JOHANNES KEPLER
UNIVERSITY LINZ

Author
Angela Laguia Jimenez

Submission
Institute of
Networks and Security

Thesis Supervisor
Univ.-Prof. DI.
Dr. Rene Mayrhofer

December 2021

Secure Chat Export for Signal-Android

Bachelor's degree thesis

to obtain the academic degree of

Bachelor of Science (BSc)

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
The submitted document here present is identical to the electronically submitted text document.

Place, Date

Signature

Abstract

This bachelor thesis looks at the development of securely exporting single conversations in IM for Android apps, specifically for the Private Messenger Signal-Android, a cross-platform centralized encrypted messaging service that is free and AOSP. Initially, this paper looks at the existing Messenger apps and their chat export tools that allow users to obtain similar results as the designed feature. This document evaluates the user's risks when trusting these other services but does not reflect much about its components or technological aspects. The present paper ignores OSs and platforms other than Android, considering that the main point of the work lies in the security of the IM app and not on the device reliability.

The writing re-examines the characteristics of different Messenger apps and presents a simple solution for exporting single chats in Signal. In this context, some other options as apps that present alternatives to the designed choice were researched. The proposal of the chat export feature for Signal-Android begins in the next section, explaining mostly primary functionalities for exporting chats, including the functionality analysis, the solution design, and the implementation. The following chapter focuses on developing this new function added to the Signal-Android APP and compares the obtained outcomes with other results of a similar solution. The final part summarizes the project, explains the problems and reviews possible improvements and subsequent development steps.

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung des sicheren Exports von Einzelgesprächen im Bereich des IM für mobile Android-Apps, konkret für den Private Messenger Signal, ein zentralisierter verschlüsselter Cross-Plattform Messaging Dienst, der kostenlos und Open-Source ist.

Zunächst zeigt diese Arbeit einen Überlick über die bestehenden Messenger Apps und ihre Chat-Export-Tools, die Benutzern erlauben, ähnliche Ergebnisse wie das entworfene Feature zu erhalten. Sie bewertet die Risiken des Nutzers, wenn sie diesen Diensten vertrauen, aber geht nicht tief auf deren Bestandteile oder deren technologische Aspekte ein. Andere Betriebssysteme und Plattformen außer Android werden ignoriert, wenn man bedenkt, dass der Fokus der Arbeit auf der Sicherheit der IM-App liegt und nicht auf der Zuverlässigkeit des Geräts.

Das Schreiben überprüft erneut die Eigenschaften von verschiedenen Messenger Apps und stellt eine einfache Lösung für den Export einzelner Chats aus Signal vor. In diesem Zusammenhang wurden einige Apps untersucht, die Alternativen zu der entworfenen Option bieten. Der Vorschlag der Chat-Export-Funktion für den Signal Messenger für Android beginnt im nächsten Abschnitt, erklärt hauptsächlich primäre Funktionalitäten für den Export von Chats einschliesslich der Funktionsanalyse, des Lösungsdesigns und der Implementierung. Das dritte Kapitel beschäftigt sich mit der Entwicklung dieser neuen Funktion in Signal-Android und vergleicht das Resultat mit einer ähnlichen Lösung. Eine Zusammenfassung, einige Problemdarstellungen sowie eine Diskussion über Verbesserungen und weitere Entwicklungsschritte werden im finalen Abschnitt behandelt.

Index

Abbreviations

AES	Advanced Encryption Standard
AOSP	Android Open Source Project
API	Application Programming Interface
APK	Android Application Package
APP	Mobile Application
AVD	Android Virtual Device
CBC	Cipher Block Chaining Algorithm
DH	Diffie–Hellman key exchange
E2EE	End-to-End Encryption
GDPR	General Data Protection Regulation
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HTML	HyperText Markup Language
HMAC	Keyed-Hash Message Authentication Code
IDE	Integrated Development Environment
IM	Instant Messaging
iOS	iPhone Operating System
IP	Internet Protocol

IP address	Unique address that identifies a device or local network by using IP for communication
IV	Initialization Vector
JSON	JavaScript Object Notation
KB	Kilobyte
MAC	Message Authentication Code
MiTM	Man-in-the-Middle
MTProto	Mobile Transport Protocol
OS	Operating System
QR Code	Quick Response Code
SDK	Software Development Kit
SRTP	Secure Real Time Transport Protocol
RSA	$Rivest - Shamir - Adleman$ Asymmetric cryptography algorithm
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSD	XML Schema Definition
ZIP	Archive File Format that supports lossless data compression

List of Figures

List of Tables

1 Introduction

1.1 Motivation

IM programs allow smartphone owners to communicate in one-to-one conversations, a chat room for just two people, or group chats. The group's creator generates the channel of communication and usually has the right to add more people. Here every member can read and answer the messages. With Messenger apps, users can text themselves, share links, graphics, multimedia files and make video or voice calls. There is an extended list of chat clients; not all are part of chat-oriented programs but as an additional service to the main app.

The OS of the user's device is essential. It provides the set of programs that manage the hardware and the tools to handle it. Moreover, it determines which other programs can run on the device. That is why applying a single solution is not an option for the IM Signal-Android itself, but designing different solutions is expected for each platform on Android, iOS and each Desktop OS. Nevertheless, all these Signal apps share the same Signal Server. To participate, users need to download Moreover, install a chat client on their device to connect with the chat server, managing all conversations.

The most relevant chat clients have been updating their services over the years. They improved their appearance and usability to conform to their community taste, make the product engaging, compatible with different users' devices and can handle multiple protocols so that clients can talk with several users simultaneously. Other deciding properties for choosing one messaging program over others are cloud synchronization, conversation options, identity and data security, privacy, cost, analytic or ads. Among them, data privacy has become one of the biggest worries for many IM users. There is not much interest in legislating against the private use of user data[1].

Over the last few years, more and more people are migrating to other messaging services like Signal that provide data protection using E2EE protocols for all chats and calls. To identify the user, it saves the telephone number, using unique safety numbers to ensure users the protection of their messages, chat rooms and calls from external intrusions. After May 2018, the GDPR guarantees the right to data portability from any digital environment, which helps to transfer and recover the interchanged personal data within a system. However, this legal basis simultaneously exposes many questions: no law controls the services that manage the transferred data, besides users need to understand and interpret this data; furthermore, the misuse of data after their transfer may occur. However, what happens next? How would an ideal model carry out this portability?[2][3].

On the other hand, Signal does not store the interchanged data on their servers but the user's devices. One question arises concerning the stored data if someone accidentally deletes it from the device or if the user loses the device. Consequently, users cannot recover their messages, files, audio files or pictures. This fact will inevitably concern the way a user manages its stored data because, at the moment, a whole backup for recovering old messages is required. Most of the chatting apps support a backup, creating a copy of the current status of the environment. That generates a complete snapshot of the stored data from all services, including irrelevant data to the userś needs.

Given this, exporting single conversations is a pretty exciting choice. It allows easy access to past messages after a time, forgetting the complexity of recovering from a backup. Furthermore, it can be an attractive option that allows users to transfer chats onto different platforms to migrate their activity to alternative online places.

1.2 Aim of the work

This thesis aims to analyse and review the development of a secure chat export tool for single one-to-one or group conversations in Signal-Android.

Exporting chats allows storing interchanged messages and files, including media in general, and interpreting them through an external visualisation tool. Here transferring the result to other online services is not considered, but there is no doubt that this feature would complete the intention of the whole project. The integration of this solution takes place in Signal-Android. The security assurance in the developed service arises by taking the chat data over the same app functionality used to print them in the conversation content view. There are no deviations of data to somewhere else. After choosing the export chat option, the system copies the content of this conversation in a file, includes other media directly taken from the user device, and packs everything in a ZIP file. The user device saves this package without the intervention of other services.

One of the significant advantages of this proposal over the full backup is the prospect of keeping the messages users wish to preserve. Additionally, it can limit the chat content for a specified period or include shared files, an HTML Viewer, or both. That allows users to reread the conversation comfortably.

While full backups take longer to perform and retrieve, they are redundant and have many recovery restrictions. The secure chat exportation takes less time than full backups. Opening the resulting file does not depend on the chat platform and can be viewed independently.

1.3 Related Works

This section introduces the following: an overview of some popular Messenger services, as well as a comparison and contrast with existing approaches (e.g., built into other secure or non-secure messengers or export tools), how they handle the export of peer-to-peer communication and group conversations, the level of protection they offer while communicating, and, in this context, the differences between each other.

- **IM solutions**

1.3.1 Facebook Messenger

The recently announced company brand Meta, parent of the most popular social networks, integrated IM with the fusion of Messenger into its Facebook framework for ten years. It added a bunch of new functionalities like video calls, group messages or payment, among others. Facebook collects a large amount of personal data such as biometric facial data, habits, contact details, personal contacts, logs IPs, timestamps and much more. The company categorizes this information and uses it to select and filter their target public, and offers it strategically to advertisers for their profit, commercing with them.[4][5]. Messenger currently encrypts messages and attachments just in transit as well as the stored message log. Nevertheless, Facebook holds the key so that third parties like authorities, hackers, or the company itself could access the users private content. Just the organisation policy protects users data. No robust encryption techniques are applied.[6]. An integration plan of E2EE in Messenger by default is unknown, but this option can be enabled manually.[7][8].

Downloading Facebook posts, notes, or Messenger chats from both a desktop and a mobile device is an option that Facebook presents. The social network member can find it under *Settings & Privacy → Account Settings → Your Facebook Information → Download Your Information*. After selecting this option, the user shall see a long list of different kinds of data that Facebook stores (including IM chats)[9][10]. Users can select the content they are interested in, choose a date range, pick which format (HTML or JSON) they wish and the quality of the media files too. Later, the system works on the creation of a package that contains all these pieces of information. This data export does not work immediately, but Facebook sends a notification to the user when it is available to be retrieved. Before considering downloading, users must introduce their password. This copy of the user data collection will be available to be downloaded for just a few days under *Download Your Information → Available Copies*.

Nevertheless, the only way a user can extract a single chat history is to download all Messenger data.

1.3.2 WhatsApp

WhatsApp is considered one of the world's fastest-growing technology companies, was sold to Facebook in 2014 and now is part of the platform Meta. This IM company always had a clearly defined business model that offers a free, fast messaging service worldwide over the phones. Today more than two billion users have created a WhatsApp account, making the company the largest messaging app. Over the years, this company has been developing the product to send media content, build E2EE, create business accounts over other features[11]. Although, at the beginning of 2021, Whatsapp introduced a new privacy policy that allows them to collect and share user data with other businesses: phone numbers, device ID, location, transaction data, product interaction or user identifiers[12][13].

End-to-End Encryption The E2EE system and the algorithm they use correspond to the one presented in the Signal Protocol library, an Open-Source library.

For further understanding of the security which applies this E2EE protocol, a synopsis follows:[14]

- Trust establishment
  - Client and Server interchange public keys to build an encrypted session creating a master_secret by using asymmetric encryption.
  - Additionally, it establishes a Root-Key and Change-Keys; those will remain until the app's internal status changes on the device.
- Conversation security protocol
  - Users can send messages, as well if the recipient is connectionless. In this case, the recipient will build the session when the device is available. The sent message will include session setup information in its header.
  - The master_secret is calculated using this header information and gets the derived Root Key and Chain Keys once the recipient receives the message. Every message sent is protected with a unique Message Key (80 bytes) in CBC mode with HMAC for authentication.
  - For the response, a new Chain Key and Root Key are calculated based on other calculations between ephemeral public keys of sender and recipient.
- Large files are equally encrypted. A blob store stores the attachment. Afterwards, the recipient receives a standard encrypted message pointing to the encrypted blob and decrypts the data.
- Group messages are dispatched by building pairwise encrypted sessions and applying server-side fan-out where the server spreads a single message for each other member of the group instead of the recipient itself (client-side fan-out).
  - For delivering messages to all group members, the protocol gives encrypted Sender

Keys to each group participant for the first time.

    * Subsequent messages are sent by using Message Keys as described above. The sender transmits the ciphertext to the server, which distributes it to all members of the group.

    * When the number of group participants changes, all members remove the Sender Key and renew them as mentioned.

- Calls apply E2EE likewise: An encrypted session must be created in the first place if needed. Then the caller creates an encrypted message announcing an incoming call and attaches a random 32 bytes SRTP master secret. By accepting this call, an SRTP encrypted call is held.

The chat export feature developed by WhatsApp contemplates two alternatives.

- The first one is to create a backup either manually or automatically. The encryption of the messages in the backups stored in the cloud allows everyone with access to read the data[13].

- The second one is to send a copy of a single chat history as a TXT file via email.

Precisely, this last option can be found by touching the options menu and then, users select if they want to attach media files. The full report provided by the company[15] informs that around 40000 messages can be included when the chat to export does not include multimedia content. Contrarily with multimedia content, the file contains around 10000 messages. Considering the last option, while trying to export an extensive chat, some media files were included, but a lot were missing.

This type of export solution they offer shows limitations due to the maximum email size and large files that can be missing.

1.3.3 Telegram

Telegram is a strong growing community that counts with more than half a billion users. It is a free and fast cloud service. It allows synchronisation on many devices and uses an encryption system for chats, groups or multimedia. Regarding data collection: Telegram stores name, phone number, contacts, user identification and profile pictures. There are two different types of chat rooms on Telegram: cloud or secret chats. They store the cloud messages on their servers and do not keep any information about other secret chats, except for media content in secret chats[16].

The network's security is based on the MTProto 2.0 Mobile Protocol, a combination of AES symmetric encryption, RSA encryption, and DH[17][18]. For a better perception of this security, a revision follows:

- The client creates an authorization key at the beginning, shares it with the server, and obtains its server salt for all future communications.
- The encryption of sessions begins with a 64-bit number created with the authorization key.
- For ordinary chats, it seems that Telegram does not use end-to-end encryption by default, but for secret chats where servers are not involved, although client-server communication is encrypted.
- The encryption of messages is defined as follows. An external header at the beginning of the message contains an identifier and a message key. Together with the user key, this is applied to cypher these messages.
- Voice calls are E2EE using the 3-DH handshake instead of the original two messages between the parties, to avoid MiTM attacks on DH.

Regarding the export chats feature, Telegram covers this option under two different points of view:

1. Although the company does not offer an automatic backup option, it can be done manually via Telegram Desktop
2. Chat Export tool on Telegram Desktop.

This option allows the user to export all types of user information, the contact list, private chats, bot chats, private and public groups, own messages, private and public channels, exchanged multimedia content and exchanged files, the sessions data and other miscellaneous data. This choice is on the desktop APP, under *Settings* → *Advanced* → *Export Telegram data.* Then a new window facilitates the selection of the information that should be exported on clicking *Export.*

1.3.4 Wire

Wire is another popular IM AOSP app that provides private and secure communication with others, supporting all essential characteristics of IM apps. The server collects metadata during the client registration, such as device type, model, timestamp, IP location or authentification cookie. However, the device stores necessary local data. This service encrypts the messages, all types of files and calls by using E2EE, as well as it stores the data on their servers[19][20]. For a better understanding of this security, a revision follows:

– API authentication uses tokens and cookies, persistent or session-based.

– To send messages, users need a cryptographic session with each participant. Text messages are E2EE.

– Assets shared are uploaded in plain text and encrypted before sending. The server stores these assets and allows the user to recover them at any time.

– The encryption of messages is defined as follows. An external header at the beginning of the message contains an identifier and a message key. Together with the user key, this is applied to cypher these messages.

– 1-to-1 calls are SRTP-encrypted using the DTLS handshake and the authenticity of the participants is checked during the handshake.

For conversation backups, the multi-device platform allows a copy of the usersćonversations history, protecting it with an encryption password chosen by the user. It just can be restored on the same platform [21]. As an AOSP, Wire can be extended and indeed there exists a fork implementing a single chat secure export feature on this platform[22]. This export functionality has been an inspiration for this project.

1.3.5 Signal

Signal arrived as a solid alternative for the famous IM services. Since Signal is an AOSP, it has its code available for everyone and is open for testing, user changes and experiments. It is easy to analyse and adapt. Protecting user data has been considered one of the relevant aspects of the platform. Until now, most IM apps collect other unnecessary metadata, linking the phone to the user identity. With this in mind, the only user data Signal retains is the telephone number to identify the end-user, date of account creation and date of last use[23].

In addition, Signal has developed a cryptographic system based on the idea of fingerprints, a unique identifier for each person. They developed a numeric encoding system which is called Safety Numbers. Having this option, users can ensure that the communication channel is private. This encoding does not just depend on their devices but the different chat rooms, and can be done by scanning a QR code or number comparisons.

However, the fundamental security aspect of the IM service is the E2EE protocol designed by Signal, which protects the conversations and calls, making them secure and private. It is hard to say how secure it is because there is no definition of security characteristics that must be accomplished[24].Indeed, last year a bug was detected. Some users sent random images to random contacts. This fact has partially compromised the privacy of user data[25]. Additionally, a list of detected vulnerabilities[26] and many other issues are described[27] to be fixed.

As an open-source project, Signal has many contributors who build, maintain, test, etc. the software application daily. A backup service for the whole chat history is already offered by Signal. However, the export of single chats is an extra feature that the Android app does not include at the moment.

- **Third-party solutions**

1.3.6 Exporter For Facebook/WhatsApp

The Exporter app for conversations on external platforms such as Facebook or WhatsApp by GilApps (totally independent of the mentioned enterprises) are available in the Google Play Store for Android devices with at least Android 4.1., offer a trial version for free and activate all features with an In-App purchase.
Usability:

- The services offer users conversation transformation to plain text, PDF including multimedia content and text, CSV as a datasheet, JPG or HTML files.
- Besides, users can customize the resulting output by using different themes, including a preview.
- Finally, users can select conversations and send them by email or print them with an additional app that provides a printer service with the desired format in the File Management System.

This application offers the users an extensive list of choices for designing the output of the conversations, users can export the results to different formats and receive them in many different ways. There are some security problems detected in this type of application. When downloading an external app for exporting chats, granting that app permissions to access chats and media is unsafe. Trusting an external service without any guarantees may fail or not cover exceptional cases, and providers do not take responsibility for mistakes. Giving access to private data to an unknown third party that does not provide detailed documentation about what service they provide apart from the description on the Google Play Store is not trustworthy.

1.4 Comparison summary

Table 1.1: Comparison of conversation export tools for different applications in terms of features, security and privacy [28]

	Messenger	WhatsApp	Telegram	Wire	Signal Android	Exporter Tool
Open Source	no	no	partially	yes	yes	no
Single Chat Export	no	yes	yes	yes[22]	no	yes
Backup	yes	yes	yes	yes	yes	yes
Amount of user data collection	+++	+++	++	+	+	?
E2EE Encryption for messages	no	yes	yes	yes	yes	no
Metadata Encryption	no	yes	yes	yes	yes	no
Log of IPs and timestamps	yes	yes	yes	no	no	?
Third Data Protection	no	yes	yes	yes	yes	no

2 Methodology

2.1 Specification

2.1.1 Time schedule

The itinerary is the following:

1. Observation phase: ideas come from other related works
2. Settle the project and development environment
3. Study different approaches and create a project plan
4. List the requirements
5. Architecture
6. Detailed Design
7. Implementation and testing
8. Beta testing, review
9. Submit a pull request

Additionally, there was no deadline for completing this work.

2.1.2 Installation and Deployment

The feature was developed and built on the current Google-backed Android IDE, i.e. Android Studio powered by the IntelliJ IDEA platform. The implemented feature was tested on an x86 android emulator with APIs 26, 29 and 30.

As Signal Android is an open-source project [29], it is necessary to build it from the sources. These are the steps that have been taken [30]:

- Install Android Studio
- Install the SDKs and Add-Ons
- Import the root project
- Set up the AVD

- Build Gradle and avoid some shown errors
- Build the project

2.2 Approach

First, using Signal-Android to get familiar with it is a fundamental part. Considering relevant aspects in order to adapt the new functionality to the whole project meaningfully is expected.

As stated above, our primary purpose is to develop a tool for exporting single conversations for the Signal app on the Android platform. The different implementation approaches are the following:

- Choosing the format of the exported data:
  Regarding this aspect, it was clear to export the data in a way that could be interpreted again by other platforms or by using other tools. This resulted in the idea of exporting the data in an XML formatted file. XML uses a human-readable language, supports Unicode, data can be read and displayed with the HTML viewer. Furthermore, it allows using XSD schema to ensure the document does not contain errors. Additionally, there are implemented solutions in other IM services that use the XML language to extract information. A future project could consider the transfer of single conversations between services using a standard XSD schema. That is the reason why other similar options like JSON were discarded directly in favour of XML.

- Design of the interface:
  The Export Chat option had to be included in the proper chat the user wants to export, either under the Conversation Settings or under the three-dots Menu in the top-right corner. As Conversations Settings is, by the way, present in the three-dots menu as an option and adding an extra option to this menu could result in loss of simplicity on the APP's design. Placing the Export Chat option under the conversation settings was essential. Here, the settings show separation between elements options ordered by priority and separated from the different choices.

- Location to save the chat:
  Since our aim is, as mentioned in the Introduction, to keep the export safe, no third parties need to be involved. Therefore the elected solution was to save a ZIP file containing the conversation data on the personal Android device.

2.3 Implementation plan

Here the implementation plan responsible for chat exporting is shown. Therefore, it is necessary to:

- Create a Signal Master Fork and a new branch to implement our solution.
- Create an activity in the Signal Android project for the implementation of this functionality.
- Accessing the chat and its content.
- Create a XML document for storing the conversation details and content.
- Additionally, create an HTML viewer, where users can recall their conversations by selecting the created XML file.
- Attach files to the ZIP File.
- Allow the user to choose a download folder.
- Continuously update the system to the latest master version and apply changes if needed.

2.4 Testing plan

In the next phase of the project, the testing will be focused just on technical and programming aspects, considering the following tasks:

1. Test the feature mechanism for different types of conversation (groups, individuals, self) to check that all mentioned possibilities work well.
2. Test the feature selecting different periods and control that the data written to the XML formatted file is also delimited. Test for invalid periods as well.
3. Test each particular form of the chat export, including media/excluding media, HTML viewer, and check for correctness.
4. Test the mechanism to request storage permission for downloading.
5. Test the facility response for different storage locations.
6. Test the HTML-viewer with the different options with or without including media to examine if the information is displayed correctly.
7. Test the specific conversation components detached from each other in order to find defects.
8. Test the facility on different real and virtual devices with varying Android versions as well as various screen sizes, densities, features and check for defects.

As the implementation advances, the testing plan needs extensions in order to cover unpredictable defects. In order to have a bug-free feature, all feature requests shall follow the idea of workflow testing, although this project does not cover the automation of the tests. Here, the debugger assists as a tool for inspecting the bug in the code.

2.5 System Architecture

2.5.1 Use-Case diagram

The following diagram shows that "n" participants can hold a conversation. The Signal-Android app contains each unarchived conversation on the main screen in the app. In every conversation the user can select the option "chat export", which will ask the user for a location. The ZIP file is created, and if the user desires, they can open it afterwards.

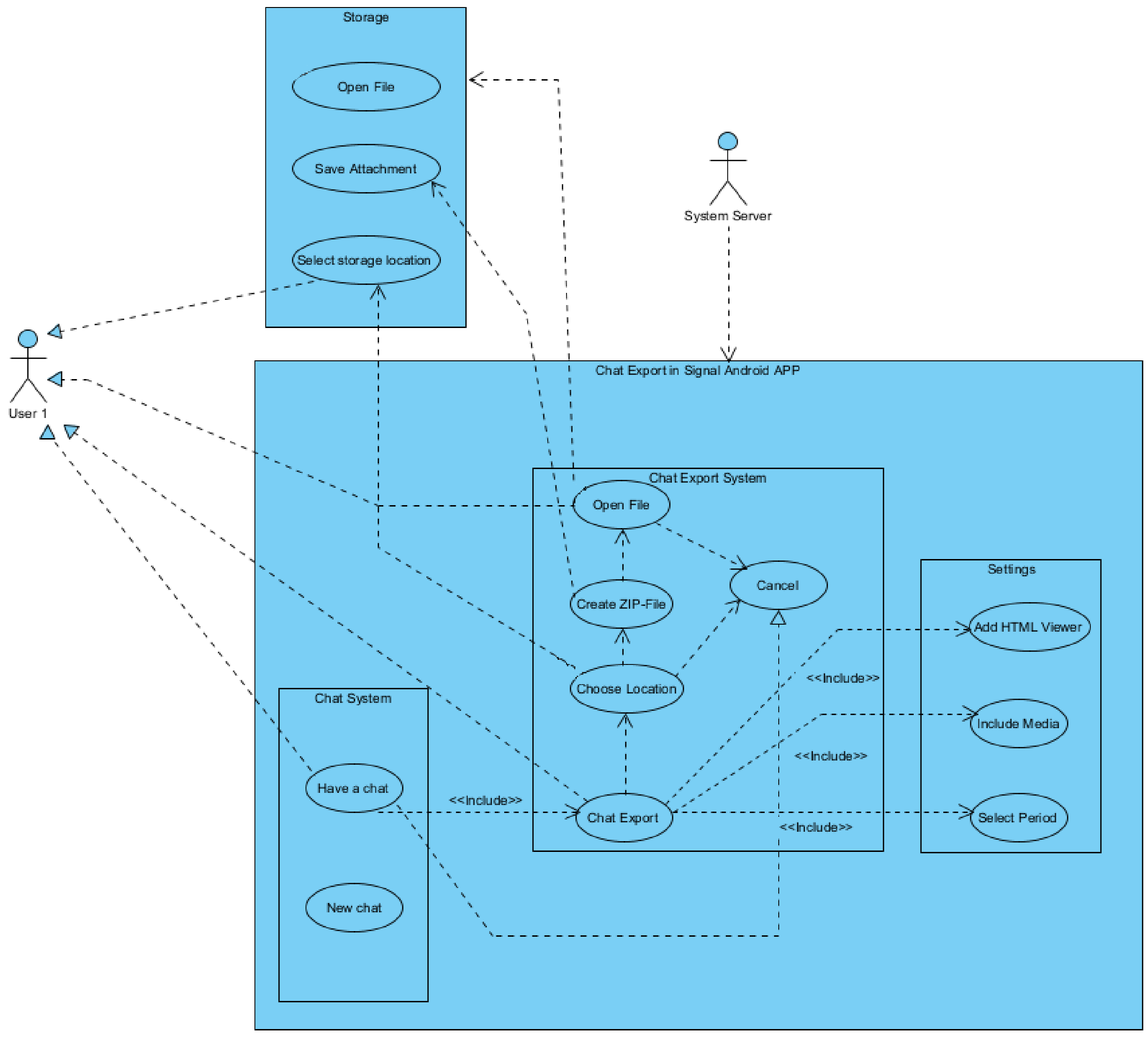


Figure 2.1: Use-Case-Diagram user's interactions for exporting a conversation

3 Results

3.1 Requirements

- Reliability: the designed feature should not fail and be available for all android devices that support the latest Signal version.
- Usability: easy to use, intuitive, adapted to the current version of the IM, reuse concepts of other options in Signal, counts with a step back option, an XML file to reuse when importing the chat content to this or other platforms, the viewer to display the content of the conversation.
- Performance: users should think that the interaction is uninterrupted, delays should be avoided, files should be attached to the ZIP file, the system should always respond about the state of this interaction.
- Maintainability: updating the source code to have the feature in the latest version of the Signal-Android app, testing functionality, and repairing failures.
- Security: exported data must be protected from third parties; data cannot be used for other purposes than exporting the chat content. The user selects where the resulting export file is downloaded on the user device.
- Signal requirements: for submitting a pull request, the contributor must sign the Contributor License Agreement (CLA) and preserve the Code Style Guidelines[31]

3.2 Android Project Structure

- `manifests`
  The "AndroidManifest.xml" contains the permissions and declarations of each activity. For the single chat export activity, it was declared as follows:

```
<activity
            android:name=".export.ChatExportActivity"
            android:configChanges="touchscreen|keyboard|keyboardHidden|
                orientation|screenLayout|screenSize"
            android:windowSoftInputMode="stateAlwaysHidden">
        </activity>
```

- `res/layout`
  contains the app layouts, the user interfaces in the app.

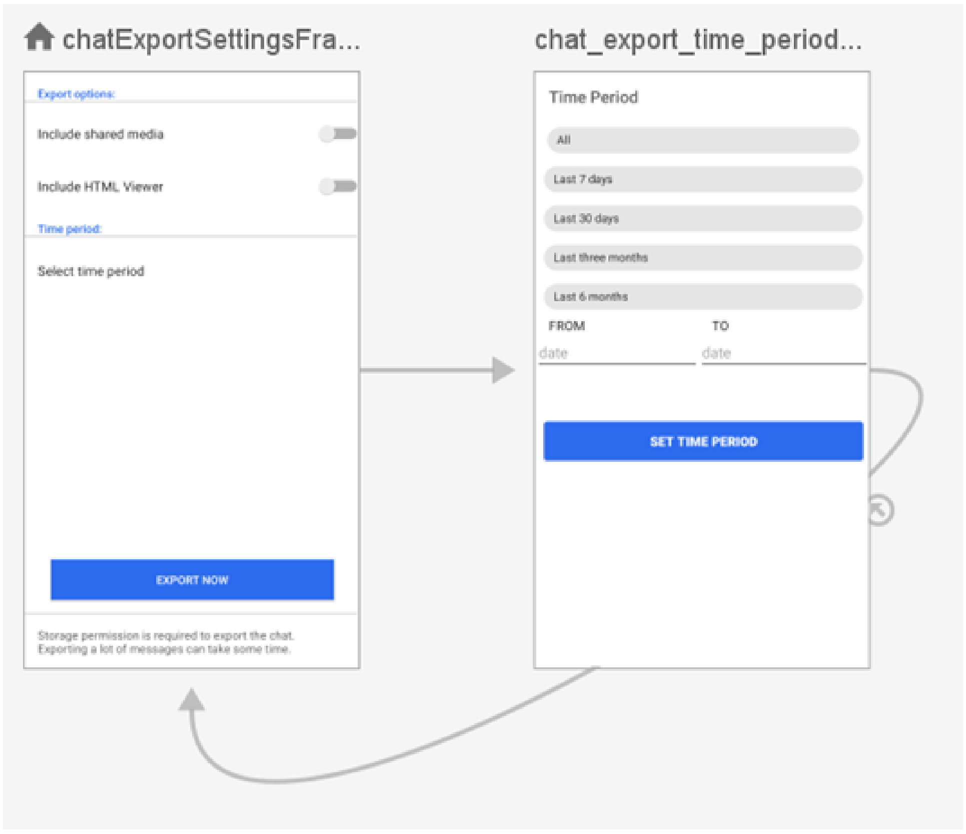
These three layouts manage the activity:

1. chat_export_activity.xml
2. chat_export_fragment.xml
3. fragment_chat_export_time_picker.xml

The main fragment *chat_export_fragment.xml* describes the different options for the settings included in a card-based layout, such as the option to include all interchanged media files that have been sent in the chat or the choice to include an HTML viewer which lets the user see its chats in a browser. Choosing a specific period to restrict the chat content export appears as a clickable text in this XML layout. Furthermore, there is a button at the bottom named *Export*, which the user touches to start the export. The other fragment *fragment_chat_export_time_picker.xml* supports selecting the specific times for the chat output, allowing the user to insert a period manually or selecting between many standard options.

- `res/navigation`
  contains different navigation graphs that describe the possible destinations and actions between layouts. Here the navigation component *chat_export_settings.xml* was created. It is used in the chat export activity to link the feature's settings with the time picker.

When the user touches the button, the Chat Export Activity starts. The activity *chat_export_activity.xml* includes two fragments. The navigation resources include the navigation component *chat_export_settings.xr* representing the destinations and actions in this activity.



Storage permission is required to export the chat.
Exporting a lot of messages can take some time.

Figure 3.1: Chat Export Activity Navigation Design

- `org/thoughtcrime/securesms/components`

  As a result of the observation phase, the conclusion was that the best way to introduce the solution is by adding an *Export Chat* button in the settings because it will just be an infrequently used action. It was implemented by attaching an option to the group and the individual layouts with the name of *Chat Export*, this selectable text has the same design as any other choice of the layout, like the same font, font size, font colour, alignment, background colours, even a "download icon" was reused for the functionality from the `\main\res\drawable\` folder. Apart from that, it is in an individual group which allows the user to easily distinguish it from other actions in this settings menu.

  Therefore, the following Android Kotlin class was modified:

  `.../settings/conversation/ConversationSettingsFragment.kt`

  In this case, this class works dynamically and takes the behaviour of the activity and the UI by using the following function:

```
fun clickPref(
    title: DSLSettingsText,
    summary: DSLSettingsText? = null,
    icon: DSLSettingsIcon? = null,
    isEnabled: Boolean = true,
    onClick: () -> Unit
  ) {
    val preference = ClickPreference(title, summary, icon, isEnabled,
       onClick)
    children.add(preference)
  }
```

Listing 1: Attach option to conversation settings

declared in the class *dsl.kt*.

Additionally, a new package called *export* was created in `org/thoughtcrime/securesms/`, containing all the JAVA-classes developed in this project:

ChatExportActivity First, the *ChatExportActivity.java* class was created in the `org/thoughtcrime/securesms/export` source set directory in the project explorer. For managing different views in the same activity, the use of "Fragments" has been considered.

ChatExportFragment The main fragment contains all options the user can select. Once the settings are selected, the chat export interaction can start. The following methods were implemented:

– This method gives functionality to all elements declared on this fragment layout, as well as fragment transactions through the ChatExportViewModel are covered here.

```
public void onViewCreated(@NonNull View view, @Nullable
   BundlesavedInstanceState)
```

Listing 2: Set up the initial state of the view for the chat export settings fragment

– The following method triggers an AlertDialog object, i.e. shows a pop-up screen asking the user to choose a location for storing the data to export. On selecting "choose location", the permission will be checked, and the gallery opened.

```
private void chooseSaveFileLocation()
```

Listing 3: Show dialog to select a location folder

– This other method checks for storage permissions.

```
public boolean isUserSelectionRequired (@NonNull Context context)
```

Listing 4: Check for storage permission

– To ask for accepting permissions if they are denied or still not accepted:

```
private void allowPermissionForFile()
```

Listing 5: Allow to accept permissions if they are not already accepted

– This method call creates an intent that opens the gallery and allows to choose a location in the storage. The result is the selected path, provided after touching the "allow access" button.

```
private void openGallery()
```

Listing 6: open the gallery for choosing a location in the storage

– To obtain the path for the chosen location.

```
public void onActivityResult (int requestCode, int resultCode, Intent
    data)
```

Listing 7: Return the selected storage path

– After that, the following method is called to start the export process. It instantiates the ChatFormatter and passes it to *getMediaIfNecessary(...)*.

```
private void getFormattedChat (ChatExportViewModel viewModel)
```

Listing 8: Obtain the chat data in a ChatFormatter object

– Check if there are messages/media to export, then call *handleSaveMedia(...)*.

```
void getMediaIfNecessary (@NonNull ChatFormatter exp,
    ChatExportViewModel viewModel, Uri uri)
```

Listing 9: Proccess media file if they were requested

– This method checks if storage permissions are ok and then calls *performSaveToDisk(...)*.

```
private void handleSaveMedia (
            Uri path, @NonNull Collection<ChatFormatter.MediaRecord>
                mediaRecords, HashMap<String, Uri> moreFiles,
            boolean currentSelectionViewer,
            String result)
```

Listing 10: Check storage permissions before attaching any file to the selected path

– This method creates an AsyncTask, where the ChatExportZipUtil includes media files declared as Attachments. Afterwards, a ChatExportZipUtil is instantiated and calls the method *startToExport(...)* to include the formatted data in the ZIP file.

```java
private static void performSaveToDisk (@NonNull Context context,
        Uri path,
        @NonNull Collection<ChatFormatter.MediaRecord> mediaRecords,
        HashMap<String, Uri> moreFiles,
        boolean hasViewer,
        String resultXML)
```

Listing 11: starts to attach the files to the zip file

ChatExportTimePickerFragment This second fragment appears to determine a period in which the conversation occurred to delimit the messages between these two dates.

– This method declares all elements included in this fragment layout.

```java
public View onCreateView(LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState)
```

Listing 12: Instantiate the elements of the chat export time picker fragment

– The following method describes the functionality of the layout elements. Besides, the ChatExportViewModel acts as the intermediary between the selected data here and in the main fragment.

```java
public void onViewCreated(@NonNull View view, @Nullable Bundle
    savedInstanceState)
```

Listing 13: Set up the initial state of the view for the chat export time picker fragment

– To check for a valid period selection.

```java
private boolean isValidPeriod ()
```

Listing 14: Check if the time period is valid

– To change the view elements which display the start export date and the end export date:

```java
private void setPeriodTime (Date untilDate, Date fromDate)
```

Listing 15: Change the time values on the text fields

– This method changes the period on the view model:

```java
private void changePeriodTime (Date untilDate, Date fromDate)
```

Listing 16: Update time period on the view model

ChatExportViewModel The view model acts as a mediator between these two fragments so that each fragment's time variables can be accessible. Besides, it preserves the settings in the last state. These are the variables that allow modifying the settings states:

```
private static final Boolean INITIAL_HTML_VIEWER_STATE,
   INITIAL_MEDIA_STATE = false;
private static final String INITIAL_TIME_PERIOD = "Default (whole chat)";
private final MutableLiveData<Optional<Date>> startDateControls,
   endDateControls = new MutableLiveData<>();
private final MutableLiveData<String> selectedTimePeriod = new
   MutableLiveData<>(INITIAL_TIME_PERIOD);
private final MutableLiveData<Boolean> enableIncludeMediaControls = new
   MutableLiveData<>(INITIAL_MEDIA_STATE);
private final MutableLiveData<Boolean> enableHTMLViewerControls = new
   MutableLiveData<>(INITIAL_HTML_VIEWER_STATE);
```

Listing 17: Variables of the view-model

The methods in this class are getters and setters for the above mentioned MutableLiveData variables.

ChatFormatter

– Constructor

```
ChatFormatter (@NonNull Context context, long threadId, Date fromDate
    , Date untilDate)
```

Listing 18: ChatFormatter constructor

In this class, the required conversation is extracted from the MmsSmsDatabase in the form of a cursor called *conversation*. Accordingly, the following method gets the cursor:

```
public Cursor getConversation(long threadId, long offset, long limit)
```

Listing 19: Cursor of the conversation

The following method calculates the *offset*:

```
public int getMessagePositionOnOrAfterTimestamp(long threadId, long
   timestamp)
```

Listing 20: Message position for a timestamp or after it

Here, the *limit* is an integer number extracted from the difference between the number of messages in the conversation at the selected end date and the number of messages at the selected start date. The method that counts the messages at a specific time is:

```java
public int getConversationCount(long threadId, long beforeTime)
```

Listing 21: Number of messages before a date

– Output

In order to create a document that contains an XML formatted file, the following imported objects are needed to obtain the desired result:

```java
TransformerFactory transformerFactory = TransformerFactory.newInstance
    ();
Transformer transformer = transformerFactory.newTransformer ();
transformer.setOutputProperty (OutputKeys.INDENT, "yes");
transformer.setOutputProperty (OutputKeys.METHOD, "xml");
transformer.setOutputProperty (OutputKeys.ENCODING, "UTF-8");
transformer.setOutputProperty ("{http://xml.apache.org/xslt}indent-
    amount", "4");
DOMSource source = new DOMSource (dom);

StringWriter outWriter = new StringWriter ();
StreamResult result = new StreamResult (outWriter);
transformer.transform (source, result);
```

Listing 22: Create the xml formatted document

In the first instance, the *threadId* variable determines the conversation that was selected. Therefore, the implicated participants' data are easily obtained with this *threadId* constant variable by accessing the thread and recipient databases and choosing the recipient for the thread id. Equally, other data are provided for the selected conversation, such as group details and a member list.

```java
ThreadDatabase threadDatabase = SignalDatabase.threads();
RecipientDatabase recipientDatabase = SignalDatabase.recipients();
Recipient recipient = threadDatabase.getRecipientForThreadId (threadId
    );
assert recipient != null;
RecipientDatabase.RecipientSettings settings = recipientDatabase.
    getRecipientSettings(recipient.getId ());
```

Listing 23: Obtain the recipient

Once the cursor is defined, a reader for this cursor is opened. The reader works as a pointer for each contained message, an object of the class "MessageRecord". In this object, the instantiated variables are the recipient id, the body, the recipient author, dates, thread id, type, reactions, timestamps, and other variables not taken into account for the project.

– The MediaRecord object was created to include all necessary information of a media file that could later be saved in a map *selectedMedia* with its unique attachment key.

```java
 private MediaRecord (@Nullable DatabaseAttachment attachment,
                              @NonNull RecipientId recipientId,
                              long threadId,
                              long date,
                              boolean outgoing)
```

Listing 24: MediaRecord object joins the necessary attachment information

Created mediaRecord objects are retrieved easily by calling the following method:

```java
public Map<String, MediaRecord> getAllMedia ()
```

Listing 25: set of all media in the selected conversation

– Methods which give structure to the XML document:

A simple conversation export allows structuring messages creating nested tags in the XML document for these elements and attributes.

```java
private Element addElement (Element parent, String tagname, String
    content)
```

Listing 26: Add XML Element to its parent

```java
private void addAttribute (Element parent, String tagname, String
    content)
```

Listing 27: Add XML Attribute to an element

```java
private Element addElement (Element parent, String tagname)
```

Listing 28: Add XML Element to its parent without content

The highlighted elements below summarize the categories of data that can be found in a single conversation message and describe the data location. However, these different elements are not yet easily located or entirely usable, as explained below, like, for example, the path of an attachment. If media content and HTML viewer are included in the ZIP, these attachments are tracked so that the content can be retrieved, given that the attachment path changes to point to a path in the created zip file structure.

Mentions are found by means of using the MentionUtil class, specifically the method:

```java
public static @NonNull UpdatedBodyAndMentions
    updateBodyAndMentionsWithDisplayNames(@NonNull Context context,
    @NonNull CharSequence body, @NonNull List<Mention> mentions)
```

Listing 29: Obtain the right mention

and

```java
public @NonNull List<Mention> getMentions()
```

Listing 30: Obtain list of mentions from a message

For each Mention object, the following data to be preserved are selected: mentioned person's id, name, and some integers that indicate the position of the message where this mention starts and how many characters it has (start value and length).

Shared contacts are stored in the MmsMessageRecord object as a list of Contact objects. From this Contact item, the following data are obtained: contact name and other optional data objects such as email addresses (Email), phone numbers (Phone) or postal addresses (PostalAddress).

```java
public @NonNull List<Contact> getSharedContacts()
```

Listing 31: Obtain a shared contact from a message

Link previews likewise are found as a list of LinkPreview objects in the MmsMessageRecord object. Accordingly, a link preview in a message keeps the following variables: title, URL, text description, date and optional attachments.

```java
public @NonNull List<LinkPreview> getLinkPreviews()
```

Listing 32: Obtain the link preview from a message

Quotes appear as a list of Quote objects for the message record. Usually, a message record contains just one quote. Under this presumption, users can attach a single pre-selected message when they are writing a new reply to this previous message. Consequently, the necessary data are quote id, author name, quote text, quote attachment and timestamp.

```java
public @Nullable Quote getQuote()
```

Listing 33: Obtain the quote from a message

Any media content shared but stickers are considered as Attachments and included in the attachments database. Attachments store the following data about the file: attachment id, key, filename, size, content type, width, height and upload timestamp.

Additionally, some boolean variables which indicate its file type, such as voice note, borderless, video GIF, quote (for quote attachments), are equally included.

```
@NonNull private List<DatabaseAttachment> getDatabaseAttachments(
    @NonNull MessageRecord record)
```

Listing 34: Obtain the list of attachments of a message

Stickers are also attachments, but have null as attachment key, so they must be considered independently from other Attachment objects. To identify them the object Sticker was chosen by selecting the pack key string attached to the sticker id.

Reactions are part of the message record. A reaction object is defined by the following variables: author id, author name, timestamp and emoji.

```
public @NonNull List<ReactionRecord> getReactions()
```

Listing 35: Obtain the reactions to a message

ChatExportZipUtil

– Constructor

```
public ChatExportZipUtil (Context context, Uri storagePath, int count
    , long threadId) throws IOException, NoExternalStorageException
```

Listing 36: ZIP constructor

- To proceed to create the ZIP file, the next method must be called:

```
protected void startToExport(Context context, boolean hasViewer,
    String data) throws IOException
```

Listing 37: Method which calls other methods for creating the ZIP file

- The given storage path represents the selected directory tree. In this path, a DocumentFile with the conversation name is created:

```
private DocumentFile instantiateZipFile()
```

Listing 38: Method which creates the ZIP file

- A ZipOutputStream is instantiated after creating the ZIP file by using its URI path in the following method:

```
public ZipOutputStream getZipOutputStream() throws IOException
```

Listing 39: Create the ZipOutputStream object

- Next, the XML files and desired complements are attached to the ZIP file.

```java
public void addFile (String name, InputStream data)
```

Listing 40: Files can be attached to the ZIP-file

– Attachments:

```java
public Attachment(Uri uri, String contentType, long date, long size)
```

Listing 41: Attachment object for media included in the ZIP file

– ProgressDialogAsyncTask<ChatExportZipUtil.Attachment, Void, Pair<Integer, String>>:

The ChatExportZipUtil implements this extension of an AsyncTask<Input, Progress, Output>, in which the inputs are the ZIP attachments, the progress is calculated in the background, and the output is a Pair<Integer, String> in which the integer is assigned to an exit status of the task. This functionality works by using the following method:

```java
@Override
    protected Pair<Integer, String> doInBackground(ChatExportZipUtil.
        Attachment... attachments)
```

Listing 42: Background task for ZIP file generation

After performing the media attaching process, the following method is called to evaluate the result of the previous method, in this case, if all the uploading was successful:

```java
@Override
    protected void onPostExecute(final Pair<Integer, String> result)
        onPostExecute()
```

Listing 43: Task for attaching files to the ZIP file

When all child files are attached to the ZIP file, the AsyncTask proceeds to close the ZipOutputStream by calling the inherited method close() here:

```java
public void closeZip () throws IOException
```

Listing 44: Close ZIP file

– Methods:

* Create the name of the ZIP file:

```java
private String createFileName ()
```

Listing 45: Creates filename for the ZIP file

* Attach the viewer to the ZIP file

```
private void includeHTMLViewerToZip (Context context)
```

Listing 46: Include HTML viewer to the ZIP file

* Retrieve the path where a media file should be saved:

```
public @NonNull
static String getMediaStoreContentPathForType (@NonNull String
    contentType)
```

Listing 47: Get a path in the ZIP for a media file

* Name the attachments and create their paths:

```
private String createOutputPath(@NonNull String outputUri, @NonNull
    String fileName)
     throws IOException
```

Listing 48: Create output path for attachments

* Create a name for the file, which must be included in the XML document to link the extracted file to the media element:

```
public static String generateOutputFileName (@NonNull String
    contentType, long timestamp, @NonNull String uriPathSegment)
```

Listing 49: Create output file name for attachments

* This method adds the given Attachment to the ZIP file:

```
@NonNull
private String saveAttachment (Context context, Attachment attachment)
      throws IOException
```

Listing 50: Add attachment to the ZIP file

Depending on the attached file, the InputStream can be created either directly from the String data, by opening the InputStream calling the AssetManager in order to append the HTML-Viewer files or by calling the following method when the file to attach is an Attachment:

```
public static InputStream getAttachmentStream(@NonNull Context context
    , @NonNull Uri uri)
     throws IOException
```

Listing 51: InputStream for attachments

* Retrieve the content type of an attachment

```java
private String getContentType (@NonNull Attachment attachment)
```

Listing 52: Content type of an attachment

* Add the given content of the XML document to the ZIP file

```java
private void addXMLFile (@NonNull String data)
```

Listing 53: Add XML file to the ZIP package

* Communicate if the ZIP File was created successfully:

```java
private void createFinishDialog (Context context)
```

Listing 54: Show chat export finish dialog

* Create a new intent to show the selected storage directory:

```java
private void openDirectory (Uri storageDirectoryUri)
```

Listing 55: Open the directory where the ZIP-file was saved

- `assets` provide extra files that the package .apk includes when it is generated.

  Here the HTML-viewer tool has been added, specifically these three files:

  1. `chatexport.htmlviewer/jquery-3.6.0.min.js`
  2. `chatexport.htmlviewer/signal-app.png`
  3. `chatexport.htmlviewer/viewer.html`

  Another file was created there in order to provide the XSD schema for the chat.xml

  - `chatexport.xsdschema/export_chat_xml_schema.xsd`

3.3 XML File

3.3.1 XML File Use-Case Diagram View

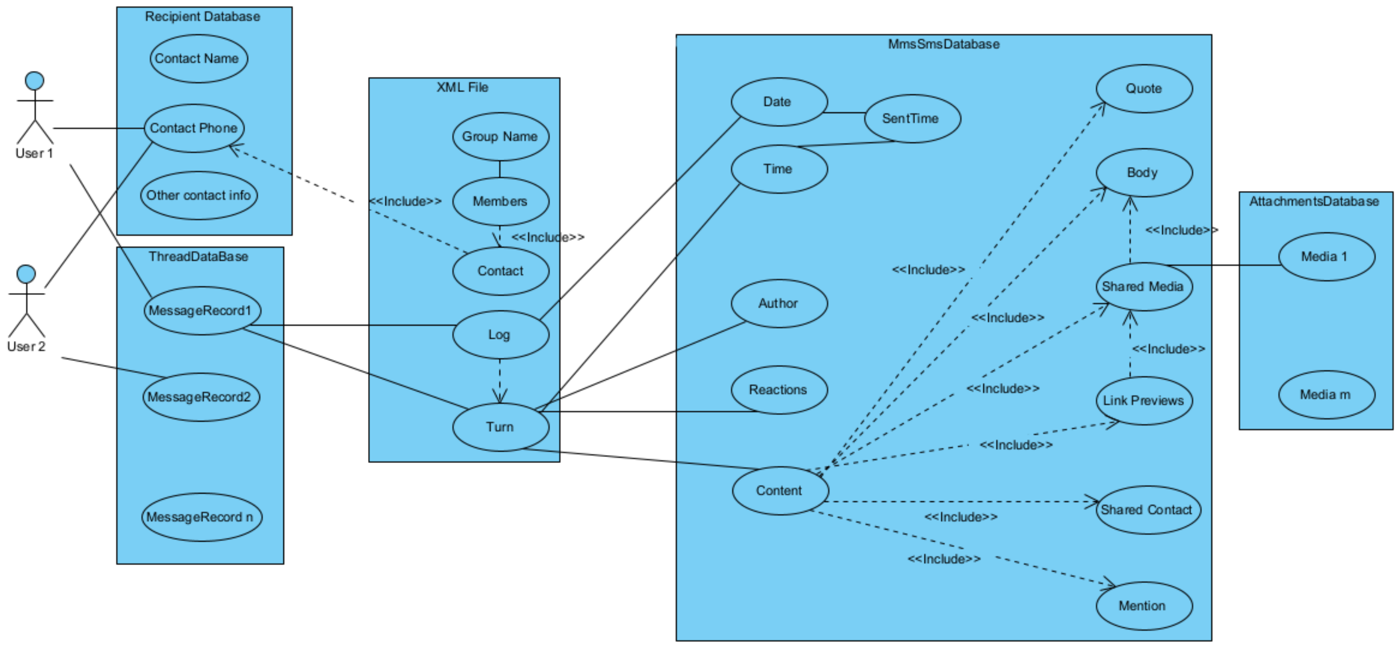


Figure 3.2: Use-Case-Diagramm for the XML Document

3.3.2 XML File Structure

Here are some examples of the XSD-schema elements for the different blocks in which the resulting XML document is divided.

```
<xs:complexType name="contactType">
   <xs:annotation>
      <xs:documentation>
         Personal data of the individual
      </xs:documentation>
   </xs:annotation>
   <xs:all>
      <xs:element name="profile_name" type="xs:string" minOccurs="0"/>
      <xs:element name="relation" type="xs:string" minOccurs="0"/>
      <xs:element name="trackingid" type="xs:string" minOccurs="0"/>
      <xs:element name="teamid" type="xs:string" minOccurs="0"/>
      <xs:element name="about" type="xs:string" minOccurs="0"/>
      <xs:element name="phone" type="xs:string" minOccurs="0"/>
      <xs:element name="email" type="xs:string" minOccurs="0"/>
   </xs:all>
   <xs:attribute name="id" type="xs:string"/>
   <xs:attribute name="name" type="xs:string"/>
```

```
</xs:complexType>
```

Listing 56: Contact type definition in the XSD-Schema

```
<xs:complexType name="logListType">
   <xs:annotation>
      <xs:documentation>
         List of entries for the found messages
      </xs:documentation>
   </xs:annotation>
      <xs:sequence>
         <xs:element name="turn" type="turnListType" minOccurs="0" maxOccurs="
            unbounded"/>
      </xs:sequence>
   <xs:attribute name="date" type="xs:string"/>
</xs:complexType>


<xs:complexType name="turnListType">
   <xs:annotation>
      <xs:documentation>
         Specifies the author and includes the output
      </xs:documentation>
   </xs:annotation>
   <xs:all>
      <xs:element name="message" type="messageType"/>
   </xs:all>
   <xs:attribute name="author" type="xs:string"/>
</xs:complexType>

<xs:complexType name="messageType">
   <xs:annotation>
      <xs:documentation>
         Specifies the output details
      </xs:documentation>
   </xs:annotation>
   <xs:all>
      <xs:element name="body" type="bodyType"/>
   </xs:all>
   <xs:attribute name="id" type="xs:string"/>
   <xs:attribute name="time" type="xs:string"/>
   <xs:attribute name="is_deleted" type="xs:boolean"/>
   <xs:attribute name="status" type="xs:string"/>
   <xs:attribute name="mismatched_identities" type="xs:string"/>
   <xs:attribute name="expires_in" type="xs:string"/>
</xs:complexType>

<xs:complexType name="bodyType">
   <xs:annotation>
```

```
      <xs:documentation>
        Specifies the output type of content
      </xs:documentation>
    </xs:annotation>
    <xs:all>
      <xs:element name="text" type="xs:string" minOccurs="0"/>
      <xs:element name="shared_contact" type="sharedContactType" minOccurs="0"/>
      <xs:element name="mention" type="mentionType" minOccurs="0"/>
      <xs:element name="quote" type="quoteType" minOccurs="0"/>
      <xs:element name="link" type="linkPreviewType" minOccurs="0"/>
      <xs:element name="media_content" type="mediaContentType" minOccurs="0"/>
      <xs:element name="reactions" type="reactionsType" minOccurs="0"/>
    </xs:all>
  </xs:complexType>

  <xs:complexType name="reactionsType">
    <xs:annotation>
      <xs:documentation>
        Registers reactions
      </xs:documentation>
    </xs:annotation>
    <xs:all>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="time" type="xs:string"/>
      <xs:element name="emogi" type="xs:string"/>
    </xs:all>
    <xs:attribute name="author_id" type="xs:string"/>
  </xs:complexType>
```

Listing 57: Example of different types for a conversation in the XSD-Schema

3.3.3 Comparison of the XML output files for Signal and for Wire

The first view in the XSD-schema published on the Wire GitHub[22] shows that its chat export data structure was detailed. The goal of this structure was to make further processing easier like for importing chats to other apps in the future by using this same schema.
The Wire XSD schema, which describes the structure of the XML document of the Wire solution for single chat export was a reference to start with. At the beginning of the project, it was puzzling to identify which parts both solutions have in common and how to integrate the conversation export data of Signal in this other schema. Accordingly, an XML file with a similar structure was created step by step containing the element tags observed in the analysed objects' functions — besides, the data found in the Signal Databases attaches additional tags to the XML file. The resulting solution contains general elements, which are also contained in the Wire solution, but many of these elements were structured differently.

The following element types comparisons inspect the essential differences between the two solutions:

Comparison of conversation element types In Wire the following elements define conversations: id, remoteid, creator, convType, verified, userroles, messages, name, teamid, access, accessrole and link. In the XML file in Signal, the elements that describe conversations are not defined as in Wire: threadid, selected_time_period, a list of logs (list of messages sorted by date as an attribute). All other elements are not included, because they were not available on the database, private (as userrole) or nonexistent.

Comparison of message element types

- In the case of the messages element types, in Wire each message element can be defined with the following data: id, msg_type, userid, state, time, contents, protos, local_time, first_message, error, members, recipient, edit_time, ephemeral, expiry_time, duration, asset_id, quote.

  While in Signal, a message type can be defined with the following elements: author, author_id, id, is_deleted, expires_in, time, local_time, status, msg_type(which differs with the declared on Wire), body, mismatched_identities, reactions.

- The defined object protoType in the Wire XSD-schema includes a mix of data that includes the following references: to assets, calling, cleared, clientaction, deleted, edited, external, hidden, image, knock, lastread, reaction, text, location, confirmation, availability, composite, buttonaction, buttonactionconfirmation, datatransfer and debug_data.

  Although many of these elements are found in the Signal database, the resulting solution does not use this protoType element, but they are included as media_content elements. The elements which describe the protoType are defined by other names in Signal, are private or not existent (as availability, clientaction, verified, access_type, accessrole_type, conversation_type_type, conversation_role_type, message_part_type).

- The description of the XSD schema published on the Wire GitHub describes the message element types with many redundant data that are not included in the MessageRecord object like first_message, members, asset_id or quote. On the other hand, there is another class (MessageRecordUtil) that was used to define the content of messages (body), which includes: mms, sharedcontact(not existent in the Wire XSD-Schema as an element), linkpreview, quote, mentions, media_content, sticker, location, audio, captionlessmms, borderless, document, viewoncemsg, extratext, text. These other elements are not defined together, in the same element type, in the Wire XSD schema, but included as message_text_type, message_composite_type or the protoType mentioned above.

Comparison of media content element types On Signal media content refers to an object with id, filename, fast_preflight_id, key, size and the following attachment data types:

- audio (content_type, name, voice_note, caption, duration_sec)
- video (content_type, name, width, height, caption, video_edited, video_trim, duration_sec)
- location (description)
- image (content_type, name, width, height, caption)
- document (content_type, name)
- sticker (content_type, id, is_borderless, emoji, describe_contents, name)
- unknow (content_type, name)

On Wire, the assetType is described with mime_type, name, size, filepath and the following asset info:

- audio (duration_in_millis, normalized_loudness)
- image (width, height, tag)
- video (width, height, duration_in_millis)

Comparison of Quotes element types In the presented solution, quotes are defined by their id, author, quote_text, is_missing, attachment and timestamp, while on Wire's solution, they are described by using a unique boolean.

Comparison of LinkPreview element types Looking at Wire's XSD schema, the linkpreviewType is defined by permanentUrl, url, title, summary, urloffset, image and tweet (author, username). In Signal link preview elements are pretty similar: title, url, description, date, link_preview(id, filename, content_path, content_type).

Comparison of Mentions element types For Wire, mentions are described with start, length, userid. In this project, mentions are defined with id, name, start, length.

3.4 ZIP directory structure

The ZIP file will consist of the following data:

- chat.xml (always included)
- Media (attached directory if media content was selected and any media content is present)
  - Signal Audios
  - Signal Documents
  - Signal GIFs
  - Signal Images
  - Signal Stickers
  - Signal Videos
- viewer.html + jquery-3.6.0.min.js + signal-app.png

  (attached when the option HTML viewer is selected)

3.5 HTML-Viewer Structure

The HTML file adds some properties for the HTML-Elements, specified between tags. Defined with the HTML *style* attribute, the image of the site and the different elements in there are declared. Furthermore, it includes the jQuery library in the HTML "script" attribute, which is relatively light, for extracting the content of the XML file—and putting it on the web page. These last attachments are compressed in the resulting ZIP file and when extracted, have a file size of 116KB.

The chosen layout for the chat display is divided into three parts as follows:

- Header
  It is placed at the top of the page. It includes the icon of Signal, the title of the site and a form for selecting the XML file included in the ZIP, and by clicking the button, the content will be displayed in the content part of the page.
- Content
  The content is placed between the header and footer and split up into two columns. The first column shows the list of members who participate in the conversation and some personal information about them. The second column would include the selected conversation and its media if selected by the user. Here single messages are displayed as indented paragraphs. The

date of the records is shown just once as a new message is recorded for this date. Same day messages are highlighted with the author's name, the time the message was sent, the content of the message and reactions if they exist.

- Footer
  Here some hyperlinks which refer to the company are attached:

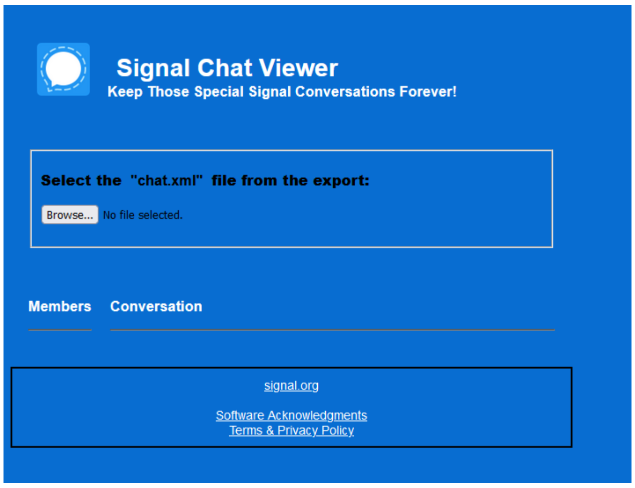


Figure 3.3: HTML Viewer

3.6 Layout Design and Functionality

1. Open the conversation the user wants to export and head over to the chat settings by clicking on the name of the chat3.4.

2. Choose “Export conversation” and enter the preferences for the chat export. Optionally include all media, an HTML viewer or select a period to delimit the chat extract3.5.

3. Change the dates to select an extract of the chat3.6.

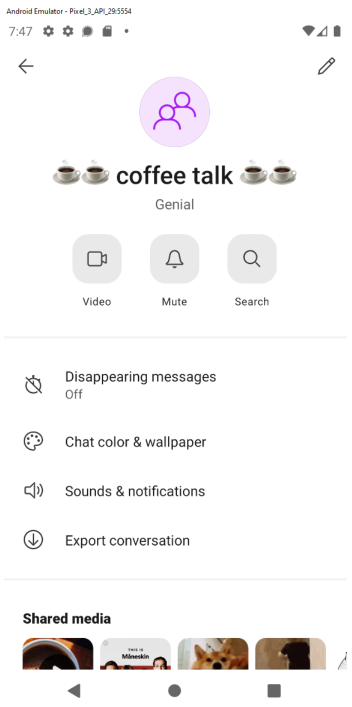
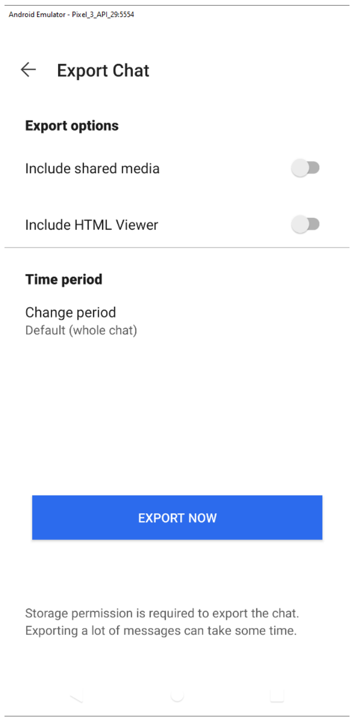


Figure 3.4: Conversation Settings



Storage permission is required to export the chat.
Exporting a lot of messages can take some time.

Figure 3.5: Chat Export Setting Main Layout

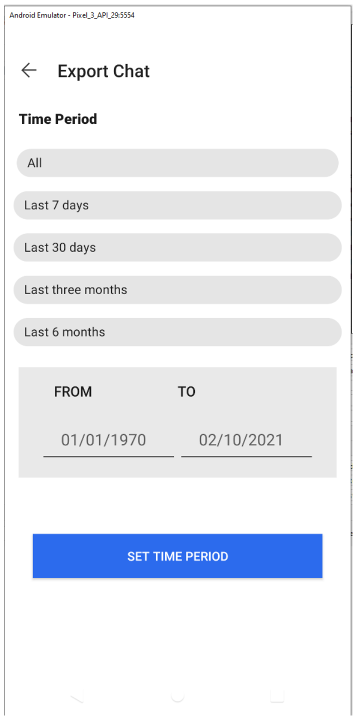
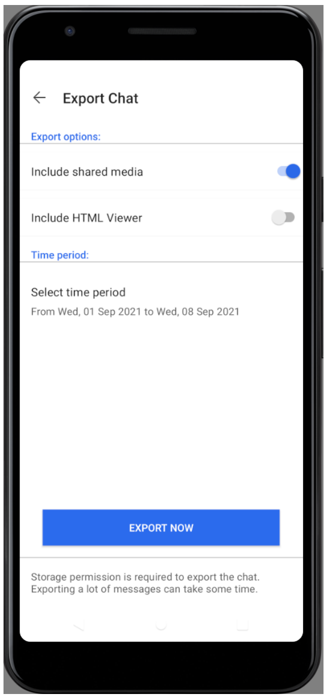


Figure 3.6: Chat Export Period Settings

4. Press the EXPORT NOW button to export the chat content3.7 and select the download location.

5. Before the export starts, a pop-up message asks for the selection of a location3.8.

6. The file manager allows the user to select any storage location on the device to store the resulting ZIP file3.9.



Export Chat

Export options:

Include shared media

Include HTML Viewer

Time period:

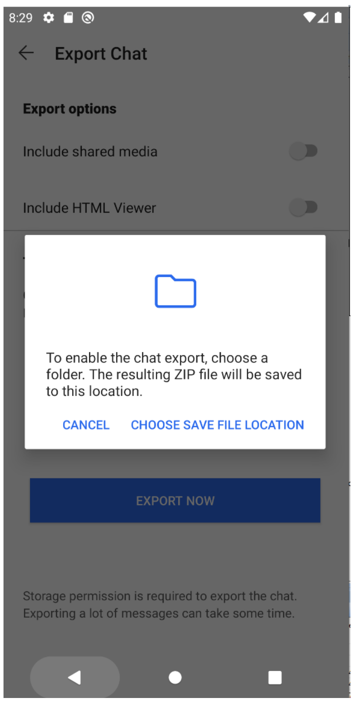
Select time period

From Wed, 01 Sep 2021 to Wed, 08 Sep 2021

EXPORT NOW

Storage permission is required to export the chat. Exporting a lot of messages can take some time.

Figure 3.7: Chat Export with Media Files option selected



8:29

Export Chat

Export options

Include shared media

Include HTML Viewer

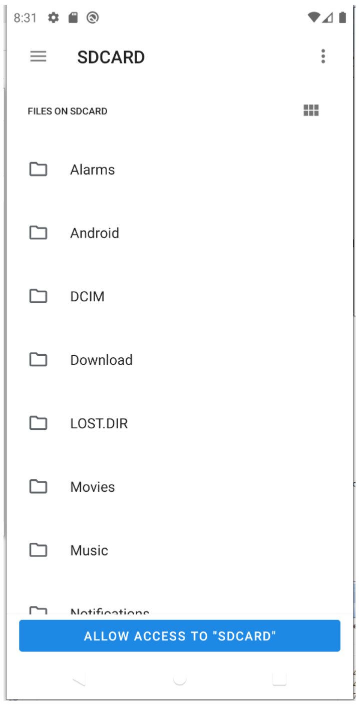
To enable the chat export, choose a folder. The resulting ZIP file will be saved to this location.

CANCEL CHOOSE SAVE FILE LOCATION

EXPORT NOW

Storage permission is required to export the chat. Exporting a lot of messages can take some time.

Figure 3.8: Choose location



8:31

SDCARD

FILES ON SDCARD

Alarms

Android

DCIM

Download

LOST.DIR

Movies

Music

Notifications

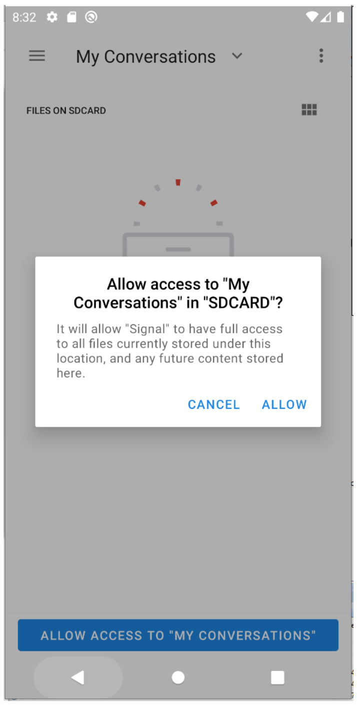
ALLOW ACCESS TO "SDCARD"

Figure 3.9: Select a folder to save the ZIP-file

7. Press the ALLOW button to grant the app access3.10.

8. After completing the export, a message appears, allowing the user to open the download directory or come back to the previous setting screen3.11.

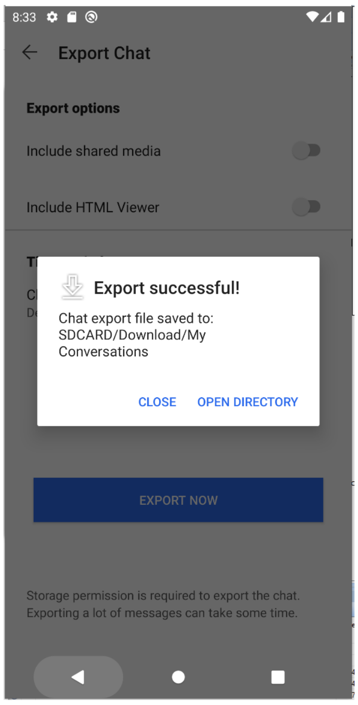
9. The selected directory with the ZIP file is opened3.12.



Allow access to "My Conversations" in "SDCARD"?

It will allow "Signal" to have full access to all files currently stored under this location, and any future content stored here.

Figure 3.10: Allow the access to the selected location



Include shared media

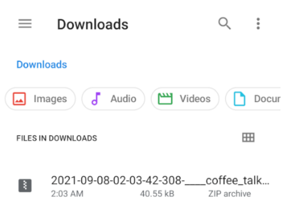
Include HTML Viewer

Export successful!

Chat export file saved to: SDCARD/Download/My Conversations

Storage permission is required to export the chat. Exporting a lot of messages can take some time.

Figure 3.11: Chat Export results successful

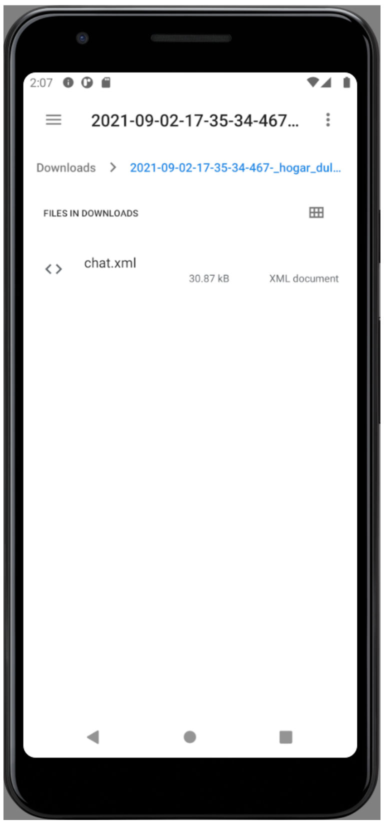


2021-09-08-02-03-42-308-___coffee_talk...
2:03 AM 40.55 kB ZIP archive

Figure 3.12: ZIP file appears in the storage

10. Look at the chat content offline in the browser or personal device.

    - Open the ZIP file to see the chat export generated file. If the HTML viewer was included, click on it to open the conversation in a browser3.13, 3.14, 3.15.



2:07

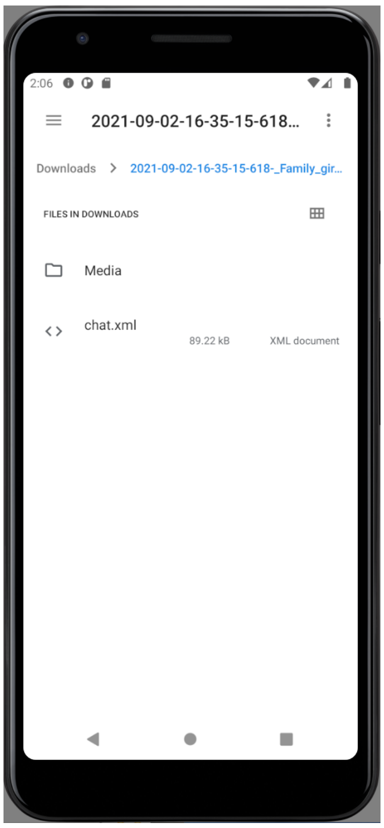
2021-09-02-17-35-34-467...

Downloads > 2021-09-02-17-35-34-467-_hogar_dul...

FILES IN DOWNLOADS

chat.xml 30.87 kB XML document

Figure 3.13: ZIP content - just conversation



2:06

2021-09-02-16-35-15-618...

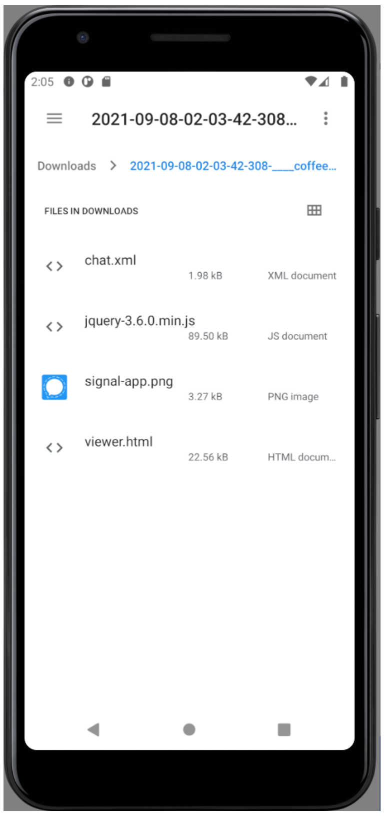
Downloads > 2021-09-02-16-35-15-618-_Family_gir...

FILES IN DOWNLOADS

Media

chat.xml 89.22 kB XML document

Figure 3.14: ZIP content - chat and media



2:05

2021-09-08-02-03-42-308...

Downloads > 2021-09-08-02-03-42-308-___coffee...

FILES IN DOWNLOADS

chat.xml 1.98 kB XML document

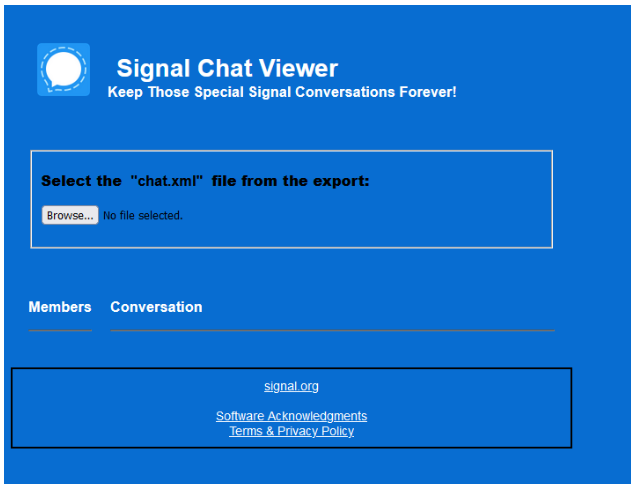
jquery-3.6.0.min.js 89.50 kB JS document

signal-app.png 3.27 kB PNG image

viewer.html 22.56 kB HTML docum...

Figure 3.15: ZIP content - chat and viewer, no media

- In the HTML viewer: Select the *chat.xml* file of the ZIP file to see the exported chat3.16. Users can see all the interchanged messages in XML format. Additionally, if media were included in the export preferences, they are also included in the ZIP file.



Signal Chat Viewer

Keep Those Special Signal Conversations Forever!

Select the "chat.xml" file from the export:

Browse... No file selected.

Members Conversation

signal.org

Software Acknowledgments

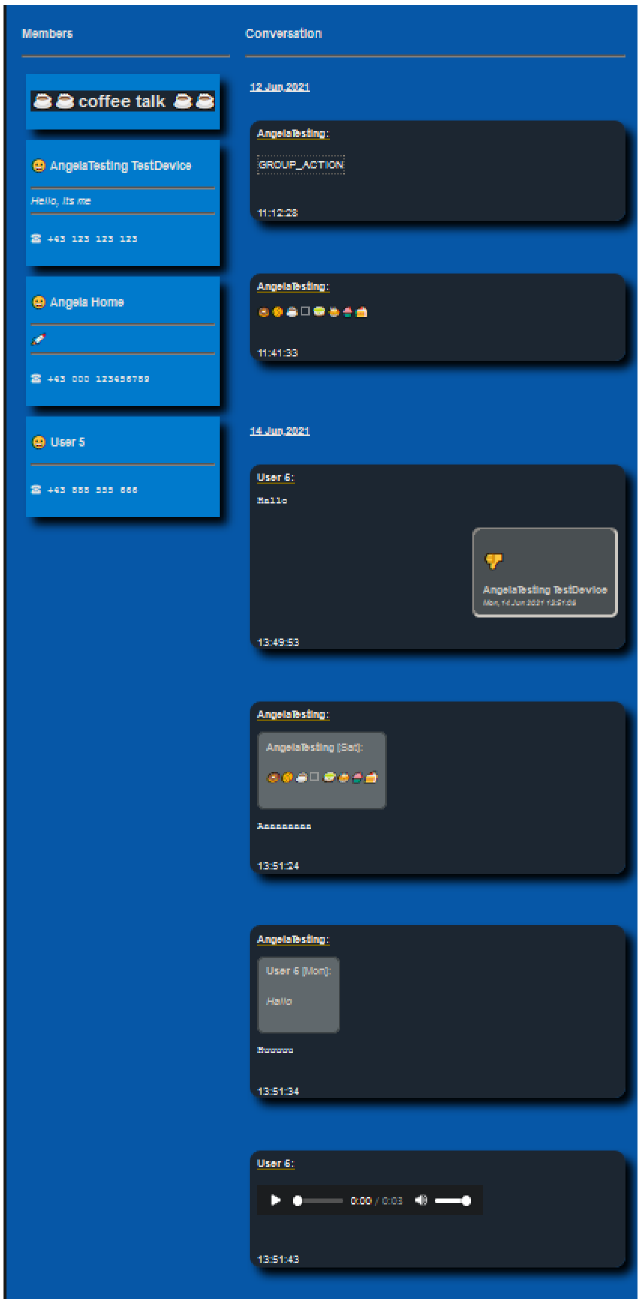
Terms & Privacy Policy

Figure 3.16: HTML Viewer

11. Watch the exported chat extract3.17:

    a) On the left side of the page, the *Members* column is found. For groups, it shows its group name. Next, the participants who are part of the conversation appear including some other personal data they have authorized to be public.

    b) On the right side, the *Conversation* column is found. The chat messages are displayed in ascending order sorted by date and time. Each message includes the author, content, sent time and possible reactions of the other participants.



Members

☕☕ coffee talk ☕☕

AngelaTesting TestDevice

Hello, its me

☎ +43 123 123 123

Angela Home

☎ +43 000 123456789

User 5

☎ +43 555 555 666

Conversation

12 Jun 2021

AngelaTesting:

GROUP_ACTION

11:12:28

AngelaTesting:

11:41:33

14 Jun 2021

User 6:

Hallo

AngelaTesting TestDevice
Mon, 14 Jun 2021 13:51:08

13:49:53

AngelaTesting:

AngelaTesting [Sat]:

Aaaaaaaaa

13:51:24

AngelaTesting:

User 6 [Mon]:

Hallo

Hhhhhhh

13:51:34

User 6:

0:00 / 0:03

13:51:43

Figure 3.17: Example of an exported conversation extract on the HTML Viewer

3.7 Maintenance

The maintenance consists of periodical updates with the upstream of the official Signal-Android GitHub source code. During the project development, some fundamental changes took place:

- After some other project contributors changed/deleted basic classes and introduced new ones where the code language changed to Kotlin, updates in the project code were necessary to adapt the feature to the new changes.
- Perform changes in the app's design line, adapting the functionality to its new design line.
- Improving and refining the quality of the output
- Testing is done regularly after each change in order to avoid failures.
- Cleaning code

4 Conclusion

4.1 Summary

This approach has given a solution for exporting single conversations from the Android Signal app and saving a package with its content directly into the android device storage. This work has involved several phases of a project, from planning through implementing to maintenance. On balance, the project creator has obtained a real insight into the world of application development and data security. The last step considers presenting the results that could totally or partially fit into the current Signal Android app.
This writing involved the author as a contributor in a professional android application, and it has shown how a service development works in real life.
In conclusion, this dissertation gives essential lessons concerning software development in a professional environment.

4.2 Problems

Firstly, numerous complications arose when unexpected results appeared either when building the app or in general because of the project size and its complexity. Besides, external app code changes resulted in wasted efforts that involved re-examining and re-adapting some fundamental parts of the code.
Additionally, due to the project volume, the compilation time delayed the whole work.

4.3 Future Work

To conclude, some questions have come up that can be considered an expansion of this project.

- Shall it be considered to pass the Java code to Kotlin as it is happening already in the Signal app for Android?
- Who will maintain the feature working when the application updates? How often should it be reviewed?
- Shall the transferred chat be imported in this or other apps? A common XSD-Schema for different IM apps would make this step possible.
- Could it be interesting/possible to filter messages under established preferences, i.e., date, person, or words?
- Is there any privacy dilemma to export the chat without all participants consent? A possible

solution would be to add an extra option that allows a user to deny exporting certain chats in which the user participates.

- Shall it be considered to insert a shadow password to the downloaded file for protection or an invisible watermark that allows traceability?

Listings

References

[1] Clegg, N.: *Europa darf sich nicht gegen die kreative Nutzung von Daten wenden.* (accessed: 30.10.2021). https://www.welt.de/politik/ausland/article225387677/Nick-Clegg-Europa-darf-sich-nicht-gegen-die-kreative-Nutzung-von-Daten-wenden.html.

[2] Urquhart, L. ; Sailaja, N., and McAuley, D.: „Realising the right to data portability for the domestic Internet of things“. In: *Personal and Ubiquitous Computing* 22.2 (2018), pp. 317–332.

[3] *Charting a Way Forward on Privacy and Data Portability.* (accessed: 30.10.2021). https://about.fb.com/wp-content/uploads/2020/02/data-portability-privacy-white-paper.pdf.

[4] *Facebook Data Policy.* (accessed: 24.11.2021). https://www.facebook.com/privacy/explanation/.

[5] *Does Facebook sell my information?* (accessed: 24.11.2021). www.facebook.com/help/152637448140583/?helpref=uf_share.

[6] *What Is End-to-End Encryption, and Why Does It Matter?* https://www.howtogeek.com/711656/what-is-end-to-end-encryption-and-why-does-it-matter/. (accessed: 18.10.2021).

[7] *Facebook Says Encrypting Messenger by Default Will Take Years.* https://www.wired.com/story/facebook-messenger-end-to-end-encryption-default/. (accessed: 28.09.2021).

[8] Steve Satterfield, D. o. P. and Policy, P.: *Transfer Your Facebook Posts and Notes With Our Expanded Data Portability Tool.* (accessed: 30.10.2021). https://about.fb.com/news/2021/04/transfer-your-facebook-posts-and-notes-with-our-expanded-data-portability-tool/.

[9] *Download Facebook Messenger Chat History.* https://www.techspotty.com/download-facebook-messenger-chat-history-how-to/. (accessed: 28.10.2021).

[10] Gail Kent, M. P. D.: *Messenger Policy Workshop: Future of Private Messaging.* (accessed: 30.10.2021). https://about.fb.com/news/2021/04/messenger-policy-workshop-future-of-private-messaging/.

[11] *About WhatsApp.* https://www.whatsapp.com/about/. (accessed: 08.10.2021).

[12] *What Whatsapp collect and share with Facebook.* (accessed: 28.11.2021). https://www.androidauthority.com/whatsapp-privacy-1189873/.

[13] *Whatsapp privacy concerns.* (accessed: 28.11.2021). https://www.wired.com/story/how-to-boost-whatsapps-privacy-and-better-protect-your-data/.

[14] *WhatsApp Encryption Overview.* https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. (accessed: 08.10.2021).

[15] *WhatsApp Help Center - How to Save Your Chat History.* https://faq.whatsapp.com/android/chats/how-to-save-your-chat-history/?lang=en. (accessed: 08.10.2021).

[16] *Telegram Privacy.* (accessed: 25.11.2021). https://telegram.org/privacy.

[17] *Telegram Messenger.* https://play.google.com/store/apps/details?id=org.telegram.messenger. (accessed: 08.10.2021).

[18] *Telegram MTProto Mobile Protocol.* https://core.telegram.org/mtproto. (accessed: 08.09.2021).

[19] *Wire Messaging.* (accessed: 24.11.2021). https://wire.com/en/product/messaging/.

[20] *Wire Conferencing.* (accessed: 24.11.2021). https://wire.com/en/blog/upgrades-to-wire-conferencing/.

[21] *Wire - Back up your conversation history.* (accessed: 24.11.2021). https://support.wire.com/hc/en-us/articles/360000824805-Back-up-your-conversation-history.

[22] *Wire - Add the possibility to export conversations as zip.* (accessed: 24.11.2021). https://github.com/kanashius/wire-android/commit/b55e9b18de4d88cc5c2a0d4d33bdf294ca14

[23] *Looking back at how Signal works, as the world moves forward.* (accessed: 24.11.2021). https://signal.org/blog/looking-back-as-the-world-moves-forward/.

[24] Cohn-Gordon, K. ; Cremers, C. ; Dowling, B. ; Garratt, L., and Stebila, D.: „A formal security analysis of the signal messaging protocol“. In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983.

[25] Sharma, A.: *Signal fixes bug that sent random images to wrong contacts.* (accessed: 25.10.2021). https://www.bleepingcomputer.com/news/security/signal-fixes-bug-that-sent-random-images-to-wrong-contacts/ (visited on 07/26/2021).

[26] CVE: *Security Vulnerabilities of Signal.* (accessed: 26.10.2021). https://www.cvedetails.com/vulnerability-list/vendor_id-17912/Signal.html.

[27] *Signal Android Issues.* (accessed: 30.10.2021). https://github.com/signalapp/Signal-Android/issues.

[28] Botha, J. ; Vant, W., and Leenen, L.: „A comparison of chat applications in terms of security and privacy“. In: (2019), p. 55.

[29] *Signal Android. A private messenger for Android.* (accessed: 25.10.2021). https://github.com/signalapp/Signal-Android.

[30] Emil, A.: *How to build Signal from the sources.* (accessed: 25.10.2021). `https://github.com/signalapp/Signal-Android/wiki/How-to-build-Signal-from-the-sources` (visited on 08/11/2021).

[31] *Signal-Android Contributing.* (accessed: 24.11.2021). `https://github.com/signalapp/Signal-Android/blob/master/CONTRIBUTING.md`.