнаружить в библиотеке wickr-crypto-c в файле “storage.proto” (листинг 5).

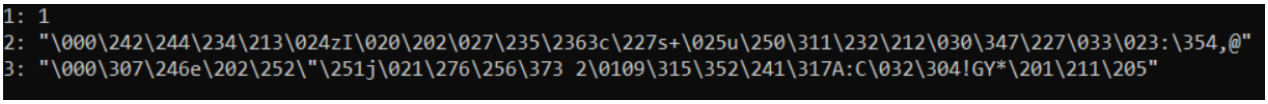

Рис. 2: Полученное в результате расшифровки protobuf—сообщение

```
message storage_keys {
  required uint32 version = 1;
  optional bytes local_storage = 2;
  optional bytes remote_storage = 3;
}
```

Листинг 5: Описание формата полученного protobuf-сообщения

[1]аббр. от англ. “Galois/Counter Mode” — счётчик с аутентификацией Галуа, режим аутентифицированного шифрования.

[2]сокр. от “Counter”, режим блочного шифрования с помощью счетчика.

3.4. Формирование ключа шифрования по паролю пользователя

В ходе исследования метода attemptLoginWithPassword класса AuthHelper удалось обнаружить альтернативный способ формирования ключа шифрования данных пользователя. В этом методе из базы данных извлекается значение zpt (таблица ZSECEX_ACCOUNT) и передается в качестве аргумента в метод initWithData:passphrase: класса WickrStorageKeys (находится в библиотеке wickr_crypto_objc). В свою очередь, в initWithData:passphrase: ключи шифрования получаются с помощью функции wickr_ctx_import_storage_keys библиотеки wickr-crypto-c, куда передается уже упомянутое значение zpt и пароль пользователя.

В wickr_ctx_import_storage_keys вызывается функция wickr_crypto_engine_kdf_decipher, где и содержится механизм расшифровки переданного значения с помощью пароля пользователя. Он состоит из трех этапов:

1. Определение параметров функции формирования ключа.
2. Формирование промежуточного ключа шифрования с помощью пароля пользователя.
3. Окончательная расшифровка ключей шифрования данных приложения.

Определение параметров функции формирования ключа. В wickr_kdf_meta_create_with_buffer с помощью функции _find_dkf_algo_with_id по первому байту значения zpt происходит выбор структуры, содержащей параметры функции формирования ключа (листинг 6). Так как первый байт zpt равен единице, становится понятно, что Wickr Me использует криптографическую функцию Scrypt [14] с параметрами, содержащимися в структуре KDF_SCRYPT_2_17 (определена в файле kdf.h библиотеки wickr-crypto-c):

- $N = 2^{17}$, N — параметр, задающий сложность вычислений;
- $r = 8$, r — параметр, задающий размер блока;
- $p = 1$, p — степень параллельности;
- $dkLen = 32$, $dkLen$ — размер выходного ключа.

```
wickr_kdf_algo_t *_find_dkf_algo_with_id(uint8_t algo_id)
{
    switch (algo_id) {
        case KDF_ID_BCRYPT_15:
            return &KDF_BCRYPT_15;
        case KDF_ID_SCRYPT_17:
            return &KDF_SCRYPT_2_17;
        case KDF_ID_SCRYPT_18:
            return &KDF_SCRYPT_2_18;
        case KDF_ID_SCRYPT_19:
            return &KDF_SCRYPT_2_19;
        case KDF_ID_SCRYPT_20:
            return &KDF_SCRYPT_2_20;
        default: return NULL;
    }
}
```

Листинг 6: функция _find_dkf_algo_with_id (здесь KDF_ID_SCRYPT_17 — это константа, равная единице, а KDF_SCRYPT_2_17 — это структура, содержащая параметры алгоритма)

Формирование промежуточного ключа шифрования. Далее в функции wickr_perform_kdf_meta с помощью функции Scrypt и выбранных ранее параметров по паролю пользователя и байтам zpt с 1 по 17 формируется 32-байтный ключ.

Расшифровка ключей шифрования данных приложения. Полученный на предыдущем этапе ключ используется для расшифровки оставшихся байтов значения zpt с использованием алгоритма AES-256.

Это происходит аналогично описанному в разделе 3.3 (параграф “Алгоритм расшифровки”) механизму. В результате расшифровки получается protobuf—сообщение, идентичное сообщению, полученному с помощью метода формирования ключа шифрования без использования пароля пользователя (рис. 2).

3.5. Расшифровка и извлечение данных

Поиск способа расшифровать необходимые данные в БД приложения был начат с метода decryptMessageWithMessage класса MessageDecryptor (был обнаружен в WickrEnterprise).

Цепочка вызовов внутри метода привела к функции decodePayload (библиотека wickr_crypto_objc), где происходит расшифровка значений колонки messagePayload таблицы ZWICKR_MESSAGE с помощью алгоритма AES-256 в режиме GCM, при этом:

- механизм расшифровки оказался идентичен принципу, описанному в разделе 3.3 (параграф “Алгоритм расшифровки”);
- в качестве ключа шифрования используется поле localStorage protobuf-сообщения, полученного на предыдущих этапах (листинг 5).

По результатам расшифровки были получены protobuf-сообщения неизвестного формата (рис. 3), содержимое которых менялось в зависимости от типа переданного сообщения (текст, геометка, медиафайлы и т.д.).

В декомпилированной библиотеке WickrMessagingProtocol был обнаружен код obj-c классов для сериализации и десериализации данного формата protobuf-сообщений, а также вложенных в них других protobuf-сообщений.

MessageBody. Описание формата protobuf-сообщений из колонки messagePayload таблицы ZWickr_Message удалось восстановить по методу serializedSize класса MessageBody (листинг 7).

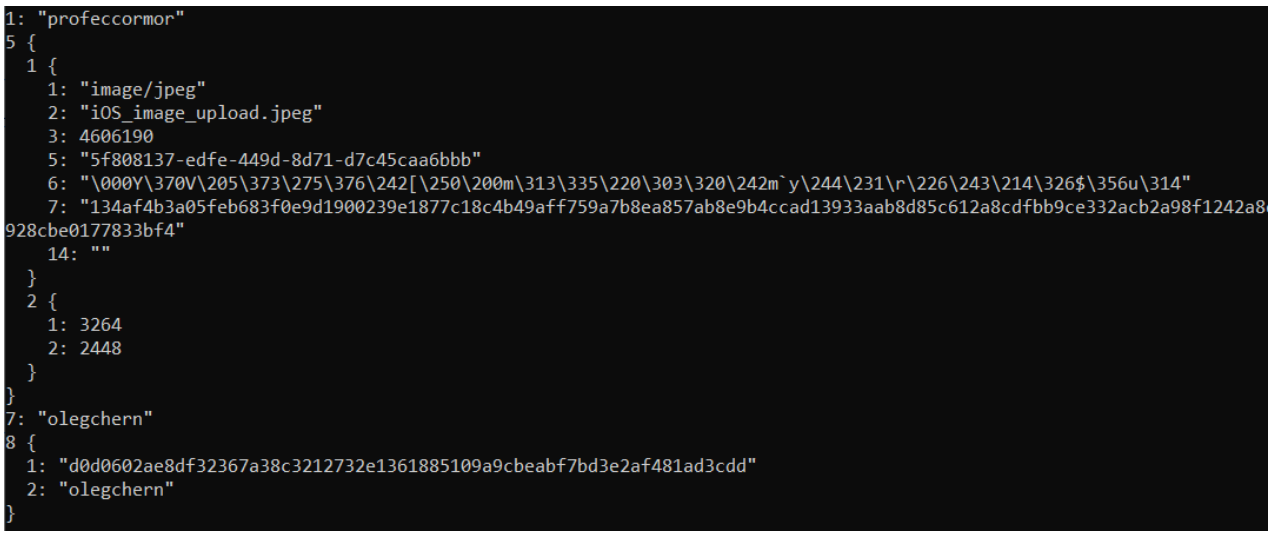

Рис. 3: Пример расшифрованного protobuf-сообщения из колонки messagePayload

```
message MessageBody {
    string senderUserName = 1;
    MessageBodyText text = 2;
    MessageBodyKeyVerify keyVerify = 3;
    MessageBodyControl control = 4;
    MessageBodyFile file = 5;
    MessageBodyCallMessage callmessage = 6;

    repeated string targetUsersArray = 7;
    repeated MessageBodyUser targetUsersv2Array = 8;

    MessageBodyLocation location = 9;

    MessageBodyEdit edit = 10;
    uint64 editTimestamp = 11;
    string inReplyTo = 12;

    MessageBodyDecryptionError decryptionError = 14;
    repeated MessageBodyUploadError uploadErrorsArray = 15;
    repeated MessageBodyReaction reactionsArray = 16;
    MessageBodyReactionMessage reactionMessage = 17;
}
```

Листинг 7: Описание формата MessageBody

По результатам дальнейшего исследования библиотеки удалось выделить поля MessageBody, содержащие информацию, которая может вызвать интерес экспертов в сфере цифровой криминалистики:

- senderUserName — строковое поле, содержит имя отправителя;
- text — вложенное protobuf-сообщение (MessageBodyText), содержит текст сообщения;
- file — вложенное protobuf-сообщение (MessageBodyFile), содержит информацию о прикрепленном к сообщению файле (имя, формат, ключ шифрования и др.);
- callmessage — вложенное protobuf-сообщение (MessageBodyCallMessage), содержит информацию о совершенном пользователем звонке (продолжительность, статус и др.);
- targetUsersArray — массив строк, содержащий имена адресатов;
- location — вложенное protobuf-сообщение (MessageBodyLocation), содержит информацию об отправленной геометке (широта и долгота).

MessageBodyFile. Информация о приложенном к сообщению файле хранится в поле file, имеющем тип MessageBodyFile (листинг 8).

```
message MessageBodyFile {
    string mimetype = 1;
    string name = 2;

    uint64 size = 3;

    string comment = 4;
    string guid = 5;

    bytes key = 6;
    string fileHash = 7;
```

```
    string uploadedByUser = 8;
    uint64 uploadedTimestamp = 9;

    string pinnedByUser = 10;
    uint64 pinnedTimestamp = 11;

    string modifiedByUser = 12;
    uint64 modifiedTimestamp = 13;

    string domain = 14;
}
```

Листинг 8: Описание формата MessageBodyFile

Из этого вложенного protobuf-сообщения можно извлечь метаинформацию о файле: его имя, размер в байтах, формат. Однако наибольший интерес представляют поля guid и key:

- guid представляет собой имя зашифрованного файла в поддиректории “Documents/download”, соответствующего прикрепленному к сообщению медиафайлу;
- key является 33-байтным ключом, который используется для расшифровки этого файла (в соответствии с описанным в 3.3 механизмом);

Таким образом, используя информацию из этого вложенного protobuf-сообщения, возможно расшифровать медиафайлы, которыми обменивались владелец аккаунта и его собеседники.

4. Версия Wickr Me для Android

В ходе исследования Android-версии мессенджера для изучения структуры хранимых Wickr Me данных и исполняемого файла приложения потребовалось снять логический образ файловой системы тестового устройства (смартфон Xiaomi Mi5). Как и в случае iOS-версии, это было сделано с помощью инструмента Belkasoft X.

4.1. Структура хранимых приложением данных

В 2016 году исследователями из Университета Кентербери Крайст Черч была опубликована работа [4], в рамках которой была исследована структура файлов версии приложения Wickr Me для платформы Android и был изучен механизм удаления информации с истекшим временем жизни.

Выяснилось, что все данные приложения хранятся в папке "/data/datacom.mywickr.wickr2", для доступа к которой исследователям понадобилось получить права администратора посредством рутинга устройства (рутингом называют процесс получения прав суперпользователя root, посредством эксплуатирования уязвимостей в системе). В ходе дальнейшего исследования были обнаружены поддиректории "databases" и "files".

В папке "databases" был обнаружен файл "wickr_db", у которого отсутствовали какие-либо файловые заголовки, из чего был сделан вывод, что данный файл является зашифрованной базой данных приложений. Исследователи декомпилировали исполняемый файл приложения и обнаружили класс "WickrDBAdapter" среди полученного исходного кода. В нем были найдены переменные, явно указывающие на природу хранимой в базе данных информации (см. листинг 9). Таким образом, было установлено, что "wickr_db" содержит личную информацию пользователей (имена пользователя и его контактов, их идентификаторы, публичные и приватные ключи) и их переписку.

В директории "files/temp", в свою очередь, были найдены файлы без расширения, у которых, как и у "wickr_db", отсутствовали какие-либо

заголовки. Посредствовом сопоставления количества прикрепленных к входящим сообщениям файлов и набора файлов с этим расширением было обнаружено, что в них в зашифрованном виде приложение хранит медиафайлы, которыми обмениваются пользователи.

```
private static final String DATABASE_NAME = "wickr_db";
private static final int DATABASE_VERSION = 12;
static final String Partner_User_KEY_Wickr_User = "wickrUser";
static final String Partner_User_KEY_avatarUrl = "avatarUrl";
static final String Partner_User_KEY_id = "_id";
static final String Partner_User_KEY_isHidden = "isHidden";
static final String Partner_User_KEY_status = "status";
static final String Partner_User_KEY_userName = "userName";
static final String TABLE_Partner_User = "Partner_User";
public static final String TABLE_Wickr_Account = "Wickr_Account";
```

Листинг 9: Фрагмент исходного кода класса WickrDBAdapter [4]

К сожалению, исследователям из Университета Кентербери Крайст Черч не удалось расшифровать данные приложения, что, тем не менее, не умаляет ценность их работы. Описанные выше результаты были перепроверены в рамках данной работы и полностью подтверждены: спустя четыре года структура файлов приложения не изменилась.

4.2. Исполняемый файл и используемые библиотеки

Как и в случае версии Wickr Me для iOS, было принято решение декомпилировать исполняемый файл приложения и некоторые из используемых им библиотек с целью изучения внутренней логики приложения и определения его механизма шифрования хранимых на конечном устройстве данных.

Для этого APK-файл (формат архивных исполняемых файлов-приложений для платформы Android) приложения был декомпилирован с использованием инструмента для реверс-инжиниринга с открытым кодом под названием JADX [22]. По результатам его первичного исследования выяснилось, что, как и в случае iOS-версии Wickr Me, ло-

гика шифрования и расшифровки данных вынесена в отдельную библиотеку libwickrCore, которая была декомпилирована с помощью упомянутого ранее интерактивного дизассемблера IDA Pro. В свою очередь libwickrCore полагается на базовые механизмы шифрования, реализованные в открытой библиотеке wickr-crypto-c.

4.3. Извлечение ключа шифрования без пароля пользователя

Исследование было начато с метода processLogin класса LoginManager, в котором происходит авторизация пользователя. Далее вызывается метод restoreCachedSession класса WickrSessionManager, где происходит попытка восстановить сохраненную сессию приложения. Здесь присутствует указание на два способа получения ключа шифрования: по паролю пользователя и по закэшированным значениям (листинг 10).

```
if (bArr != null) {
    // get db key from cache
    keyUsingPass = WickrDBKey.getKeyUsingCache(bArr);
} else {
    String str2 = str;
    if ((str2 == null || str2.length() == 0) || bArr2 == null) {
        keyUsingPass = null;
    } else {
        // get db key with password
        keyUsingPass = WickrDBKey.getKeyUsingPassword(str, bArr2);
    }
}
```

Листинг 10: Фрагмент метода restoreCachedSession, в котором упоминаются два способа получения ключа шифрования

Процесс расшифровки ключа шифрования без пароля пользователя начинается с извлечения содержимого файлов kck.wic и kcd.wic, содержащихся в поддиректории “files” директории приложения. Детальное изучение файлов показало:

- по своей структуре содержимое kcd.wic идентично значению zph

из базы данных Wickr Me для iOS (см. раздел 3.3);

- kck.wic содержит 33 байта информации, при этом первый байт — ноль, что, как и в случае с iOS-версией приложения, вероятнее всего указывает на то, что это ключ шифрования для алгоритма AES в режиме GCM.

Далее происходит вызов метода getUuid класса WickrSDKDevice, где из файловой системы устройства извлекается значение android_id [9], уникальное для каждой комбинации Android-приложения, его пользователя и устройства, на котором это приложение установлено. Полученное значение преобразуется в специальный идентификатор посредством метода nameUUIDFromBytes класса UUID стандартной библиотеки Java.

Наконец, полученные значения (содержимое kck.wic, kcd.wic и идентификатор, сгенерированный по android_id) передаются в качестве аргументов в нативную функцию recoverCachedKeys библиотеки libWickrCore. Следуя цепочке вызовов из recoverCachedKeys, удалось обнаружить функцию passwordCacheDecode, где происходит расшифровка значения из файла kcd.wic с помощью механизма, идентичного принципу, описанному в 3.3, только:

- в качестве ключа для первого этапа расшифровки используется преобразованное ранее значение android_id;
- в роли ключа для второго этапа выступает значение из файла kck.wic;
- вектор инициализации, тег аутентификации и шифротекст берутся из файла kcd.wic;

В результате расшифровки получается protobuf-сообщение StorageKeys, описанное ранее (листинг 5). Это сообщение содержит ключ шифрования данных.

4.4. Формирование ключа шифрования по паролю пользователя

Как было отмечено ранее метод restoreCachedSession класса WickrSessionManager содержит также и описание механизма формирования ключа шифрования по паролю пользователя.

В рамках этого механизма извлекается содержимое файла sk.wic (содержащегося в упомянутой ранее директории "files"), которое структурно оказалось идентично значению zpt из версии Wickr Me для iOS (см. раздел 3.4). Далее, оно вместе с паролем пользователя в качестве аргумента передается в нативную функцию getDBKey из библиотеки libWickrCore. Цепочка вызовов в этой функции приводит к wickrStorageKeysInitWithData, где вызывается уже упомянутая ранее функция wickr_ctx_import_storage_keys библиотеки wickr-crypto-c, в которую и передается содержимое sk.wic и пароль (листинг 11).

```
v31 = __readgsdword(0x14u);
v2 = CFStringCreateExternalRepresentation(0, a2, 134217984, 0);
result = 0;
if ( a1 && v2 )
{
    wickr_crypto_engine_get_default(&v30);
    v28 = CFDataGetLength(a1);
    v29 = CFDataGetBytePtr(a1);
    v26 = CFDataGetLength(v2);
    v27 = CFDataGetBytePtr(v2);
    qmemcpy(&v5, &v30, 0x54u);
    v4 = wickr_ctx_import_storage_keys(&v28, &v26);
}
```

Листинг 11: Фрагмент функции wickrStorageKeysInitWithData, где происходит вызов из wickr_ctx_import_storage_keys библиотеки wickr-crypto-c

Далее механизм расшифровки полностью повторяет принцип, описанный в 3.4. Единственное отличие — вместо значения zpt для формирования ключа используется содержимое файла sk.wic. Результатом

расшифровки, так же, как и в предыдущих случаях, является protobuf-сообщение StorageKeys (листинг 5), содержащее ключ шифрования данных приложения.

4.5. Расшифровка и извлечение данных

В отличие от версии Wickr Me для iOS, в версии для Android база данных приложения зашифрована, поэтому в процессе получения данных пользователя необходим дополнительный этап ее расшифровки.

В файлах приложения присутствует библиотека SQLCipher [10], также она упоминается и в полученном в результате декомпиляции коде (листинг 12). Из этого был сделан вывод, что именно с помощью нее и была зашифрована база данных Wickr Me. Однако выяснить точный механизм преобразования ключа шифрования в пароль от БД по коду не удалось, и поэтому было принято решение модифицировать приложение, добавив журналирование пароля с помощью инструмента logcat, повторно собрать его, развернуть на тестовом устройстве, извлечь пароль от базы данных и сравнить его с полученным ранее ключом шифрования, определив таким образом связь между ними.

```
import net.sqlcipher.database.SQLiteDatabase;
import net.sqlcipher.database.SQLiteDatabaseHook;
import net.sqlcipher.database.SQLiteOpenHelper;
import timber.log.Timber;

public class WickrDBAdapter extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "wickr_db";
    private static final int DATABASE_VERSION = 51;
```

Листинг 12: Фрагмент файла, содержащего класс WickrDBAdapter, где упоминается библиотека SQLCipher

Данная процедура была проведена с использованием плагина APKLab [1] для редактора исходного кода Visual Studio Code. По ее результатам оказалось, что паролем к базе данных Wickr Me является hex-строка, сформированная по полю localStorage protobuf-сообщения

StorageKeys (листинг 5), которое было извлечено на предыдущем этапе.

```
GnssLocationProvider: WakeLock acquired by sendMessage(REPORT_SV_STATUS, 0, com..
GnssLocationProvider: WakeLock released by handleMessage(REPORT_SV_STATUS, 0, co
log-tag : 00fb2c2c95557c8133b7a4b92e3508c035fa79c2f570c605083a84ab7a09febbf1
GnssLocationProvider: WakeLock acquired by sendMessage(REPORT_SV_STATUS, 0, com..
GnssLocationProvider: WakeLock released by handleMessage(REPORT_SV_STATUS, 0, co
```

Рис. 4: Фрагмент журнала, полученного с помощью утилиты logcat, с паролем от базы данных приложения

Базу данных приложения удалось расшифровать с помощью библиотеки SQLCipher. Ее внутреннее устройство во многом повторяет устройство базы данных версии Wickr Me для iOS: также присутствуют таблицы, содержащие данные о пользователе, его собеседниках, их переписке и приложенных к сообщениям медиафайлах. Более того, часть полей зашифрована, и получить доступ к ним возможно, используя тот же метод, что и в случае iOS-версии приложения (см. раздел 3.5, параграф “Расшифровка и извлечение данных”).

Информация о сообщениях, которыми обменивались пользователи, хранится в колонке messagePayload таблицы Wickr_Message, содержимое которой представляет собой protobuf-сообщения в формате, описанном ранее в главе 3.5. Таким образом, извлечение данных, представляющих интерес для экспертов в сфере цифровой криминалистики происходит аналогично iOS-версии приложения.

5. Реализация

Описанный в предыдущих разделах механизм расшифровки и извлечения данных из приложения Wickr Me был реализован и интегрирован в продукт компании Белкасофт Belkasoft X.

Часть логики, отвечающая за формирование ключа шифрования данных, была написана на языке программировния C++ с использованием открытой криптографической библиотеки OpenSSL [12]. Она представляет собой набор функций, используемых основной частью приложения, написанной на языке C#, посредством механизма вызова неуправляемого (unmanaged) кода:

- CreateWickrKeyFromPassword — функция, реализующая механизм формирования ключа шифрования данных по паролю пользователя. В качестве параметров принимает строку-пароль и сам зашифрованный ключ (zpt или android_id в зависимости от платформы). Использует реализации алгоритмов AES-256-GCM и Scrypt библиотеки OpenSSL;

- CreateWickrKeyFromKeychain — функция, используемая для расшифровки пароля пользователя по значениям из iOS Keychain. В качестве параметров принимает значения zph из базы данных iOS-версии приложения, BundleID, а также !devid! и activeAccount из iOS Keychain. Использует реализацию режима GCM алгоритма AES-256 библиотеки OpenSSL;

- CreateWickrKeyFromDeviceId — функция, аналогичная CreateWickrKeyFromKeychain, но использующаяся для расшифровки ключа шифрования в версии приложения для Android. Принимает на вход содержимое файлов kck.wic и kcd.wic, а также значение android_id. Также полагается на реализацию AES-256-GCM библиотеки OpenSSL.

Основная часть логики, отвечающая за взаимодействие с пользователем, расшифровку базы данных и ее значений, а также извлечение

данных, была реализована на языке C# в соответствии с устоявщейся архитектурой Belkasoft X. Механизмы по расшифровке iOS Keychain и взаимодействию с библиотекой SQLCipher на тот момент уже были реализованы сотрудниками компании.

6. Апробация

Реализованное решение было апробировано с использованием двух тестовых устройств (в обоих случаях установленная версия Wickr Me — 5.74.7):

- смартфон iPhone 5S (операционная система iOS) с установленным Jailbreak Unc0ver;
- смартфон Xiaomi Mi5 (операционная система Android), рутированный.

Тестовые данные были созданы с использованием аккаунта Wickr, предоставленного компанией Белкасофт.

6.1. Извлечение данных из полного логического образа устройства без пароля пользователя

В ходе первого эксперимента с помощью Belkasoft X с тестовых устройств был снят полный логический образ системы и был запущен процесс извлечения данных из установленных версий Wickr Me. Belkasoft X автоматически обнаружил в образах файловой системы необходимые для расшифровки значения и извлек все данные пользователя тестового аккаунта.

	Тестовый аккаунт
Текстовые сообщения	19
Медиафайлы	5
Контакты	6
Геометки	2

Таблица 1: Количество единиц данных разного типа в образе тестового аккаунта

6.2. Извлечение данных из фрагмента логического образа устройств с паролем пользователя

В ходе второго опыта из полученных в прошлом эксперименте полных логических образов тестовых устройств были извлечены фрагменты, соответствующие только данным, хранимым Wickr Me, и их анализ был запущен в Belkasoft X. В ходе анализа был введен запрошенный с помощью диалогового окна пароль от тестового аккаунта. Как и в предыдущем опыте, были извлечены все данные тестового аккаунта (таблица 1). Пример извлеченных данных представлен на рис. 5.

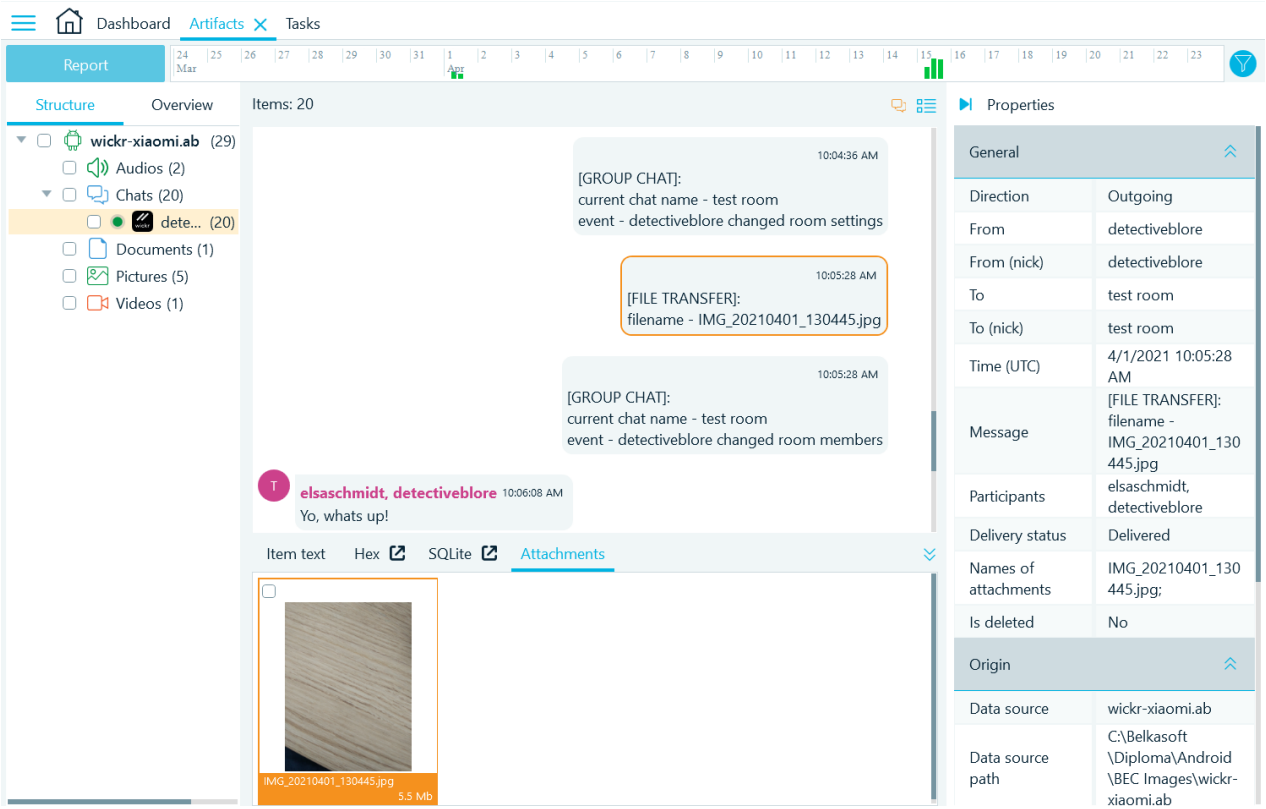

Рис. 5: Пример извлеченных в ходе эксперимента данных

Заключение

Таким образом, были достигнуты следующие результаты:

- было описано, каким образом Wickr Me хранит данные пользователей и какая их часть является зашифрованной в версиях для Android и iOS;
- были определены применяемые приложением алгоритмы шифрования данных: Wickr Me использует алгоритм AES-256 в режиме GCM, а также функцию формирования ключа Scrypt;
- были описаны два способа получения ключа шифрования данных Wickr Me в версиях для iOS и Android: один из них подразумевает использование сохраненных в файловой системе устройства значений, другой — пароля пользователя;
- механизм расшифровки данных приложения с помощью полученного ключа был также изложен. Описан формат protobuf-сообщений, в которых эти данные хранятся;
- описанный механизм был реализован с помощью языков программирования C# и C++, интегрирован в продукт Belkasoft X компании Белкасофт и апробирован.

Отдельно хочу выразить благодарность специалистам компании Белкасофт, в частности Никите Тимофееву и Михаилу Виноградову, за предоставление тестовых устройств и лицензий для использованных программных инструментов, а также за неоценимую помощью в исследовании и написании текста выпускной квалификационной работы.

Список литературы

[1] APKLab — Android Reverse-Engineering Workbench for VS Code. — online; accessed: https://github.com/APKLab/APKLab (online; accessed: 25.04.2021).

[2] Dworkin Morris, Barker Elaine, Nechvatal James et al. Advanced Encryption Standard (AES). — 2001. — 2001-11-26.

[3] Apple. Keychain Services. — online; accessed: https://developer.apple.com/documentation/security/keychain_services (online; accessed: 09.12.2020).

[4] Barton Thomas, Azhar M A Hannan Bin. Forensic Analysis of the Recovery of Wickr's Ephemeral Data on Android Platforms. — 2016. — 10.

[5] Cellebrite. Industry-first: Forensically sound, full file system extraction with Cellebrite Responder. — online; accessed: https://shorturl.at/sDIT4 (online; accessed: 09.12.2020).

[6] Inc. Apple. Bundle IDs. — online; accessed: https://developer.apple.com/documentation/appstoreconnectapi/bundle_ids (online; accessed: 25.04.2021).

[7] Inc. MSAB. XRY – Extract. — online; accessed: https://www.msab.com/products/xry/ (online; accessed: 09.12.2020).

[8] L. Vidino S. Hughes. ISIS in America: From retweets to Raqqa. — online; accessed: http://www.stratcomcoe.org/download/file/fid/2828 (online; accessed: 09.12.2020).

[9] LLC Google. ANDROID_ID. — online; accessed: https://developer.android.com/reference/android/provider/Settings.Secure#ANDROID_ID (online; accessed: 25.04.2021).

[10] LLC Zetetic. SQLCipher — open-source extension to SQLite. — online; accessed: https://www.zetetic.net/sqlcipher/ (online; accessed: 25.04.2021).

[11] Magnet Forensics. Using Magnet AXIOM to Get Data From Wickr Me. — online; accessed: https://www.magnetforensics.com/resources/using-magnet-axiom-to-get-data-from-wickr-me/ (online; accessed: 09.12.2020).

[12] OpenSSL — cryptography and SSL/TLS toolkit. — online; accessed: https://www.openssl.org/ (online; accessed: 25.04.2021).

[13] Oxygen Forensics. Wickr some forensics up! — online; accessed: https://blog.oxygen-forensic.com/wickr-some-forensics-up/ (online; accessed: 09.12.2020).

[14] Percival C. The scrypt Password-Based Key Derivation Function draft-josefsson-scrypt-kdf-05. — online; accessed: https://tools.ietf.org/html/draft-josefsson-scrypt-kdf-05 (online; accessed: 25.04.2021).

[15] Ravnås Ole André V. Frida — Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. — online; accessed: https://frida.re/ (online; accessed: 25.04.2021).

[16] Roussinous A. The social media Accounts of British Jihadis in Syria just got a lot more distressing. — online; accessed: http://www.vice.com/en_uk/read/british-jihadis-beheadingprisoners-syria-isis-terrorism (online; accessed: 09.12.2020).

[17] SA Hex-Rays. IDA Pro — A powerful disassembler and a versatile debugger. — online; accessed: https://www.hex-rays.com/ida-pro/ (online; accessed: 25.04.2021).

[18] Torok R. How social media was key to Islamic State's attacks

on Paris. — online; accessed: http://shorturl.at/nqzY9 (online; accessed: 09.12.2020).

[19] Wickr Inc. The Most Secure Private Collaboration Platform. — online; accessed: https://wickr.com/ (online; accessed: 09.12.2020).

[20] Wickr Inc. Why Wickr? — online; accessed: https://wickr.com/why-wickr/ (online; accessed: 09.12.2020).

[21] Wickr Inc. wickr-crypto-c — An implementation of the Wickr Secure Messaging Protocol in C. — online; accessed: https://github.com/WickrInc/wickr-crypto-c (online; accessed: 25.04.2021).

[22] jadx - dex to Java decompiler. — online; accessed: https://github.com/skylot/jadx (online; accessed: 25.04.2021).