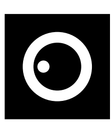


POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

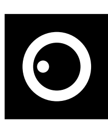
Forensic Analysis of Signal Desktop

Decrypting Artifacts and Automating Evidence Extraction from Signal Desktop

Gonçalo Alexandre Bernardino Rosa Paulino

School of Management and Technology
Department of Computer Engineering
Master in Cybersecurity & Digital Forensics

Leiria, September 2025



Forensic Analysis of Signal Desktop

Decrypting Artifacts and Automating Evidence Extraction from Signal Desktop

Gonçalo Alexandre Bernardino Rosa Paulino
Student No. 2230469

Supervisor: Miguel Frade
Professor, Polytechnic of Leiria

Co-supervisor: Patrício Domingues
Professor, Polytechnic of Leiria

Miguel Negrão
Professor, Polytechnic of Leiria

School of Management and Technology
Department of Computer Engineering
Master in Cybersecurity & Digital Forensics

Project

Leiria, September 2025

This page intentionally left blank.

Forensic Analysis of Signal Desktop

Copyright © 2025 - Gonçalo Alexandre Bernardino Rosa Paulino, School of Management and Technology.

This dissertation is original work, written solely for this purpose, and all the authors whose studies and publications contributed to it have been duly cited. Partial reproduction is allowed with acknowledgment of the author and reference to the degree, academic year, institution—*Polytechnic University of Leiria*—and public defense date.

Preparation of this work was facilitated by the use of the *IPLeiria-Thesis* template.

This page intentionally left blank.

Acknowledgements

I would like to express my deepest gratitude to my mentors, Miguel Frade, Miguel Negrão, and Patrício Domingues, for their guidance, mentorship, and invaluable support throughout this project. Their expertise and encouragement were fundamental in helping me overcome challenges and stay focused during the entire process.

I would also like to express my gratitude to my friend Tiago Santos, whose help in testing Signal's functionalities and generating test data for analysis was invaluable, and whose constant encouragement and moral support greatly motivated me throughout this project.

I am deeply thankful to my parents, Célia and Jorge, my brother, Dinis, and my extended family and friends for their unwavering encouragement and support throughout this journey.

Finally, I would like to dedicate this work to my late grandmother, Maria Olinda, whose memory continues to inspire me and motivate me every day. I miss her dearly and know she would be proud of this accomplishment.

This page intentionally left blank.

Resumo

Com a crescente preocupação com a segurança e privacidade das conversas pessoais, as aplicações de mensagens instantâneas com encriptação ponta-a-ponta tornaram-se um foco importante da investigação forense. Este estudo apresenta uma metodologia detalhada e um script em Python para desencriptar e analisar artefactos forenses do Signal Desktop em ambientes Windows e Linux. A abordagem divide-se em duas fases: i) a desencriptação dos dados armazenados localmente e ii) a análise e documentação dos artefactos forenses. Para preservar a integridade dos dados, a extração pode ser realizada sem iniciar o Signal Desktop, evitando alterações indesejadas. Em sistemas Linux, é ainda possível realizar uma extração forense completa diretamente a partir dos ficheiros. Os dados extraídos são processados e organizados em vários relatórios, facilitando o trabalho dos investigadores forenses. Adicionalmente, descrevemos um processo detalhado, passo a passo, para extrair dados da Gnome Keyring e da KWallet, útil em ambientes Linux onde o Signal Desktop depende destes sistemas para armazenamento seguro, mas também aplicável a outros dados não relacionados com o Signal. Os métodos apresentados oferecem uma base sólida para a extração e análise de artefactos encriptados do Signal em várias plataformas desktop, facilitando a realização de investigações forenses rigorosas.

Palavras-Chave: Informática forense, Signal, Electron, KWallet, Gnome Keyring, Desencriptação, Mensagens instantâneas

This page intentionally left blank.

Abstract

With growing concerns over the security and privacy of personal conversations, end-to-end encrypted instant messaging applications have become a key focus of forensic research. This study presents a detailed methodology along with an automated Python script for decrypting and analyzing forensic artifacts from Signal Desktop on both Windows and Linux environments. The methodology is divided into two phases: i) decryption of locally stored data and ii) analysis and documentation of forensic artifacts. To ensure data integrity, this approach enables retrieval without launching Signal Desktop, preventing potential alterations. On Linux, it additionally allows a full forensic extraction directly from stored files. A reporting module organizes the extracted data for forensic investigators, enhancing usability. We also provide a comprehensive step-by-step process for forensically extracting data from Gnome Keyring and KWallet, supporting Linux environments where Signal Desktop relies on these systems for secure storage, while also enabling potential applications beyond Signal-related data The methods presented provide a robust framework for extracting and analyzing encrypted Signal artifacts across desktop platforms, supporting thorough forensic investigations.

Keywords: Digital forensics, Signal, Electron, KWallet, Gnome Keyring, Decryption, Instant messaging

Contents

List of Figures xi

List of Tables xiii

Glossary xv

Acronyms xvii

1 Introduction **1**

2 Related Work **5**

3 Background **8**

3.1 Digital Forensic Concepts 8

3.2 Relevant Signal Internals 9

4 Materials and Methods **12**

4.1 Hardware 12

4.2 Software 12

4.3 Method 14

5 Decrypting Signal Artifacts **17**

5.1 Important File Locations 17

5.2 Signal Desktop Artifacts' Decryption Process 18

5.2.1 First Step: Decrypting SQLCipher Database 22

5.2.2 Second Step: Decrypting Attachments and Avatars 24

5.3 Forensically Extracting Data from Gnome Keyring 25

5.3.1 Gnome Keyring File Structure 26

5.3.2 Extracting Secrets from the Decrypted Data 28

5.3.3 Example Implementation 30

5.4 Forensically Extracting Data from KDE Wallet 30

5.4.1 KWallet File Structure 30

5.4.2 Decrypting KWallet Data 33

6 Forensic Artifacts **35**

6.1 Items Table 35

6.2 Conversations Table 36

6.3 Calls History Table . . . 39
6.4 Messages Table . . . 40
6.4.1 Message Types . . . 40
6.4.2 Body, Mentions and Text Formatting . . . 41
6.4.3 Preview . . . 43
6.4.4 Attachments . . . 43
6.4.5 Edit History . . . 43
6.4.6 Reactions . . . 43
6.4.7 Read Receipts and Send Status . . . 45
6.4.8 Quote . . . 47
6.4.9 Stories . . . 47
6.4.10 Group Changes . . . 47
6.5 Redundant Tables . . . 48

7 The SignalForensics Script 50
7.1 Execution Modes . . . 50
7.1.1 Live (`-m live`) . . . 52
7.1.2 Forensic (`-m forensic`) . . . 52
7.1.3 Auxiliary Key Provided (`-m aux`) . . . 53
7.1.4 Decryption Key Provided (`-m key`) . . . 53
7.2 Additional Options . . . 54
7.2.1 Flags . . . 54
7.2.2 Output Management . . . 54
7.3 Input/Output Arguments . . . 54
7.3.1 Signal Data Directory . . . 55
7.3.2 Output Directory . . . 55
7.3.3 Key Input Arguments . . . 55
7.3.4 Passphrase and Password Input Arguments . . . 55
7.3.5 Gnome Keyring File . . . 55
7.3.6 KWallet File . . . 56
7.3.7 KWallet Salt File . . . 56
7.3.8 GPG Key File . . . 56
7.4 Usage Examples . . . 56
7.4.1 Live Mode . . . 56
7.4.2 Forensic Mode (Gnome Keyring) . . . 56
7.4.3 Forensic Mode (KWallet, Blowfish) . . . 57
7.4.4 Forensic Mode (KWallet, GPG) . . . 57
7.4.5 Auxiliary Key Mode . . . 57
7.4.6 SQLCipher Key Mode . . . 57
7.4.7 Passphrase Mode . . . 57
7.5 Relevant Updates . . . 58

8 Conclusion **59**

Bibliography 62

Appendices

A Gnome Keyring Data Extraction: Example Implementation **69**

B Signal Desktop's SQLCipher Database Schema **72**

C Possible Group Changes In Signal Desktop **94**

This page intentionally left blank.

List of Figures

5.1 Structure of the Signal Desktop data directory. 18
5.2 Flowchart of the database's decryption process in Windows. 19
5.3 Flowchart of the database's decryption process in Linux. 20
5.4 Flowchart of the decryption process for attachments and avatars. 21
5.5 Encrypted auxiliary key in `Local State` (Windows) 23
5.6 Encrypted SQLCipher key in `config.json` (Windows) 23
5.7 Encrypted SQLCipher key in `config.json` (Linux/Gnome) 23
5.8 Hexadecimal view of a sample Gnome Keyring file. 27
5.9 Decrypted payload from an example Gnome Keyring file. 29
5.10 Hexadecimal view of a decrypted KWallet file. 31

6.1 ER diagram of the SQLCipher database schema. 37
6.2 An example message displayed in Signal Desktop. 43

7.1 Example HTML report generated by SignalForensics 52

This page intentionally left blank.

List of Tables

4.1 List of operating systems used in the study and their respective versions. . 12
4.2 List of the software used in this study and their respective versions. 13
4.3 List of Signal Desktop versions used in this study. 14

6.1 Relevant Tables in the SQLCipher database. 36
6.2 Relevant attributes found in the database's items table. 36
6.3 Conversation fields that are expected in every conversatio. 38
6.4 Conversation fields that are expected in private conversations. 38
6.5 Conversation fields that are expected in group conversations. 39
6.6 Key elements in a message's JSON object 41
6.7 Possible read receipt states for received messages. 45

7.1 Execution modes in SignalForensics. 51
7.2 Compatibility of execution modes across supported environments. 51

C.1 List of possible group changes. 94

Acronyms

ACI Account Identifier. (*p. 10, 36, 38, 39, 42, 47, 95*)
ACL Access Control List. (*p. 29*)
AES Advanced Encryption Standard. (*p. 2, 5, 6, 22, 24, 28*)
API Application Programming Interface. (*p. 2, 6, 7, 13, 22, 28*)

CBC Cipher Block Chaining. (*p. 24, 28, 31–33*)
CSV Comma Separated Values. (*p. 3, 16, 50, 54, 59*)

DPAPI Data Protection API. (*p. 2, 3, 6, 13, 18, 22, 52, 59, 60*)

E2EE End-to-End Encryption. (*p. 1, 2*)
ECB Electronic Code Book. (*p. 31, 32, 34*)
ER Entity-Relationship. (*p. xi, 35, 37*)

GCM Galois/Counter Mode. (*p. 6, 22*)

IDFIF Integrated Digital Forensics Investigation Framework. (*p. 5*)
IV Initialization Vector. (*p. 24, 25, 28, 33, 34*)

PBKDF2 Password-Based Key Derivation Function 2. (*p. 24, 31–33*)
PNI Phone Number Identifier. (*p. 10, 36, 38*)

UTC Universal Time Coordinated. (*p. 35, 54*)
UUID Universally Unique Identifier. (*p. 38, 41, 47*)
UWP Universal Windows Platform. (*p. 6*)

1 Introduction

Instant messaging applications have been the focus of numerous studies aimed at understanding their functionality and the different artifacts that can be collected for forensic investigations (Son et al., 2022). Due to the nature of these applications, their data have significant forensic value, enabling investigators to infer aspects from interpersonal relationships to an individual's mental state (Steel, 2014; Andriotis et al., 2014).

With the increasing concern of everyday users about the security and privacy of their personal conversations (Madden, 2014; Auxier et al., 2019), the number of instant messaging solutions that market themselves as secure has grown. These solutions incorporate various security mechanisms — such as End-to-End Encryption (E2EE), self-destructing messages, local storage encryption, and forward secrecy — which introduce new challenges to the forensic analysis of these applications. Self-destructing messages — often referred to as ephemeral messages — in particular, have attracted significant interest within the digital forensic scientific community (Abegaz et al., 2024; Heath et al., 2023).

Notable examples of privacy-focused instant messaging applications include centralized solutions such as Signal[1], WhatsApp[2], Threema[3], and Wire[4], along decentralized alternatives such as Briar[5], Session[6], and SimpleX Chat[7]. Among these, Signal is one of the most widely adopted and pioneered the Signal Protocol, which has also been adopted by WhatsApp (Marlinspike, 2016) — the leading messaging app — making it an ideal candidate for this study. At the time of this writing, Signal has surpassed 100 million downloads on the Google Play Store[8] and 900 000 ratings on the Apple Store[9].

Signal is an open-source instant messaging application that offers robust privacy

[1] https://signal.org
[2] https://www.whatsapp.com
[3] https://threema.ch
[4] https://wire.com
[5] https://briarproject.org
[6] https://getsession.org
[7] https://simplex.chat
[8] https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms
[9] https://apps.apple.com/app/signal-private-messenger/id874139669

and security features (Cohn-Gordon et al., 2020). Its centerpiece is E2EE, powered by the Signal Protocol, an open source cryptographic protocol developed internally by the Signal Foundation (Signal Foundation, 2025b).

The Signal Protocol combines multiple algorithms, including Double Ratchet (Perrin et al., 2016), Curve25519 (Langley et al., 2016), and AES-256 (NIST, 2001), to ensure the security of user communications, guaranteeing not only End-to-End Encryption, but also forward and future secrecy (Signal Foundation, 2025b; Kret et al., 2024).

Signal offers Android[8] and iOS[9] clients for mobile devices, as well as a cross-platform Electron-based desktop application compatible with Windows, macOS, and Linux. This project focuses on the forensic analysis of the desktop version — Signal Desktop — in both Windows and Linux environments. Signal Desktop offers standalone messaging capabilities but requires an initial link to a mobile device for account creation and synchronization. On Windows, Signal Desktop secures its local data using SQLCipher and leverages the Data Protection API (DPAPI) for key protection. On Linux systems, the situation is more complex, as multiple key storage backends are supported. This work considers the three possible Linux scenarios: i) environments where the Gnome Keyring (via `libsecret`) is available; ii) environments using KWallet; and iii) those with neither secret storage system. Our research spanned Signal Desktop[10] versions **7.32.0** to **7.60.0**, released between November 2024 and July 2025.

This project aims to address the forensic challenges posed by Signal Desktop's encrypted local across the aforementioned environments. Specifically, we set out to:

1. Investigate how Signal Desktop stores and secures forensic artifacts locally.
2. Identify and document the cryptographic mechanisms involved in securing those artifacts.
3. Develop and validate a methodology for decrypting the locally stored data without launching the application.
4. Analyze the decrypted artifacts to determine their forensic value;
5. Create an automated tool that streamlines both the decryption and reporting processes.

The key contributions of this work to the field of digital forensics are summarized as follows:

i) **Documentation of Signal Desktop's Decryption Process:** We document the step-by-step process required to decrypt Signal Desktop's locally stored data, including the retrieval of cryptographic materials and the use of SQLCipher and Electron's safeStorage Application Programming Interface (API). This fills a gap in the literature with respect to the forensic analysis of this application.

ii) **Methodology for Forensic Extraction of Data from Gnome Keyring and KDE Wallet:** We present a detailed process for forensicaly extracting data the Gnome

[10] `https://signal.org/download`

Keyring and KDE Wallet, supporting forensic workflows on common Linux desktop environments. In particular, this process enables access to encrypted data in scenarios where Signal Desktop relies on the Gnome Keyring or KDE Wallet for secure data storage, making it possible to recover critical artifacts from a cold system image.

iii) **Development of an Automated Forensic Tool:** We introduce SignalForensics — `https://github.com/labcif/SignalForensics` — a Python-based tool that automates the decryption and extraction of forensic artifacts from Signal Desktop's local files. This tool eliminates the need for manual decryption, significantly reducing the time and complexity involved in analyzing Signal's encrypted data.

iv) **Comprehensive CSV/HTML Reports for Forensic Analysis:** SignalForensics generates detailed CSV and HTML reports that record the forensic artifacts found in Signal's local files. These reports transform raw decrypted data into a structured and accessible format, making it easier for forensic investigators to analyze and interpret the evidence.

v) **Support for Expired or Deleted Messages:** Unlike when reading directly from the application, our approach enables the extraction of content that would otherwise no longer be available, such as expired or deleted messages. This ensures that valuable evidence is preserved even after the intended lifespan of the data within the application has passed.

vi) **Forensically Sound Artifact Retrieval:** The proposed methodology and tool facilitates artifact retrieval in a manner that avoids interference with the data, maintaining the integrity of forensic evidence to the greatest extent possible. In addition, the tool generates a cryptographic hash for all processed files, enabling investigators to verify the integrity of the extracted data. In Windows environments, where the operating system must be active to retrieve encryption keys via Data Protection API (DPAPI), the process still avoids launching Signal Desktop directly, thereby preventing any modifications to user data within the application. For example, view-once media messages, which normally self-destruct upon being opened in the official application, can be extracted and preserved using this approach. This allows investigators to securely access and analyze such media without triggering their deletion, ensuring availability of otherwise ephemeral content.

vii) **Addressing Key Cryptographic Challenges:** Although SignalForensics requires access to the execution environment or the ability to decrypt DPAPI-protected keys externally in Windows environments, the situation differs in Linux systems, where full decryption is possible using only the keyring or wallet artifacts and their associated credentials. In both cases, the tool represents a significant advancement in forensic analysis by automating previously labor-intensive steps and providing new insights into Signal's encryption mechanisms and secure data storage practices. It should be noted that if the underlying disk is protected using full-disk encryption technologies such as BitLocker, LUKS, or similar, the disk

must first be decrypted before this methodology can be applied.

viii) **Scientific Dissemination:** A portion of this project has been published in the scientific article "Decrypting messages: Extracting digital evidence from Signal Desktop for Windows", in **Forensic Science International: Digital Investigation**, available online since June 10, 2025 (Gonçalo Paulino, Miguel Negrão, Miguel Frade and Patrício Domingues, 2025) at https://doi.org/10.1016/j.fsidi.2025.301941. This article covers the core methodology and tool development for the Windows platform, forming the foundation upon which this project expands. The article was published as open access, ensuring it is freely available to the public without subscription or payment barriers, thus promoting wider dissemination and use of its findings within the forensic and research communities.

The remainder of this report is organized as follows. Chapter 2 reviews existing research in this field. Chapter 3 introduces key concepts and explains how certain Signal features are implemented, providing essential context for the following chapters. Chapter 4 outlines the hardware, operating systems, and software tools used in this study, as well as the methodology followed to achieve the research objectives. Chapter 5 details the locations of relevant files within Signal Desktop and describes the decryption scheme, employed by the application, that must be mimicked to decrypt these files. It also includes dedicated sections outlining the process of forensically extracting data from the Gnome Keyring and the KDE Wallet. Chapter 6 examines the forensic artifacts retrievable from Signal Desktop's local files and explains their interrelations, detailing the forensically relevant information that can be extracted from them. Chapter 7 describes the developed script, which automates Signal Desktop's decryption process and generates informative reports. Finally, Chapter 8 concludes this work.

2
Related Work

Given that Electron applications run on the Chromium engine (OpenJS Foundation et al., 2023), it is relevant to review the existing literature on the cryptographic methods implemented in this platform, as well as forensic analyses conducted in similar environments. This section reviews the existing literature, addressing methodologies, challenges, and key findings pertinent to the domain.

Karo Karo et al. (2024) examined the forensic artifacts generated by the Signal Desktop application on Windows 11 under three distinct scenarios: i) Sending messages, photos, and PDFs without deleting messages; ii) Deleting messages, photos, and PDFs after sending; iii) Uninstalling the Signal Desktop application. Although no chat logs were successfully recovered in any of the scenarios, all shared attachments were retrieved in the first case, whereas only some were recovered in the second.The study mainly focused on applying the Integrated Digital Forensics Investigation Framework (IDFIF) v2 (Kohn et al., 2013) methodology to a real-world scenario and utilizing artifact acquisition tools, such as FTK Imager and Autopsy, for data collection. However, no in-depth analysis of the generated artifacts was conducted.

Bilz (2021) investigated the forensic artifacts generated by **version 5.4.1** of the Signal Desktop application on Windows 10. They observed that the directories within Signal's Roaming and Local folders follow the expected structure of an Electron-based application, something we do confirm in our work. According to Bilz (2021), the database present in the `sql` directory within Signal's Roaming folder contains most of the relevant forensic data. Although the database format is SQLCipher 4 — an open source SQLite extension that provides transparent 256-bit Advanced Encryption Standard (AES) full database encryption (Zetetic, LLC, 2024) — the encryption key is stored in plaintext within a JSON configuration file (Bilz, 2021). Using the DB Browser for SQLCipher tool and this key, the investigator gained full access to the contents of the encrypted database. However, when we applied this methodology in **version 7.32.0** of the application, it yielded different results. It was impossible to decrypt the database due to changes in the storage of cryptographic keys or the way data is encrypted.

Other works involving forensic analysis of applications with instant messaging ca-

pabilities built on top of Electron include the work of Gupta et al. (2024), which analyzes Discord, and Bowling et al. (2023), which focuses on Microsoft Teams.

Although non-Windows and non-Electron related, Son et al. (2022) forensically analyzed the Android version of three instant messaging applications, one of which was Signal, version **5.19.4**. They exported the messenger backup migration from the non-rooted Android device to a rooted one, thus leveraging root access to study the Signal application. The authors also developed an Android application designed to interact with the Android Keystore leveraging key stored in the Keystore to perform decryption.

Kim et al. (2025) investigated the forensic artifacts generated by the Web and Universal Windows Platform (UWP) versions of WhatsApp, analyzing their encryption schemes and proposing decryption methods. Their study revealed that while the Web version stores most decryption-related elements in the browser's storage, additional steps were required to fully decrypt the local files. Similarly, for the UWP version, they identified a complex encryption mechanism in which the database encryption key was secured through multiple layers, necessitating an alternative approach to retrieve it without executing the application. Despite these challenges, the researchers eventually succeeded in decrypting the UWP version using only locally stored data. They discovered that the database encryption key was quadruple-encrypted and required a call to the UWP API for decryption. However, since running the application and hooking into it to collect data is impractical in real-world forensic scenarios, they analyzed the detailed operation of the API and proposed a method to collect the data required without invoking the API.

A.P. Taneva (2023) examined how cookies are encrypted in Google Chrome on Windows systems. The researcher observed that Chrome employs a combination of AES-256-GCM[1] encryption and DPAPI[2] protection to secure the database containing the browser cookies. Although the study offers valuable insights, it only briefly touches upon this encryption process. A closer examination will be relevant to our investigation since both Chrome and the Electron framework — in which Signal Desktop is based on — are built on the Chromium engine (OpenJS Foundation et al., 2023; The Chromium Project, 2024).

In July 2024, Signal addressed a significant problem regarding the decryption key for the Desktop applications' local database. Previously, this key was stored in plaintext on the local disk, which presented a potential vulnerability if an attacker gained access to the file system (Abrams, 2024). Listing 1 exemplifies the contents of a `config.json` file generated by an old Signal Desktop version. An X (formerly Twitter) user[3], demonstrated that it was possible to usurp the session of a Signal user by copying the applica-

[1] Galois/Counter Mode is a mode of operation for symmetric-key cryptographic block ciphers, designated authenticated encryption, as the name suggests it provides both encryption and authentication by combining counter mode encryption with Galois field multiplication.

[2] DPAPI, or Data Protection API, is a Windows service for encrypting sensitive data, using user or system keys for protection.

[3] https://x.com/mysk_co/status/1809287118235070662

tion's data from one system to another (Abrams, 2024). Following public scrutiny, Signal implemented a fix by adopting Electron's safeStorage API. This update introduces an opportunity for forensic and cybersecurity analysis, warranting a re-evaluation of Signal Desktop's resilience against potential attack vectors and its suitability for forensic artifact collection.

Listing 1: *An example of the contents of the* `config.json` *file from when the SQLCipher key was stored unprotected.*

```
{
  "key": "7eeec7c5f94553a2e6caf6665e4255c4c427ca2be6d6fedfb25ba0bc803fd231",
  "mediaPermissions": true,
  "mediaCameraPermissions": true
}
```

In summary, prior research highlights both the opportunities and the evolving challenges of examining encrypted messaging platforms. Earlier versions of Signal Desktop and comparable Electron applications revealed weaknesses—such as plaintext key storage—that simplified forensic access, while more recent releases have introduced stronger protections that demand new techniques. This context underscores the need for the methodology presented in this thesis, which addresses these modern encryption mechanisms and extends forensic analysis across multiple operating systems.

3 Background

Understanding how forensic information can be extracted from Signal Desktop requires context from both the field of digital forensics and the technical workings of the application itself. This chapter is divided into two sections. The first outlines key terminology and foundational concepts in digital forensics, providing the vocabulary and perspective needed to frame the artifacts analyzed throughout this work. The second focuses on the internal design of Signal Desktop, detailing some of the features that generate such artifacts and explaining concepts that will be referenced in later chapters.

3.1 Digital Forensic Concepts

Before examining Signal Desktop from a forensic standpoint, it is important to define key terms commonly used in digital forensics. This section introduces foundational concepts, establishing a common vocabulary for the remainder of this report that will help contextualize the data retrieved and analyzed over the course of this project.

Digital Forensic Investigation. A digital forensic investigation is the process identifying, collecting, preserving, analyzing, and presenting digital evidence in a manner that maintains its integrity and admissibility in legal or disciplinary proceeding (Kent et al., 2006). This process adheres to established forensic principles to ensure that conclusions drawn from digital data are both reliable and verifiable.

Digital Evidence. Digital evidence refers to any information of probative value that is stored or transmitted in a digital format and that may be used in a court of law or during an investigation (Casey, 2014a). It encompasses data found on digital devices such as computers, smartphones, and cloud services, including logs, files, messages, metadata, and network traces.

Forensic Artifact. A forensic artifact is any piece of residual data created or modified by user activity or system operations that can serve as digital evidence in a forensic investigation (Casey, 2014a). Artifacts may include logs, cache files, configuration files, deleted content, or database entries — essentially any traceable remnant that reveals past interactions with a device or application.

Post-Mortem Forensics. Post-mortem forensics refers to the examination of digital systems after they have been powered down, typically through analysis of disk images or exported files (Casey, 2014b). This approach minimizes the risk of altering evidence during analysis, but may limit access to volatile data such as RAM contents or active processes.

Live Forensics. Live forensics involves analyzing a system while it is running, allowing access to volatile data, active processes, network sessions, and other information not preserved in post-mortem images (Casey, 2014b). Although it enables acquisition of ephemeral evidence, this method introduces potential risks of modifying the system and its stored data during investigation.

Evidence Integrity. Maintaining the integrity of digital evidence is a fundamental principle in forensic investigations. Any alteration to the data — whether accidental or intentional — can compromise its admissibility in legal proceedings. Cryptographic hash functions, such as SHA-256, are commonly employed to generate unique fingerprints of digital artifacts. By calculating and storing these hash values at the time of acquisition, investigators can later verify that the evidence has not been modified throughout the investigation. This practice is essential for preserving the chain of custody and ensuring that forensic findings remain reliable and legally defensible (Casey, 2014b).

3.2 Relevant Signal Internals

Signal is a centralized end-to-end encrypted instant messaging application. As outlined in the Introduction chapter, Signal incorporates various security and privacy-oriented solutions. Most of these operate seamlessly in the background, remaining largely unnoticed by the user. In this project, the focus is on the features that users actively perceive and interact with, which in turn generate valuable forensic artifacts. This section explores the implementation of some of these features within Signal, explaining key concepts that will be referenced throughout the report.

Conversations. In Signal's context, a "conversation" refers to a *chat room* – a place where messages can be exchanged. A chat room can be either a one-on-one private channel between two users or a group chat where up to 1 000 users can communicate with each other (Signal Foundation, 2025a). These can be distinguished by the type

of conversation, which will be "private" for the former and "group" for the latter, as well as the differing metadata each contains. For example, as evidenced in Listings 2 and 3, only a group conversation will have a "groupId" field, whereas only a private conversation will feature a "serviceId" field. It is relevant to note that the "Notes to self" function is handled under the hood as a single-user private chat that is only accessible by the current user.

Listing 2: *Excerpt of a group conversation's information stored in the `json` column of the `conversations` table.*

```
{
  "id": "0c66f702-102c-45f2-b8f6-367b57da7970",
  "groupId": "5AxQsdshh8XupFF8gwQPFY6QIAWz6c+AyTxugrBl0cQ=",
  "type": "group",
  "name": "Test Group"
}
```

Listing 3: *Excerpt of a private conversation's information stored in the `json` column of the `conversations` table (phone number redacted for privacy).*

```
{
  "id": "bbf29929-62ac-4371-aa2b-ef8f50fe168d",
  "serviceId": "1ed50204-ea7f-498b-8bfa-6999f63c4143",
  "e164": "+35191XXXXXXX"
  "type": "private",
  "name": "Person A"
}
```

Contacts. In Signal, the concept of a "contact" does not exist as a distinct internal entity. While the user interface provides the impression of a contact list (*e.g.*, when adding group members, or when sending the first message to a recipient), Signal manages contacts as private chat conversation instances. As described in the previous subsection, they can be identified by their `serviceId` field. These conversations are what we refer to as "contacts" in this report. When the user has not yet received any messages from a particular Signal contact, the `serviceId` corresponds to that contact's Phone Number Identifier (PNI). However, once the user receives a message, the `serviceId` changes to an Account Identifier (ACI) — *e.g.*, `1ed50204-ea7f-498b-8bfa-6999f63c4143`.

Stories. Similar to other social media and instant messaging applications, a "story" is a temporary post consisting of text, images, or videos, shared with selected contacts. These posts are only visible for a limited duration of 24 hours[1], after which they auto-

[1] `https://support.signal.org/hc/articles/5008009166234-Stories`

matically disappear. In Signal, an outgoing story is handled as a single message sent to the selected contacts mentioned above. Likewise, an incoming story is regarded as a message received from the contact who shared the story. In both scenarios, the message instance will have the `type` field set to `story`.

View-once Media. This feature enables users to send messages containing individual photos or videos that are automatically removed from the conversation after being viewed. Although sending these messages is exclusive to the Android and iOS apps, the Desktop version can still receive and view these messages. These messages retain their standard `incoming` or `outgoing` type, depending on whether they were received or sent by the current user, but they are distinguished by the `isViewOnce` field being set to `true`.

Current User. Throughout this report, the term "current user" is frequently used. This refers to the user whose account was logged into the Signal Desktop application at the time the analyzed artifacts were generated. It distinguishes the "current user" from the broader term "user" which, unless otherwise specified, refers to any Signal user.

Chapter Summary

In summary, this chapter established some fundamental terminology of digital forensics and outlined the internal mechanisms of Signal Desktop most relevant to evidence collection. By clarifying concepts such as live versus *post-mortem* analysis, evidence integrity, and the structures underlying Signal conversations and messages, it provides the technical and methodological foundation on which the subsequent chapters build the proposed forensic methodology and its practical application.

4 Materials and Methods

This chapter describes the hardware and operating systems we conducted our research on, the software tools employed during the investigation, and the methodology followed to achieve the study's objectives.

4.1 Hardware

The research was conducted using systems running Windows 11. Additionally, an Android mobile device was utilized to set up the Signal account, as the Signal Desktop application requires an initial connection to a mobile device for account creation. This requirement arises from Signal's design, which mandates a phone number for account registration. While the mobile device is not necessary for receiving messages on Signal Desktop once the account is established, it remains essential for the initial registration process. Table 4.1 provides an overview of the operating systems used in the study, along with their respective versions.

Table 4.1: *List of operating systems used in the study and their respective versions.*

OS	version	Build number
Windows 11	24H2	26100.3194
Windows 11	23H2	22621.4317
Ubuntu 24	24.04.2	—

4.2 Software

A wide range of tools played a critical role throughout the study, serving various purposes such as reverse engineering, data analysis, and environment simulation. For instance, CyberChef[1], with its versatile "recipe" system for building custom processing pipelines, was invaluable for encoding, decoding, and cryptographic operations

[1] https://gchq.github.io/CyberChef/

across multiple stages of the research. NirSoft's DataProtectionDecryptor[2] — a Windows tool that exposes DPAPI decryption capabilities — proved essential during the reverse engineering of Electron's safeStorage API on Windows. Additionally, utilities such as HxD and DB Browser enabled the analysis of different file types and the extraction of valuable information. Table 4.2 enumerates all the tools employed along with their respective versions, while Table 4.3 documents the Signal Desktop versions used in our study. No relevant differences were identified between the examined Signal Desktop versions that would significantly impact our methodology or the findings of this study. Virtualization software such as VMWare Workstation[3] allowed for controlled testing across different environments, facilitating a comprehensive examination of Signal's behaviors across different operating systems and settings. In Linux environments, Signal Desktop relies on Gnome Keyring[4] or KWallet[5] for part of the encryption process. During the development and testing of the methodologies to extract data from these sources, the versions in use were Gnome Keyring `46.1` and KWallet `5.115.0`.

Table 4.2: *List of the software used in this study and their respective versions.*

Software	Version
DB Browser	3.13.1
NirSoft's DataProtectionDecryptor	1.13
HxD	2.5.0.0
CyberChef	10.19.4
Visual Studio Code	1.96.4

DB Browser for SQLite/SQLCipher is an open-source tool designed to simplify the querying, creation, design, and editing of SQLite and SQLCipher databases. DB Browser for SQLCipher[6] was crucial for analyzing the SQLCipher database generated by the Signal application.

HxD is a versatile hex editor designed for low-level data inspection and manipulation at the byte level, making it an invaluable tool for examining binary data and raw file structures. In our research, HxD[7] was utilized in reverse engineering tasks and in-depth analyses of data files produced by Signal.

NirSoft's DataProtectionDecryptor is a Windows tool that provides an interface for data decryption through DPAPI. In our research, it proved useful during the reverse engineering of Electron's safeStorage.

[2] https://www.nirsoft.net/utils/dpapi_data_decryptor.html
[3] https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion
[4] https://wiki.gnome.org/Projects/GnomeKeyring
[5] https://wiki.archlinux.org/title/KDE_Wallet
[6] https://sqlitebrowser.org/
[7] https://mh-nexus.de/en/hxd/

Table 4.3: *List of Signal Desktop versions used in this study.*

Version	Release Date	Version	Release Date
7.32.0	Nov 6, 2024	7.45.1	Mar 7, 2025
7.33.0	Nov 13, 2024	7.46.0	Mar 12, 2025
7.34.0	Nov 20, 2024	7.46.1	Mar 17, 2025
7.35.0	Dec 4, 2024	7.47.0	Mar 19, 2025
7.35.1	Dec 6, 2024	7.48.0	Mar 27, 2025
7.36.0	Dec 12, 2024	7.49.0	Apr 2, 2025
7.36.1	Dec 18, 2024	7.50.0	Apr 9, 2025
7.37.0	Jan 8, 2025	7.51.0	Apr 17, 2025
7.38.0	Jan 15, 2025	7.52.0	Apr 23, 2025
7.39.0	Jan 23, 2025	7.53.0	Apr 30, 2025
7.40.0	Jan 29, 2025	7.54.0	May 13, 2025
7.40.1	Feb 2, 2025	7.55.0	May 21, 2025
7.41.0	Feb 5, 2025	7.56.0	May 28, 2025
7.42.0	Feb 13, 2025	7.56.1	Jun 4, 2025
7.43.0	Feb 19, 2025	7.57.0	Jun 11, 2025
7.44.0	Feb 26, 2025	7.58.0	Jun 18, 2025
7.45.0	Mar 5, 2025	7.60.0	Jul 2, 2025

CyberChef is an open-source web application offering an intuitive interface that allows users to create processes by arranging operations in a block-like, flowchart style. This “recipe” format enables users to easily perform various data manipulation and cryptographic operations, such as encryption, decryption, hashing and encoding. CyberChef was used in almost every stage of this study, assisting with a wide range of encoding and cryptographic operations.

VMWare Workstation is a virtualization software that provides an intuitive interface to manage virtual machines. In our research, it enabled us to validate hypotheses across multiple Signal environments, facilitating controlled testing and analysis in distinct setups.

Visual Studio Code is a versatile open-source code editor with built-in debugging, Git integration, and extensive extension support. In our research, Visual Studio Code[8] was employed it for various code and data analysis tasks, such as inspecting Signal’s source code and visualizing JSON data in a user-friendly format.

4.3 Method

The methodology adopted in this study is divided into two distinct phases: i) the decryption of Signal Desktop’s locally stored data and ii) the analysis and documentation of digital forensic artifacts.

[8] https://code.visualstudio.com/

Each phase aimed to address different research objectives, ensuring a comprehensive understanding of the application's data storage mechanisms and the artifacts it generates. Additionally, as a secondary objective integrated into both phases, this study aimed to develop an automated Python script capable of decrypting Signal's local data and processing its forensic artifacts.

The first phase focused on decrypting the application's local data, which involved a combination of reverse engineering and cryptographic analysis. The steps undertaken were as follows:

1. **Analysis of Signal and Its Dependencies**: An in-depth examination into the application's architecture was conducted, including an analysis of its underlying frameworks — *i. e.*, Electron and Chromium. This investigation provided the necessary insights into the application's data storage and encryption mechanisms, which facilitated the mimicking of its local data decryption process.
2. **Collection of Cryptographically Relevant Data**: Key cryptographic materials, including encryption keys stored in specific configuration files, were identified and extracted. Additionally, the encrypted local files (*i. e.*, the cipher data) were recovered from the application's local storage in preparation for the next step.
3. **File Decryption**: Using the obtained cryptographic materials and the knowledge gained from the initial analysis, the encrypted files were successfully decrypted. This step confirmed the viability of the devised decryption process and validated the approach.
4. **Documentation of a Replicable Decryption Process**: Once the decryption process was confirmed to be effective, a detailed and structured method, described in Chapter 5, was formulated. This plan was designed to simplify the decryption process, ensuring that it could be easily replicated by other forensic investigators.
5. **Automation of the Decryption Process**: To streamline the devised decryption process, a Python script — described in Chapter 7 — was developed based on the previously established method. The script SignalForensics automates the extraction of cryptographic materials, decryption of files, and reconstruction of the original data. SignalForensics was designed to replicate the decryption process from various possible starting points, with minimal manual intervention, ensuring versatility and efficiency in future use.

The second phase involved the examination of the decrypted files and the identification of digital forensic artifacts generated by the application. The steps included:

1. **Analysis of Decrypted Files**: The decrypted files were analyzed to identify the types of information they contained. Particular attention was given to data that could be related to the user or their actions, such as activity records, message contents, and system logs.
2. **Simulation of User Activity**: To generate a comprehensive set of forensic artifacts, various tests were conducted on the application. These tests simulated

real-world user interactions, such as sending messages, making calls, and posting stories. The goal was to produce a diverse variety of data, including logs, caches, user metrics, and temporary files, representative of what one might encounter in a real forensic investigation.

3. **Reanalysis and Comparison of Artifacts**: The artifacts generated during the simulated user activity were compared with those identified in the initial analysis. This comparison aimed to establish a correlation between specific application functionalities and the artifacts they produce. For example, arbitrary values in certain fields (*e. g.*, 1, 2 or 3) were mapped to their corresponding meanings or actions within Signal.
4. **Documentation of Artifacts**: All identified artifacts were documented, detailing their locations, functionalities, and the relevant information they contained. This documentation provides a detailed reference for forensic investigators, enabling them to interpret and utilize these artifacts effectively in real-world scenarios.
5. **Automated Reporting of Forensic Artifacts**: Building upon the written documentation of the previous step, a reporting module was added to SignalForensics. This functionality generates detailed reports that summarize the identified artifacts in CSV and HTML format, transforming raw data into readable and interpretable information. The reporting module was designed to assist forensic investigators by extracting raw data and key insights from the application's forensic artifacts and presenting them in a structured, accessible format, thereby enhancing the efficiency and clarity of forensic analysis.

Chapter Summary

This chapter detailed the experimental setup and the methodology followed throughout the study. It described the hardware and operating systems used, the software tools that supported tasks such as reverse engineering, cryptographic analysis, and environment simulation, and the structured, two-phase approach adopted to decrypt and examine Signal Desktop's local data. By presenting both the practical environment and the step-by-step procedures, this chapter establishes the foundation for the subsequent analysis of decrypted artifacts and the presentation of results in the following chapters.

5 Decrypting Signal Artifacts

Signal Desktop stores various files locally to ensure smooth operation, data synchronization, and some offline functionality. Forensic analysis of these files requires identifying their locations and understanding how they are secured. This chapter outlines the structure of the relevant files and explains the decryption processes used to access them through different environments.

5.1 Important File Locations

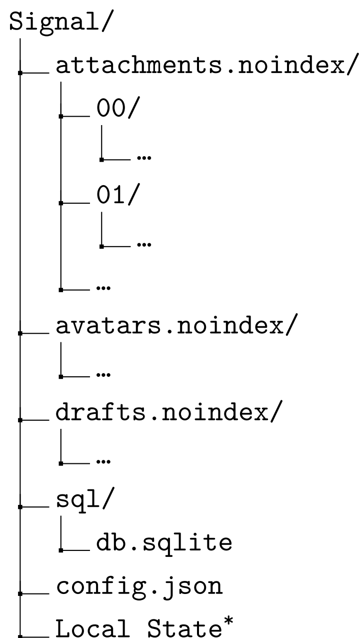
Before delineating how Signal secures its local files in the desktop application, it is important to understand which files exist and where they are located.

In Signal Desktop, all the relevant files are stored in the configuration directory, which varies depending on the operating system. On Linux, the configuration directory is located at `~/.config/Signal/`. On Windows, it is found in `%APPDATA%/Signal/` — that is, inside the user's personal Roaming folder. Henceforth, we refer to this folder as the "data directory". The data directory is structured in the following manner, as illustrated in Figure 5.1.

The `attachments.noindex` directory stores exchanged attachments and the current avatar of the user, contacts, and groups. The directory functions as a hash table in folder format, containing up to 256 subdirectories named after each possible hexadecimal value of a byte (*i. e.*, 00 to FF). Each file is assigned a filename based on the hexadecimal representation of 32 bytes of random data and is placed in the subdirectory corresponding to the first byte of the filename (*i. e.*, its key in the hash table). The directories `avatars.noindex` and `drafts.noindex` follow the same structure, with the former storing user-uploaded avatars — *i. e.*, avatars the user has uploaded to Signal, including those no longer in use — and the latter containing attachments included in the draft messages.

The `sql` directory contains a SQLCipher database named `db.sqlite`. This encrypted database stores various valuable forensic artifacts, including the user's personal information, metadata, exchanged messages, and the keys and nonces required

Figure 5.1: *Structure of the Signal Desktop data directory. The asterisk (*) indicates that this file is only present for Windows environments running Signal Desktop.*



to decrypt the attachments and avatars.

Finally, the `config.json` and `LocalState` files, located in the `Signal` directory, contain the cryptographic material required to decrypt the SQLCipher database. The `config.json` file holds the SQLCipher key in encrypted form. The `LocalState` file — present only on Windows systems — stores a DPAPI-protected key used to decrypt it. For easier comprehension, we refer to this as the "auxiliary key" in the following chapters. In Linux environments, this auxiliary key is not stored anywhere in encrypted format, but rather derived from a randomly generated passphrase saved in the user's keyring.

5.2 Signal Desktop Artifacts' Decryption Process

To examine the artifacts generated by Signal, it is first necessary to decrypt them. This chapter outlines the decryption process employed by Signal, providing a methodology for manually replicating it. We split this process in two steps. They are illustrated in the flowcharts presented in Figure 5.2, Figure 5.3 and Figure 5.4. The first step, shown in the first two figures, covers the decryption of the SQLCipher key and its use to retrieve data from the SQLCipher database. This process differs with the environment from which the artifacts were collected. The second step, illustrated in the latter figure, demonstrates how the cryptographic materials stored in the database can be used to decrypt exchanged attachments, draft message attachments, and avatars. Unlike the first step, this process is consistent across all environments, as it relies solely on the cryptographic metadata stored in the database.

Figure 5.2: *Flowchart illustration of the decryption process for the SQLCipher database. (Windows)*

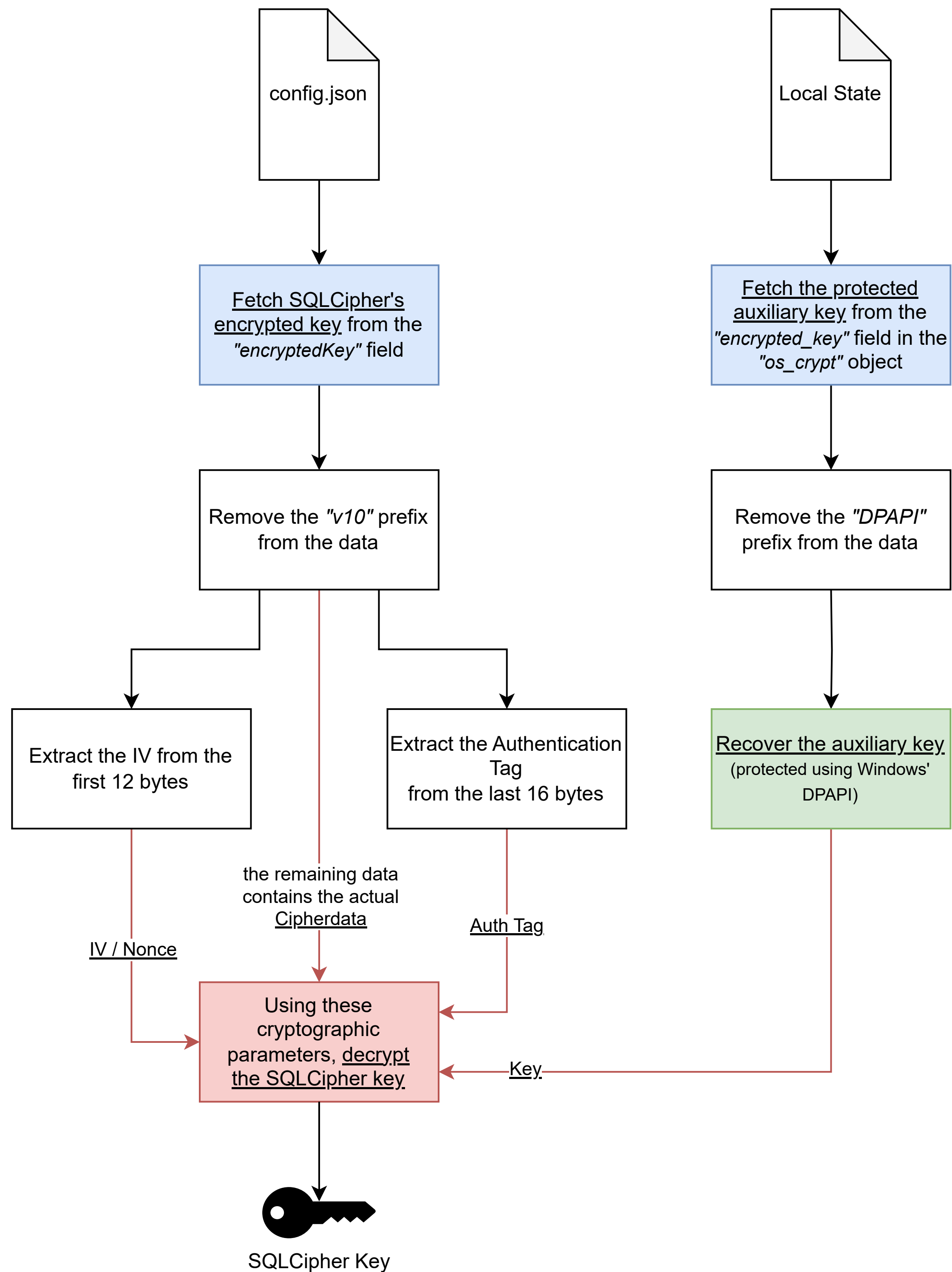


Figure 5.3: *Flowchart illustration of the decryption process for the SQLCipher database. (Linux)*

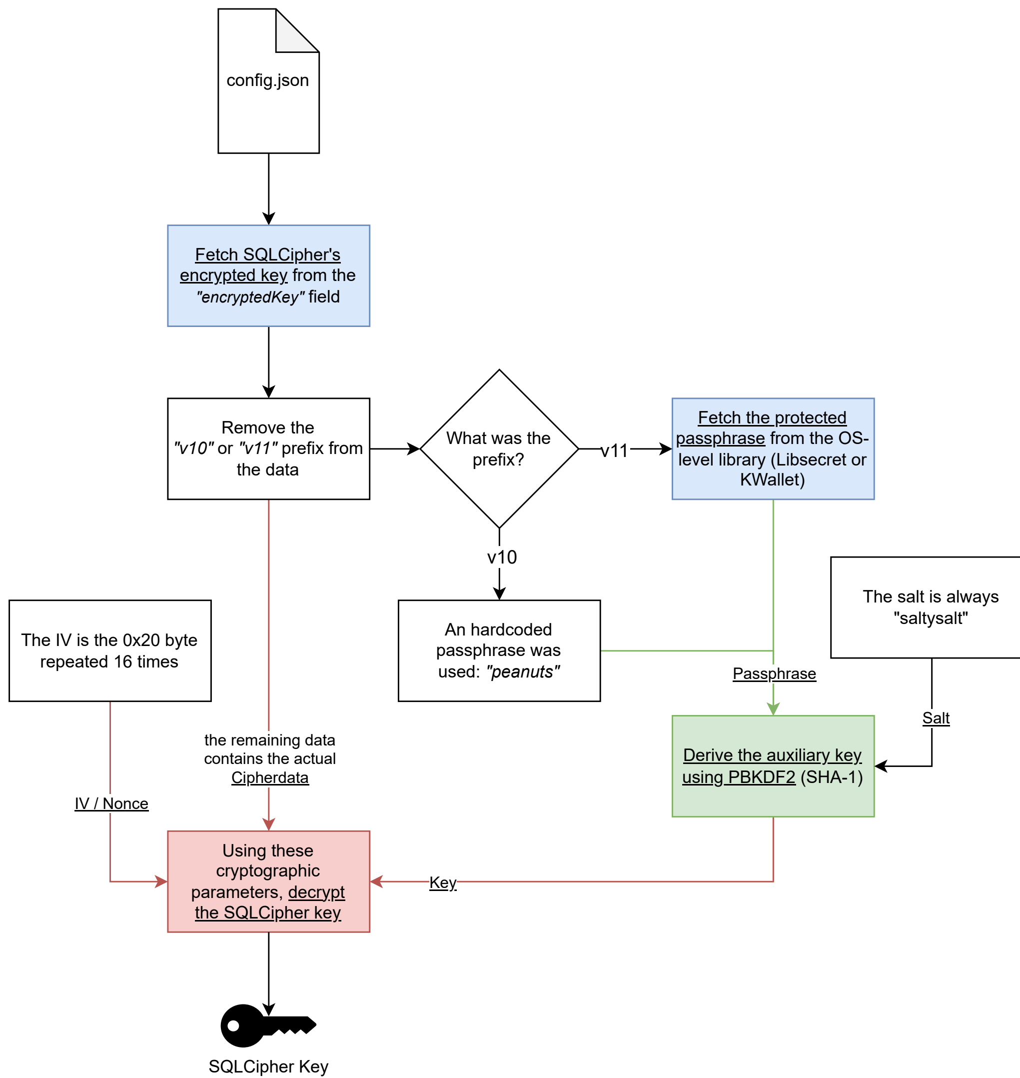
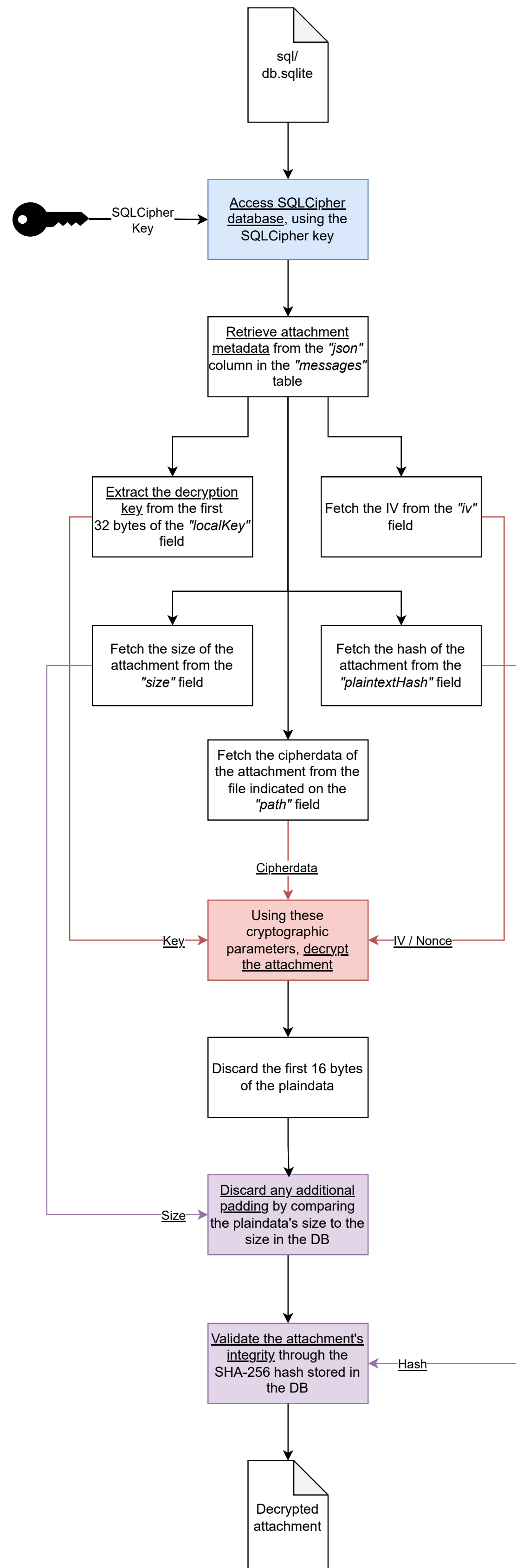


Figure 5.4: *Flowchart illustration of the decryption process for attachments and avatars.*



5.2.1 First Step: Decrypting SQLCipher Database

As mentioned in Related Work, Signal adopted Electron's safeStorage API to safely handle the SQLCipher key. Our analysis of the source code, found in Signal Desktop's GitHub repository[1], confirms that no further manipulation of the plain or cipher data is performed beyond this. Given this, we believe the following methodology is applicable to any Electron-based application utilizing the safeStorage API.

Signal retrieves the cipher data from the `encryptedKey` field in `config.json`, and then calls Electron safeStorage's `decryptString` function[2], passing the cipher data as a parameter. The `decryptString` function[3] validates the cipher data before invoking Chromium's `OSCrypt::DecryptString`. Chromium's OSCrypt implements encryption and decryption functions that leverage platform-specific cryptographic services such as the Data Protection API (DPAPI) on Windows and Keychain on macOS. In Linux environments, the configuration file additionally includes a `safeStorageBackend` field[4], which specifies the backend in use: `basic` for no integration, `gnome-libsecret` for Gnome Keyring, and `kwallet4`/`kwallet5`/`kwallet6` for KDE Wallet. Details on how these different backends affect the decryption process are provided in later sections. From this point onward, the decryption process diverges based on the operating system.

Windows

On Windows, the `DecryptString` function[5] starts by retrieving the base64-encoded auxiliary key cipher data from the `os_crypt.encrypted_key` field in `LocalState`[6]. Figure 5.5 exemplifies the content stored in this field after decoding it using `base64`.

After discarding the "DPAPI" prefix, the auxiliary key is decrypted through the DPAPI[7] and kept in volatile memory for future use. To replicate this step, we utilized NirSoft's DataProtectionDecryptor tool. With the auxiliary key, the `DecryptString` function decrypts the SQLCipher key using the AES-256 algorithm in Galois/Counter Mode (GCM)[5]. For this, it still requires two extra parameters, the nonce and the authentication tag. To acquire them, we must examine the data acquired from the `config.json` file. As we can ascertain in the source code and observe in Figure 5.6 and 5.7, the first three bytes are the `v10` prefix, the nonce is located in the following 12 bytes,

[1] https://github.com/signalapp/Signal-Desktop
[2] https://github.com/signalapp/Signal-Desktop/blob/ac09a508f1085c58afdeb6c758ccb22ab8ed8e51/app/main.ts#L1690C4-L1690C48
[3] https://github.com/electron/electron/blob/6e3a5daf62314d4de541d64a1098b0bb3a37168c/shell/browser/api/electron_api_safe_storage.cc#L118
[4] https://github.com/signalapp/Signal-Desktop/blob/203a1cc5e3f9c1533a58caff72e13aa6eaeeddc7/app/main.ts#L1606
[5] https://github.com/chromium/chromium/blob/7d17d557f6f637cbb4520f30297b1fde9e3cfc54/components/os_crypt/sync/os_crypt_win.cc#L216
[6] https://github.com/chromium/chromium/blob/7d17d557f6f637cbb4520f30297b1fde9e3cfc54/components/os_crypt/sync/os_crypt_win.cc#L111
[7] https://github.com/chromium/chromium/blob/7d17d557f6f637cbb4520f30297b1fde9e3cfc54/components/os_crypt/sync/os_crypt_win.cc#L294C8-L294C30

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   44 50 41 50 49 01 00 00 00 D0 8C 9D DF 01 15 D1   DPAPI....ÐŒ.ß..Ñ
00000010   11 8C 7A 00 C0 4F C2 97 EB 01 00 00 00 9E 0E B4   .Œz.ÀOÂ—ë....ž.´
00000020   9A 03 BF BA AB 95 11 A4 1E 0A F6 9F 57 10 00 00   š.¿º«•.¤..öŸW...
00000030   00 12 00 00 00 43 00 68 00 72 00 6F 00 6D 00 69   .....C.h.r.o.m.i
00000040   00 75 00 6D 00 00 00 10 66 00 00 00 01 00 00 20   .u.m....f......
00000050   00 00 00 A1 4C D0 E4 C5 66 90 73 3F FC F1 75 85   ...¡LÐäÅf.s?üñu…
00000060   06 75 16 14 76 42 C6 43 59 60 18 7F 6D 06 32 30   .u..vBÆCY`..m.20
00000070   FD 35 AF 00 00 00 00 0E 80 00 00 00 02 00 00 20   ý5¯.....€......
00000080   00 00 00 CB 81 DD 48 1E 2F E5 32 6E D3 4F DC 64   ...Ë.ÝH./å2nÓOÜd
00000090   0F B6 3E 7A 7A 7A 5D FC D0 5C 39 85 3A C3 00 A5   .¶>zzz]üÐ\9…:Ã.¥
000000A0   A7 F0 BD 30 00 00 00 5E 03 68 47 DA 08 2B 73 A5   §ð½0...^.hGÚ.+s¥
000000B0   B8 EF D9 F4 96 AD 72 9A 19 8D D2 01 A5 D4 23 03   ¸ïÙô–.rš..Ò.¥Ô#.
000000C0   1D CB 0C 0B 4D 4F 6D BD 17 CD 47 D2 69 5A 2C 82   .Ë..MOm½.ÍGÒiZ,‚
000000D0   CB 54 66 0E 65 03 6E 40 00 00 00 03 F5 E7 11 18   ËTf.e.n@....õç..
000000E0   57 D2 69 62 06 7B 03 E1 64 77 0A C8 C6 B6 E9 6B   WÒib.{.ádw.ÈÆ¶ék
000000F0   76 53 78 1D 5C 33 60 90 38 65 02 F4 B2 16 41 45   vSx.\3`.8e.ô².AE
00000100   F3 C0 94 C2 47 B2 B4 87 30 48 46 1A 71 36 9B 39   óÀ”ÂG²´‡0HF.q6›9
00000110   CA 9B 63 9B 63 9B 63 77 63 72 9D                  Ê›c›c›cwcr.
```

Figure 5.5: *The base64-decoded encrypted auxiliary key stored in the* `Local State` *file. (Windows)*

the actual cipher data occupies the rest of the data up until the last 16 bytes where the authentication tag is stored.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   76 31 30 A6 D5 5D 90 38 7D 7F 6A 42 7B 68 8C E8   v10¦Õ].8}.jB{hŒè
00000010   62 AB 8C 2C 16 BE 2F 83 27 FD B9 0F E2 9B C1 D7   b«Œ,.¾/ƒ'ý¹.â›Á×
00000020   53 14 D0 C9 52 5C 99 A2 1D 56 DB 38 4F 14 5C BC   S.ÐÉR\™¢.VÛ8O.\¼
00000030   28 62 84 AA FF 3E C3 F1 91 89 A1 4B D4 F0 42 43   (b„ªÿ>Ãñ'‰¡KÔðBC
00000040   42 38 79 2E 48 68 50 02 EA 93 6D 18 34 A4 38 C6   B8y.HhP.ê"m.4¤8Æ
00000050   90 14 BE 57 E4 43 5E 86 84 BC 92 28 55 92 EF      ..¾WäC^†„¼'(U'ï
```

[■] PREFIX | [■] NONCE | [■] GCM TAG

Figure 5.6: *The encrypted SQLCipher key stored in a* `config.json` *file. (Windows)*

Linux

On Linux, the `DecryptString` function[8] begins by identifying whether the cipher data of the SQLCipher Key is prefixed with `v10` or `v11`.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   76 31 31 CA 23 59 C2 05 CC D1 70 FE 7A 22 9E 9D   v11Ê#YÂ.ÌÑpþz"ž.
00000010   BB F4 D6 55 D4 68 68 8A DC 1C 58 04 3D B2 AD 32   »ôÖUÔhhŠÜ.X.=².2
00000020   65 18 7E D8 68 5D EA BA A8 81 AD 21 82 09 86 79   e.~Øh]êº¨..!‚.†y
00000030   28 2A D8 56 95 55 80 7D 5F 08 9E A9 1B 0C 73 85   (*ØV•U€}_.ž©..s…
00000040   CE 08 D0 60 DD D6 03 E2 C2 FF 9C AB B0 37 39 FA   Î.Ð`ÝÖ.âÂÿœ«°79ú
00000050   BE 9B 5A                                          ¾›Z
```

Figure 5.7: *The encrypted SQLCipher key stored in a* `config.json` *file. (Linux with Gnome Keyring)*

The `v11` prefix denotes that an OS-level library — either Libsecret or KWallet — is used during the decryption process. In such cases, the `DeriveV11Key` function[9] is invoked to retrieve a passphrase from the user's keyring and then use it to derive

[8] https://github.com/chromium/chromium/blob/1e1b1f608115db6ffceca1755cd5b3871c6c9f6e/components/os_crypt/sync/os_crypt_linux.cc#L157

[9] https://github.com/chromium/chromium/blob/1e1b1f608115db6ffceca1755cd5b3871c6c9f6e/components/os_crypt/sync/os_crypt_linux.cc#L296

the auxiliary key. The auxiliary key is computed using the PBKDF2 function[10] which, as the name implies, applies Password-Based Key Derivation Function 2 (PBKDF2) with the retrieved passphrase as input, `saltysalt` as the salt, 16 bytes as the key size, SHA-1 as the hashing algorithm and a single iteration. The result is the auxiliary key. If the cipher data is prefixed with `v10`, it indicates that no OS-level secret storage — Libsecret or KWallet — is available. In this scenario, the `DecryptString` function[8] uses a hardcoded passphrase — `peanuts` — along with the same `saltysalt` salt to derive the auxiliary key.

In both scenarios, the prefix is stripped, and the SQLCipher key is decrypted using the AES-128 algorithm in Cipher Block Chaining (CBC) mode[8]. The derived auxiliary key is used as the decryption key, and a 16-byte nonce consisting of the space character (`0x20`) is used as the Initialization Vector (IV).

5.2.2 Second Step: Decrypting Attachments and Avatars

Using the decrypted SQLCipher key, it is now possible to access and query Signal's local database through tools like DB Browser for SQLCipher. While Signal handles the decryption of attachments and avatars separately, their decryption processes are nearly identical. For the sake of simplicity, we will first describe the more complex attachment decryption process and then highlight the key differences with avatar decryption. Note that internally, view-once messages have their media handled as a normal attachment, following the exact same procedure for the decryption of their contents.

The table `messages`, as the name suggests, stores information on the messages the user exchanged. Messages containing attachments have the `hasAttachments` field set to 1. The `json` field holds the message data in JSON format. This field will be our focus moving forward. Within the JSON key `attachments`, we find an array that stores details about each attachment included in the message. As illustrated in Listing 4, the `path` field provides the path to the encrypted attachment file, while the `localKey` and `iv` fields contain the base64-encoded decryption key and nonce (IV), respectively. Also relevant are the `size` and `plaintextHash` fields — the `size` field specifies the attachment's size in bytes, while the `plaintextHash` contains the SHA-256 hash of the decrypted attachment.

Signal encrypts[11] attachments using the AES-256 algorithm in the CBC mode. The data extracted from `localKey` is 64 bytes long; the AES-256 decryption key is located in the first 32 bytes. After decrypting the attachment's cipher data, the first 16 bytes of the decrypted data must be discarded, along with any additional padding. Trailing bytes that surpass the attachment size, as indicated by the `size` field, are to be discarded to establish the precise length of padding requiring removal. Finally, to guarantee the integrity of the attachment, the SHA-256 hash of the remaining data is compared with

[10] https://github.com/chromium/chromium/blob/1e1b1f608115db6ffceca1755cd5b3871c6c9f6e/components/os_crypt/sync/os_crypt_linux.cc#L288

[11] https://github.com/signalapp/Signal-Desktop/blob/94dba11bcb3973b98784b0a7aaefa6ea03c330ea/ts/AttachmentCrypto.ts#L380

Listing 4: *An example excerpt of attachment information stored in the `json` column of the `messages` table.*

```json
{
  "size": 543394,
  "contentType": "audio/mpeg",
  "localKey": "pDFricod8L7qAzVJuAMCHfVHyFCF/kDGyfGBYPMWKg6SZpfAIw/4OiqlNSWtvVTZCcFcN3Ty
  ↪  JGKSO8UUck4e4w==",
  "iv": "Z78XQDc6YwjPpLbyVzSz0w==",
  "path": "c1\\c24c12ebd76a783a547d74ddbf8d81376823bc21fe10d4183f3c87841d744841",
  "plaintextHash": "0be6c47276804a39e30dd2674f8b26eb3f51bab3efee4f1bf715e2fa18b15e34",
}
```

the value in the `plaintextHash` field.

For draft attachments and avatars, the decryption process is largely similar to that of exchanged attachments, having two key differences. The first difference is that their cryptographic metadata is stored in the `json` field of the `conversations` table, specifically in the record corresponding to the conversation to which they pertain.

- The `draftAttachments` key contains an array with the cryptographic details of all draft attachments.
- The `avatar` key stores the cryptographic metadata of the current avatar for the conversation.
- The `avatars` key is populated only in group chats and the "Notes to self" conversation. This key maintains a list of cryptographic information for all avatars that the user has uploaded for the group or themselves.

The second difference is that none of these entries include the `iv` field. Instead, a 16-byte array of `0x00` bytes should be used as the IV during decryption.

5.3 Forensically Extracting Data from Gnome Keyring

In this section, we describe the process used to extract data from the Gnome Keyring file. This process is crucial, for example, to retrieve the passphrase required to derive the auxiliary key, which is then used to decrypt the SQLCipher key in SignalForensics' Forensic Mode (as described in Chapter 7). Notably, this extraction process is not specific to Signal-related data; it can be universally applied to retrieve any stored secrets from the Gnome Keyring. The version of **Gnome Keyring** analyzed in this section is **46.1**, released on February 13, 2024.

Before delving deeper into the subject, it is important to establish the prerequisites for this process: the keyring's master password and the keyring file itself.

> **Note**
>
> While obtaining a full copy of the keyring file is recommended, it may still be possible to extract secrets from a partial copy, provided that the necessary parameters to traverse the file and the relevant portions of the secret are not missing.

The statements and findings presented in this section are based on the analysis of the Gnome Keyring file structure's documentation[12], and have been validated through hands-on testing.

5.3.1 Gnome Keyring File Structure

The Gnome Keyring file is a binary file that contains a series of encrypted records, each representing a secret. In this section, we will describe the structure of the Gnome Keyring file and how to extract and decrypt the information it stores. While we do not provide detailed explanations for all components of the file, we indicate how to skip over irrelevant sections and focus our analysis on the parts necessary for forensic extraction. We can divide the Gnome Keyring file into three main sections, each corresponding to a distinct stage in the extraction process:

1. **File Header:** This section contains keyring metadata and parameters required for key derivation.
2. **Attributes' Metadata:** This section contains a list of attribute names and hashed values associated with the encrypted secrets that are stored in the following section of the file.
3. **Encrypted Data:** This section contains the structured encrypted secrets, which are the actual data we want to extract.

Figure 5.8 illustrates a hexadecimal view of the keyring file, annotated with descriptions to help visualize the file structure.

File Header The file begins with a 16-byte magic string: `GnomeKeyring\n\r\0\n`. This serves as a signature identifying the file as a Gnome Keyring file and can be used to verify the file's format.

Following the magic string, we skip the next 4 bytes. Immediately after, we encounter the first variable-length string, which in this case corresponds to the keyring's name. All strings in the keyring file follow the same encoding structure: a 4 byte big-endian unsigned integer indicating the string's length, followed by the UTF-8 encoded string itself. If the length is `0xFFFFFFFF`, the string is considered empty, and only the first 4-byte field is present.

After reading the keyring's name, we skip 24 bytes, after which we find:

[12] https://wiki.gnome.org/Projects(2f)GnomeKeyring(2f)KeyringFormats(2f)FileFormat.html

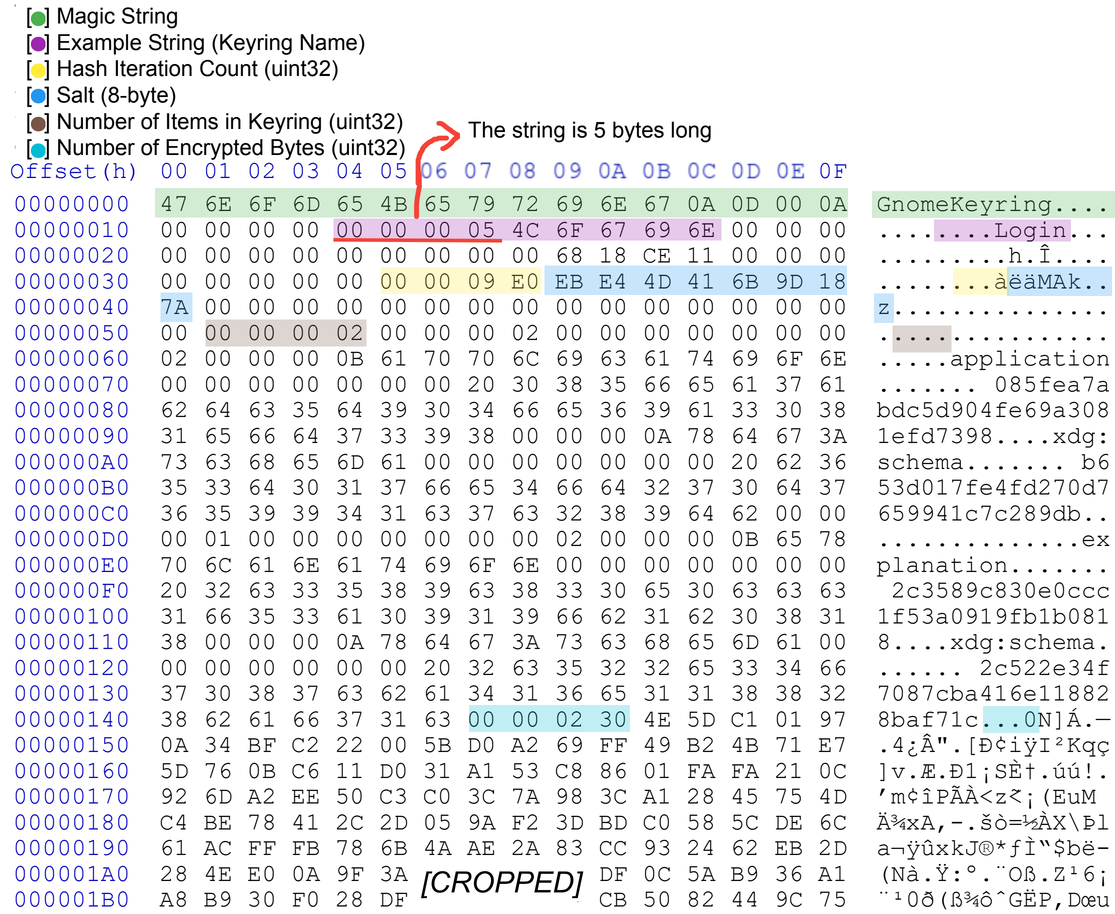


Figure 5.8: *Hexadecimal view of a sample Gnome Keyring file.*

- A 4-byte big-endian unsigned integer representing the hash iteration count.
- An 8-byte salt value used in the key derivation process.

We then skip the next 16 bytes, which are not relevant for our extraction process.

Attributes' Metadata The second major section of the Gnome Keyring file contains metadata associated with each stored secret. It lists the names and hashed values of the attributes associated with the secrets. For our extraction purposes, we do not need to delve deeply into this section, but we will outline the necessary steps to skip it.

The first 4 bytes are an unsigned integer indicating the number of items in this section. For each item, we skip the first 16 bytes, then read a 4-byte unsigned integer that tells us the number of attributes associated with this item. For each of these attributes, we skip a string — the attribute's name — and then read a 4-byte unsigned integer indicating the attribute's type. Depending on the type, we then skip the attribute's hashed value in the following way:

- If the type is `0`, we skip a string hash, which is encoded as described earlier.
- If the type is `1`, we skip the next 4 bytes, which represent the integer value.

Once all items and their attributes have been processed, we reach the end of this section.

Encrypted Data The third section of the Gnome Keyring file contains the encrypted secrets. It begins with a 4-byte unsigned integer indicating the number of encrypted bytes in the file. The subsequent bytes are the encrypted data, which is structured according to the keyring's specifications.

Key Derivation and Decryption To extract the stored secrets, the encrypted data must be decrypted using a derived key. This key derivation process uses the information gathered from the file header — *i. e.*, the salt and hash iteration count — and the keyring's master password. This key derivation function is performed using the following parameters:

- **Key derivation function:** OpenSSL's `EVP_BytesToKey` function.
- **Hash function:** SHA-256.
- **Password (or input bytes):** The keyring's master password.
- **Salt:** The 8-byte value extracted from the file header.
- **Iteration count:** The 4-byte value extracted from the file header.
- **Derived key length:** 16 bytes.

The resulting key is then used in AES-128-CBC decryption, with an Initialization Vector (IV) consisting of 16 zero bytes (`0x00`). After decryption, the resulting plaintext must be parsed to extract individual secrets.

5.3.2 Extracting Secrets from the Decrypted Data

The decrypted payload begins with 16 bytes, which appear to be a hash used to verify the integrity of the decrypted data. However, no documentation was found describing its use, and our attempts to validate or interpret this field through testing were inconclusive. For our purposes, we can skip these 16 bytes and proceed to the next section.

The decrypted structure contains a sequence of items that is equal in number to those described in the Attributes' Metadata section of the Gnome Keyring file. Each item begins with its display name stored as a string, followed by another string that contains the actual secret. This may already be the secret of interest. However, display names are often ambiguous. For instance, in the case of Signal Desktop — or any Electron-based application using Electron's safeStorage API — the display name will always be `ChromiumSafeStorage`. To distinguish between different sources, we must examine the item's attributes. To do so, for each item, we skip 16 bytes, a string, and then another 16 bytes. Next, we read a 4-byte unsigned integer indicating the number of attributes present. Each attribute then includes:

- A string storing the attribute's name.
- A 4-byte unsigned integer indicating the attribute's type.
- A value, which is either a string (if the type is `0`) or a 4-byte unsigned integer (if the type is `1`).

For example, in the case of Signal, there should be an attribute named `application` with the value `Signal`. If this attribute is present, the secret previously extracted is the one we are looking for — the passphrase required for deriving the auxiliary key in Signal Desktop's decryption process. If not, we must continue parsing the item until we reach the end, and then move on to the next item. For that, we read a 4-byte unsigned integer indicating the Access Control List (ACL) length. Then, for each ACL entry — when one exists — we skip 4 bytes, 3 strings, and then 4 more bytes. This completes the parsing for one item. Repeat the process for each item until the desired secret is located or until all items have been processed. Figure 5.9 illustrates the structure of an example decrypted payload from a Gnome Keyring File.

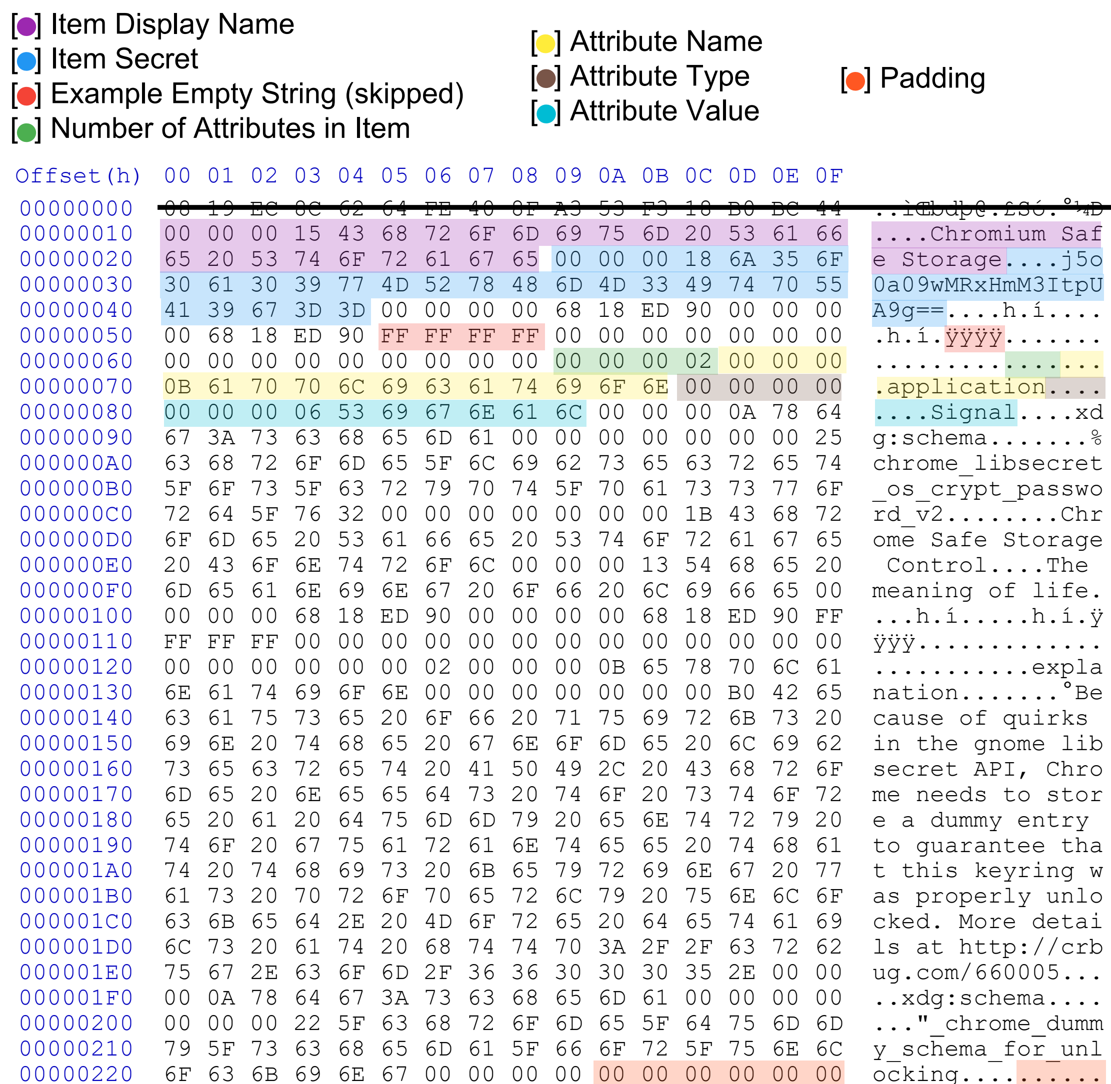


Figure 5.9: *Decrypted payload from an example Gnome Keyring file.*

5.3.3 Example Implementation

Appendix A presents a Python code snippet demonstrating how to extract a secret from a Gnome Keyring file. The example is a simplified version of the implementation integrated into SignalForensics.

5.4 Forensically Extracting Data from KDE Wallet

In this section, we describe the process of extracting data from a KDE Wallet (KWallet). This process is crucial, for example, to retrieve the passphrase required to derive the auxiliary key, which is then used to decrypt the SQLCipher key in SignalForensics' Forensic Mode (as described in Chapter 7). It should be noted, however, that this process is not limited to Signal-related data; it can also be applied to recover any secret stored within a KWallet. The version of **KWallet** analyzed in this section is **5.115.0**, released in February 10, 2024.

The extraction process varies slightly depending on whether the wallet is encrypted with Blowfish or protected using a GPG key.[13] For this reason, when necessary, we explicitly distinguish steps that differ between encryption schemes.

Before delving deeper into the process, it is important to establish its prerequisites: the wallet's master password and the KWallet file itself. In the case of Blowfish-encrypted wallets, the corresponding salt file is also required. For wallets protected with a GPG key, access to the private GPG key is necessary.

The statements and findings presented here are based on an analysis of KWallet's source code[14] and the OpenPGP message format specification[15], and have been validated through hands-on testing.

5.4.1 KWallet File Structure

The KWallet file is an encrypted binary container that stores the wallet's contents, including multiple secrets. The wallet is organized into internal folders, each of which contains entries holding the actual secret values. As with the analysis of the Gnome Keyring structure in the previous section, we do not provide a complete explanation of every component; instead, we indicate how to bypass irrelevant sections and focus only on the parts essential for forensic extraction.

The file can be divided into three main sections:

1. **File Header:** This section contains the magic string identifying the file as a KWallet and metadata specifying the algorithms used for encryption.
2. **Hashes Section:** This section contains hashed folder names and hashed entry names associated with the encrypted secrets stored in the following section.

[13] GNU Privacy Guard (GPG) is an implementation of the OpenPGP standard that provides encryption and signing for data and communications.
[14] https://github.com/KDE/kwallet/
[15] https://www.rfc-editor.org/rfc/rfc9580.html

3. **Entries Section:** This section contains the entries' secrets, which are the data of interest for extraction.

Figure 5.10 illustrates a hexadecimal view of a decrypted KWallet file, annotated with descriptions highlighting the different sections.

```
[●] Magic String            [●] Folder Count
[●] Version                 [●] Folder Name
[●] Encryption Algorithm    [●] Entry Name
[●] Hashing Scheme          [●] Entry Value

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   4B 57 41 4C 4C 45 54 0A 0D 00 0D 0A 00 01 03 02   KWALLET.........
00000010   00 00 00 01 E8 46 33 16 D2 0B AD 7A 46 7D CD F1   ....èF3.Ò..zF}Íñ
00000020   3E 90 50 1B 00 00 00 01 02 0B 44 83 88 EF 0B A4   >.P.......Dƒˆï.¤
00000030   4B C6 C8 54 2D E4 26 92 00 00 00 1A 00 43 00 68   KÆÈT-ä&’.....C.h
00000040   00 72 00 6F 00 6D 00 69 00 75 00 6D 00 20 00 4B   .r.o.m.i.u.m. .K
00000050   00 65 00 79 00 73 00 00 00 01 00 00 00 2A 00 43   .e.y.s.......*.C
00000060   00 68 00 72 00 6F 00 6D 00 69 00 75 00 6D 00 20   .h.r.o.m.i.u.m. 
00000070   00 53 00 61 00 66 00 65 00 20 00 53 00 74 00 6F   .S.a.f.e. .S.t.o
00000080   00 72 00 61 00 67 00 65 00 00 00 01 00 00 00 34   .r.a.g.e.......4
00000090   00 00 00 30 00 35 00 4E 00 74 00 31 00 31 00 66   ...0.5.N.t.1.1.f
000000A0   00 4D 00 55 00 66 00 5A 00 43 00 44 00 74 00 43   .M.U.f.Z.C.D.t.C
000000B0   00 4D 00 34 00 65 00 59 00 55 00 53 00 71 00 67   .M.4.e.Y.U.S.q.g
000000C0   00 3D 00 3D                                       .=.=
```

Figure 5.10: *Hexadecimal view of a decrypted KWallet file.*

The Entries Section is always encrypted, regardless of the encryption scheme. However, the Hashes Section is only encrypted when GPG is used; in such cases, both sections are encrypted together as a single block. Therefore, the decryption steps differ slightly depending on the scheme. These differences will be made explicit in the following subsections.

File Header The file begins with a 12-byte magic string, `KWALLET\n\r\0\r\n`, which serves as a signature identifying the file type. This allows examiners to quickly verify that the file is indeed a KWallet.

The next 2 bytes indicate the KWallet version and can be skipped. The following 2 bytes specify, respectively, the encryption algorithm used to protect the wallet and the hashing scheme employed to derive the cipher key. The algorithm byte is mapped as follows:

- **0x00** — Blowfish in Electronic Code Book (ECB) mode
- **0x01** — 3DES in CBC mode (now unsupported)
- **0x02** — GPG
- **0x03** — Blowfish in CBC mode

The hash byte is mapped as follows:

- **0x00** — SHA-1 (deprecated since KDE Wallet 4.13)
- **0x01** — MD5 (now unsupported)
- **0x02** — PBKDF2 with SHA-512

If the algorithm is GPG (`0x02`), the hash byte is always `0x00`, as no key derivation is required. In modern wallets, one should expect to encounter either Blowfish in CBC mode with PBKDF2-SHA512 (`0x0302`) or GPG (`0x0200`). These are the configurations we focus on, while also briefly noting the small difference between Blowfish ECB and CBC modes.

Hashes Section This section stores metadata that maps folders and entries without exposing their names directly. It begins with a 4-byte unsigned integer indicating the number of folders in the wallet. For each folder, we first skip 16 bytes corresponding to the MD5 hash of the folder name, followed by a 4-byte unsigned integer that specifies the number of entries within that folder. For each entry, we again skip 16 bytes corresponding to the MD5 hash of the entry name.

Entries Section This section contains the actual contents of the wallet, organized by folders and their corresponding entries. Each folder is stored as a variable-length string containing the folder name, followed by a 4-byte unsigned integer indicating the number of entries it contains. Variable-length strings in KWallet are encoded as follows: a 4-byte unsigned integer specifies the string length in bytes, followed by the big endian UTF-16 encoded string itself. If the string is empty, the length field will consist of either four bytes of `0x00` or four bytes of `0xFF`, and no contents will follow.

For each entry within a folder, we first skip a variable-length string containing the entry name, then read a signed 4-byte integer that specifies the entry type. Next, we read a 4-byte unsigned integer that indicates the length of the entry's value field, regardless of its type. At this stage, since both the folder and entry names are already available, we can decide whether to parse the value. If the entry is irrelevant, we can save processing effort by skipping forward by the exact number of bytes specified in the length field. For example, in the context of Signal Desktop we specifically look for a folder named "Chromium Keys" and an entry named "Chromium Safe Storage".

If this is the entry of interest, we proceed to parse its value. The parsing procedure depends on the entry's type, which is mapped to its corresponding integer as follows:

- **0x01** — Passwords
- **0x02** — Binary Data
- **0x03** — Maps

In the following parts, we assume the 4-byte length field delimiting the value has already been read and skipped, so that parsing can begin directly with the contents of the entry's value.

Type 0x01: Passwords This type is used to store simple string values, such as passwords or tokens. The value begins with a 4-byte unsigned integer indicating the length of the password in bytes, followed by the UTF-16 encoded string contents. In

the context of Signal Desktop, the passphrase required to derive the auxiliary key is stored in a password entry.

Type 0x02: Binary Data This type is used to store raw binary data. The value begins with a 4-byte unsigned integer indicating the size of the binary data, followed by the corresponding bytes. If the length is set to `0xFFFFFFFF`, the entry is empty, meaning that no bytes follow and the next entry starts immediately.

Type 0x03: Maps This type is used to store key–value pairs, where both the keys and values are strings. The value begins with a 4-byte unsigned integer indicating the number of pairs in the map. Each pair is then stored as two variable-length strings: the key followed by the value. Similarly to the binary data, if a string field is empty the length is set to `0xFFFFFFFF`, and no string contents are present.

5.4.2 Decrypting KWallet Data

The first step in decrypting the contents of a KWallet file is determining which sections are encrypted and which remain in plaintext. As previously discussed, this depends on whether the wallet was protected using Blowfish or GPG.

Blowfish

In wallets protected with Blowfish, only the Entries Section is encrypted. The first step is to retrieve the folder count from the Hashes Section, since this value is not repeated or stored in the Entries Section, but will be required for later processing. After traversing the entire Hashes Section, the remaining contents of the file correspond to the ciphertext of the Entries Section.

Key Derivation With the ciphertext extracted, all necessary data from the KWallet file has been obtained. The next step is to derive the key required to decrypt these contents. Modern KWallet implementations using Blowfish rely on the PBKDF2 key derivation function with SHA-512. This function requires four parameters: the password, the salt, the number of iterations, and the key length.

The password corresponds to the wallet's master password encoded in UTF-8. The salt is stored in the wallet's salt file, located in the same directory as the wallet, under the same name but with the `.salt` extension. The full contents of this file constitute the salt value. The number of iterations is fixed at 50 000, and the derived key length is 56 bytes.

With these parameters, the wallet's decryption key can be derived.

Decryption For wallets using Blowfish in CBC mode, decryption is performed with the derived key and an Initialization Vector (IV) of eight bytes set to `0x00`. After decryption, the first 8 bytes of the plaintext must be discarded, as they correspond to a

block of random padding introduced during encryption to conceal repeated plaintext patterns when the IV is fixed.

In contrast, wallets encrypted in ECB mode do not use an IV, and therefore there is no need to discard the first 8 bytes of the decrypted data.

GPG

In wallets protected with a GPG key, both the Hashes Section and the Entries Section are encrypted together as a single payload. This means that the entire file, except for the initial 16-byte header, constitutes the encrypted data.

To unlock the wallet, the corresponding GPG private key must be used, which in most cases will be the user's default key. Since GPG private keys are themselves typically protected by a passphrase, this user-supplied secret is also required in order to access the wallet's contents.

Decryption can be carried out, for instance, with the gpg command-line tool:

```
1    gpg --output decrypted.bin --decrypt cipherdata.bin
```

Once decrypted, the contents can be processed in the same way as described in Section 5.4.1, with one important caveat. Because both sections are encrypted together, each one is encapsulated: before the actual data of the Hashes Section and the Entries Section, there is a 4-byte unsigned integer indicating that section's size. This detail can be advantageous, since it allows, for example, skipping the Hashes Section entirely in a single step if it is not required for analysis.

Chapter Summary

This chapter provided a detailed overview of Signal Desktop's local file structure and the methods used to decrypt its artifacts. It explained the locations of key files across Windows and Linux, the SQLCipher database decryption process, and the methodology for decrypting attachments, avatars, and draft message data. Additionally, the chapter described how to extract cryptographic secrets from Gnome Keyring and KDE Wallet, detailing file structures, key derivation, and decryption procedures.

6
Forensic Artifacts

In this chapter, we detail the main forensic artifacts of Signal Desktop. To populate the application's database with realistic content, we simulated user activity by exchanging messages between two devices: a Windows 11 desktop running Signal Desktop and an Android 13 smartphone running Signal. Both applications were updated to the most recent available version at the time of testing. The exchanged messages included plain text (including text formatting and mentions), images, audio recordings (*i. e.*, voice messages and MP3s), videos, and other files. Additionally, features such as group creation, message deletion, story posting, and calls were tested to ensure the generation of diverse and representative forensic artifacts.

Most of the valuable forensic artifacts identified during our research were located within Signal Desktop's SQLite database. As delineated in Chapter 5, this database is encrypted at rest using SQLCipher and resides in the folder created by Signal within the Windows user's Roaming directory, specifically at the relative path `sql/db.sqlite`.

The database contains 45 tables, 76 indices and 6 triggers. In this chapter, we focus on the tables and associated artifacts that are forensically relevant: `callsHistory`, `conversations`, `items`, `mentions`, `messages` and `reactions`. Table 6.1 gives a brief description of each of these tables. The relationships between tables in the database are illustrated as an Entity-Relationship diagram using IDEF1X[1] notation in Figure 6.1. For brevity, tables without any defined relationships have been omitted. A complete printout of the database schema — *i. e.*, all tables, indices, views, and triggers — is available in Appendix B. Unless otherwise stated, all timestamps mentioned in this section are in Unix epoch time milliseconds and Universal Time Coordinated (UTC).

6.1 Items Table

The `items` table stores a wide range of information, including user settings such as enabling or disabling specific features, metadata related to blocked groups and users,

[1] https://www.essentialstrategies.com/publications/modeling/idef1x.htm

Table 6.1: *Brief description of relevant tables in the SQLCipher database.*

Name	Nr. of Columns	Description
`callsHistory`	10	Audio and video call records of the current user.
`conversations`	14	Conversations of the current user.
`items`	2	Current user and application settings.
`mentions`	4	Mentions in the exchanged messages.
`messages`	44	Exchanged messages.
`reactions`	9	Emoji reactions to exchanged messages.

as well as information on the current user – *e. g.*, ACI and phone number (in E.164 format). The `id` column stores the identifier of the attribute recorded in the `json` column, which can represent user settings, metadata, or account-related information. The `json` column contains a JSON object with two fields: `id` and `value`. The `id` field mirrors the value in the `id` column, while the `value` field holds the corresponding data. This data can take various forms, including strings, lists, base64-encoded content or objects.

During our research and tests, we identified 84 distinct records in the `items` table. Table 6.2 shows the most relevant entries from a forensic perspective — *i. e.*, attributes that reflect user configuration, identity, or interactions with other users.

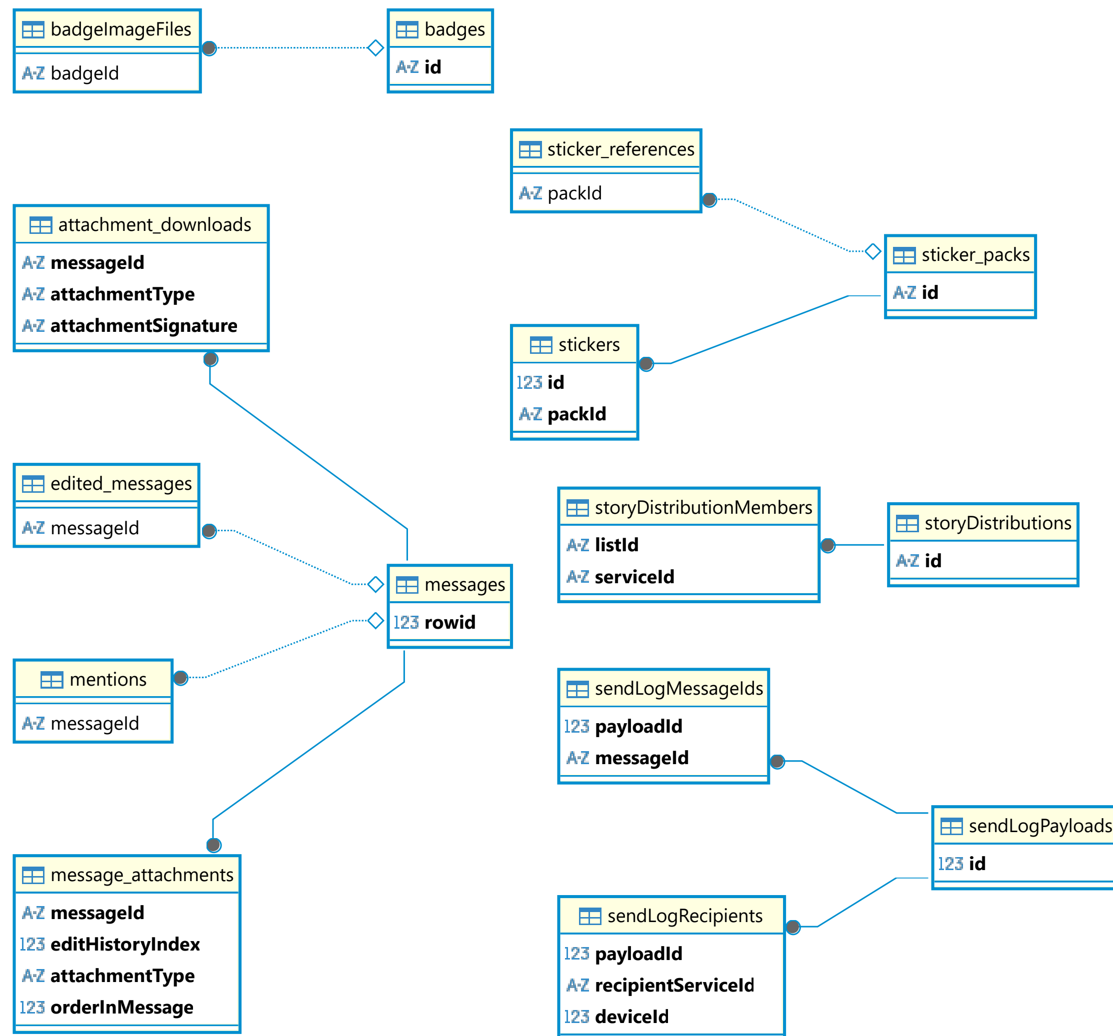
Table 6.2: *List of relevant attributes found in the database's items table.*

Attribute	Type	Contents
`blocked`	List	The phone numbers (E.164) of users blocked by the current user.
`blocked-groups`	List	The IDs of groups blocked by the current user.
`blocked-uuids`	List	The ACIs of users blocked by the current user.
`device_name`	String	The name of the device Signal Desktop is installed on.
`hasStoriesDisabled`	Boolean	True if the user disabled the stories functionality.
`number_id`	String	The phone number (E.164) of the current user.
`pni`	String	The PNI of the current user.
`read-receipt-setting`	Boolean	True if the current user has read receipts enabled.
`typingIndicators`	Boolean	True if the current user has typing indicators enabled.
`unreadCount`	Integer	Total number of unread messages throughout all conversations.
`uuid_id`	String	The ACI of the current user.
`version`	String	The version of Signal Desktop that last accessed this database.

6.2 Conversations Table

The `conversations` table stores information about the user's conversations. The `json` column stores the conversation's metadata as a JSON string. The remaining columns serve auxiliary roles, repeating key details from the JSON object, such as the conversation's name and type. This design likely facilitates the execution of SQL operations on conversation data without the need to parse the JSON.

Figure 6.1: *ER diagram of the SQLCipher database schema using IDEF1X notation. (Tables without relationships were omitted for brevity)*



After parsing the JSON object, we encounter various fields. In this section, we will only note the most interesting ones. Some of these fields are present in private conversations, others in group conversations, but most are present in both. There is still a possibility that the presence of a few of these fields might vary on a stricter case-by-case basis (*e. g.*, a draft message existing or not); for that reason, caution is recommended when parsing these objects.

Table 6.3 details the fields that are expected in every conversation. Table 6.4 details the fields that are expected to be present in private conversations. Finally, Table 6.5 details the fields expected in group conversations. Be sure to check footnotes [2] and [3] for additional context in some of the fields' descriptions.

Besides `profileAvatar` and `avatar`, there is a third field where information on

[2] This includes messages that have expired or were deleted.

[3] Given the aforementioned fields interact and complement each other similarly to how they do in a standard message, detailed explanations will be deferred to the messages section later in this document.

Table 6.3: *Conversation fields that are expected in every conversation.*

Name	Content
id	The UUID of the conversation.
type	The type of conversation.
name	The name of the conversation (either the contact's name or the group's name).
unreadCount	The number of unread messages.
messageCount	The total number[2] of messages that were exchanged.
sentMessageCount	The total number[2] of messages sent by the current user.
active_at	The timestamp for the last activity.
lastMessageReceivedAtMs	The timestamp of the last exchanged message.
lastMessageAuthor	The name of the author of the last message. When the last message was sent by the current user, this field will have a localized version of the word "You".
lastMessage lastMessageBodyRanges	The contents[3] of the last message.
lastMessageDeletedForEveryone	A flag that describes whether the last message was deleted for everyone — *i.e.*, is a "this message was deleted" notice.
lastMessagePrefix	An emoji representing the content of the last message (*e.g.*, a microphone emoji for voice messages). This field is not present on text messages.
draftTimestamp	The timestamp of the draft message, when one exists.
draft draftBodyRanges draftAttachments	The contents[3] of the draft message.
expireTimer	The current expire timer for this conversation. When an expire timer is defined, new messages exchanged in a conversation expire (and are deleted) after the number of seconds in this field.
isArchived	A flag that describes whether the conversation is archived.

Table 6.4: *Conversation fields that are expected in private conversations.*

Name	Content
serviceId	The Service ID of the contact which, as described in Section 3.2, is either the contact's PNI or ACI.
profileName and profileFamilyName	First and last name of a contact. This is the name the contact has set for their Signal account.
systemGivenName and systemFamilyName	First and last name of a contact. This is the contact name found in the current user's external contact book.
nicknameGivenName and nicknameFamilyName	The nickname the current user gave to this contact.
note	A note the current user wrote in the profile of this specific contact. A note is only visible to the user who wrote it.
e164	The phone number of the contact in E.164 format.
username	The username of the contact in Signal.
profileAvatar	The cryptographic parameters for the contact's avatar.

Table 6.5: *Conversation fields that are expected in group conversations.*

Name	Content
`groupId`	The ID of the group. This ID is 32 bytes of random data represented as a base64-encoded string.
`description`	The description of the group.
`membersV2`	A list containing information about the group's members. Each member is saved as an object that contains their ACI in the `aci` field, and their role as an integer in the `role` field. Currently, there are only two roles in Signal, `1` for "Member" and `2` for "Administrator".
`bannedMembersV2`	A list containing information about former group members that were kicked by an Administrator. Each member object contains a `serviceId` field with their Service ID.
`avatar`	The cryptographic parameters for the groups's avatar.

older avatars can be found: the `avatars` field. This field is only present in group conversations and in the "Notes to self" private conversation. It contains a list of objects that, along with simple icons[4], stores the cryptographic parameters of avatars the current user has uploaded for the group (if on a group conversation) or previous user avatars (if on the "Notes to self" conversation). These are the avatars that pop-up as options when the current user attempts to change their or a group's avatar. The avatars' decryption process is detailed in Section 5.2.2.

6.3 Calls History Table

The `calls_history` table logs the user's call history.

This table includes four key ID columns: `callId`, `peerId`, `ringerId`, and `startedById`. The first serves as the table's primary key. The data stored in the latter three tables varies based on the call mode and direction. Signal supports two call modes — "Direct" and "Group" — which are identified by the `mode` column. In direct calls, which occur between two users, the `peerId` represents the other participant. In group calls, which are initiated within a group chat, the `peerId` corresponds to the group's unique `groupId`. The direction of the call, recorded in the `direction` column, indicates whether it was outgoing — initiated by the user — or incoming — received by the user. In incoming calls, the `startedById` stores the ACI of the user who initiated the call, while the `ringerId` contains the ACI of the account that last rang the user – in incoming direct calls this column is redundant. In outgoing group calls, both the `startedById` and `ringerId` columns store the ACI of the current user; however, for outgoing direct calls, both columns are left empty.

The `status` column records the current status of a call. An example is the `Deleted` status, which indicates that the call notification message was removed or expired from the chat. Certain call statuses have specific meanings depending on the call mode and direction. For instance, a group call with an outgoing direction and a `Missed`

[4] Icons that are made up of a symbol and a foreground color.

status implies that none of the group members accepted the call. However, an incoming group call with the same status signifies only that the current user did not join, even though other members might have. Additionally, some statuses are exclusive to specific call modes. Only direct calls can have a `Pending` status, whereas only group calls may include statuses such as `Ringing`, `OutgoingRing`, `Joined`, and `GenericGroupCall`.

The remaining two columns — `timestamp` and `endedTimestamp` — store chronological data. The `timestamp` column records the timestamp of when the call was placed. The `endedTimestamp` column does not appear to be implemented, as it was consistently empty in all the tests we conducted.

6.4 Messages Table

The `messages` table records information about exchanged messages, including metadata, content, cryptographic metadata of attachments, read and view states, reactions, and edit history. In current releases of Signal Desktop, only part of this information is embedded in the `json` column; unlike the `conversations` table, many fields now reside exclusively in their own columns. While this behavior change has been confirmed only for the `messages` table at the time of writing, future versions may extend this pattern to other tables. Consequently, examiners must inspect both the JSON object and the individual columns to obtain a complete record and stay alert to further changes. Besides the raw message data, the `messages` table includes auxiliary flags such as `hasAttachments` and `hasFileAttachments`, which facilitate filtering for messages with attachments without the need to parse the JSON object.

Table 6.6 contains key elements within the JSON object that are either universal across all message types, inherently self-explanatory, or straightforward to explain.

Besides the mentioned fields, a message contains several other timestamps, such as `serverTimestamp`, `decrypted_at`, and `received_at`. We chose `received_at_ms` as the standard timestamp because it proved to be the most consistent across different message types and functionalities.

Before examining the contents of the JSON object in detail, it is important to understand how some of the data is structured and related. For instance, message text content is stored in the `body` field, but additional details like mentions and text formatting (*e. g.*, bold and italics) are found in the `bodyRanges` field, requiring both fields to be combined for a complete interpretation. The following subsections outline these relationships and their significance.

6.4.1 Message Types

The message type can be inferred from the contents of the `type` column or `type` field in the JSON object. There are 21 possible message types Signal Foundation (2025c). However, during our tests, we only observed seven of these types. Of these, five are noteworthy, specifically:

Table 6.6: *Key elements in a message's JSON object. Fields marked with an ∗ denote they are present in all messages.*

Name	Content
`id`*	The ID of the message.
`type`*	The type of message.
`conversationId`*	The UUID of the conversation the message belongs to.
`sourceServiceId`*	The Service ID of the message's author.
`sourceDevice`	An integer — *e. g.*, 1, 2, 3, etc. — representing from which of the author's known devices the message was sent.
`sent_at`*	The timestamp representing when the message was sent.
`received_at_ms`*	The timestamp representing when the message was received on the current device.
`expiresTimer`	The lifespan of the message in seconds, indicating how long it will remain before expiring. If this field is `0` or missing, the message will not expire.
`expirationStartTimestamp`	The timestamp marking the start of the expiration countdown for this message.
`isErased`	A boolean flag denoting whether the message was deleted for everyone — *i. e.*, is a "this message was deleted" notice.
`body`, `bodyRanges`, `preview` and `attachments`	The contents of the message.

- `incoming` – An incoming message, sent by another user and received by the current user.
- `outgoing` – An outgoing message, sent by the current user.
- `group-v2-change` – A change in a group's settings or metadata. This message type will be explained in a latter subsection.
- `timer-notification` – A change in a conversation's expire timer[5] state or duration. Messages with this type, will have a `expirationTimerUpdate` field that contains an object with two fields, the new duration for the expire timer — `expireTimer` — and the Service Id of the user who made that change — `sourceServiceId`. If the expire timer was disabled, the duration will be set to 0.
- `call-history` – The in-chat message that appears when a call is attempted. By cross-referencing the `callId` field in the message's JSON object with the `callId` column in the `calls_history` table, details about the call can be obtained.

Unless otherwise stated, the following subsections apply only to messages of type `incoming` and `outgoing`.

6.4.2 Body, Mentions and Text Formatting

As previously mentioned, to account for user mentions and text formatting, we must parse the `body` and `bodyRanges` fields. The `body` field contains the simple text content of the message. The `bodyRanges` field is a list that stores mention details and text

[5] The timer that determines when a message gets automatically deleted.

formatting details as objects. Listing 5 exemplifies a `bodyRanges` list with both objects. The asterisk represents the object replacement character that would be present at the mention's location.

Listing 5: *An example of the contents found in the `bodyRanges` field.*

```
"bodyRanges": [
  { "length": 10, "start": 13, "style": 1 },
  { "length": 1, "start": 30, "style": 1 },
  { "length": 8, "start": 23, "style": 2 },
  {
    "length": 1,
    "mentionAci": "ddd8e8a2-b058-4830-a38a-98cb13e97b91",
    "replacementText": "Tiago Santos",
    "start": 39
  },
  { "length": 6, "start": 34, "style": 1 },
  { "length": 6, "start": 34, "style": 2 }
],
"body": "Test message with bold italics or both *"
```

Mention objects consist of four fields:

1. `mentionAci` – The ACI of the mentioned user.
2. `replacementText` – The text that identifies the mentioned user (usually the contact's conversation name). When the mentioned user is the current user, this field is empty.
3. `start` – The index position of the object replacement character signaling a mention in the message's text body.
4. `length` – The length that the special character occupies.

To correctly introduce a mention in a message's body, one would have to replace the special character with the contents of the `replacementText` field. Text formatting objects consist of three fields:

1. `start` – The index position of where the format starts.
2. `length` – The length of the format.
3. `style` – An integer that identifies the style that will be applied to the matching text. `1` is bold, `2` is italic, `3` is a spoiler box, `4` is strike-through, and `5` is monospace.

The same matching text can have multiple styles applied to it. Figure 6.2 illustrates the example text message from Listing 5.

Additionally, a message object contains a `mentionsMe` boolean field, which indicates whether the message mentions the current user.

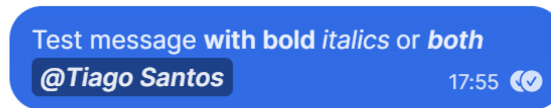


Figure 6.2: *An example message with text formatting and mentions displayed on the Signal Desktop client.*

6.4.3 Preview

The `preview` field in Signal messages is used to store data for the preview that is automatically generated when a URL is included in the message body. This field is composed of a `url` field, which specifies the URL that generated this preview, a `title` field that often represents the name or heading of the linked webpage, and a `description`, which provides a brief summary of the content. Additionally, the `preview` field includes an `image` object, which stores the cryptographic metadata required to decrypt the preview's image. This image can be decrypted using the same process as regular attachments, and it is intended to visually represent the URL, such as a website logo, a thumbnail, or any related content image.

6.4.4 Attachments

As mentioned in Section 5.2.2, the `attachments` field contains a list that provides details about each attachment included in the message. Each object within this list includes metadata about an attachment and the cryptographic parameters necessary for its decryption.

6.4.5 Edit History

The body of a message can be edited multiple times. The `editMessageReceivedAtMs` field registers the timestamp of when the latest version of the message was received, while the `editHistory` field maintains a list of all previously received versions. Each entry in the `editHistory` list includes the message content stored in the `body` and `bodyRanges` fields, alongside the timestamp of when that particular edit was received, stored in the `received_at_ms` field. Although the `attachments` field appears in these entries, it remains empty, as editing message attachments is not currently supported. Changes to the `preview` field, which reflects updated links, may occur across different versions of a message. If a specific message has never been edited, both the `editMessageReceivedAtMs` field and the `editHistory` list will be missing from the main JSON object. An excerpt of a message with an edit history can be observed in Listing 6.

6.4.6 Reactions

In Signal, users can react to messages using emojis; these are limited to one emoji per user and per message, however, the chosen emoji can be changed or removed at any time. The `reactions` field is a list containing objects that detail the reactions associated

Listing 6: *An excerpt of the fields found in a message with an edit history.*

```
{
  "received_at_ms": 1733871561461,
  "editHistory": [{
      "attachments": [],
      "body": "Message Version 2",
      "bodyRanges": [],
      "preview": [],
      "received_at_ms": 1733871689441,
    },{
      "attachments": [],
      "body": "Message Version 1",
      "bodyRanges": [],
      "preview": [],
      "received_at_ms": 1733871561461,
    }],
  "editMessageReceivedAtMs": 1733871689441
}
```

with a message. An example can be observed in Listing 7. Reaction objects have four fields:

1. `emoji` – A string containing the reaction's emoji.
2. `fromId` – The Service ID of the reactor[6].
3. `targetTimestamp` – The timestamp of the message to which the reaction applies. If the message has an edit history, this timestamp can be compared with the timestamps of all message versions to determine which specific version received the reaction.
4. `timestamp` – The timestamp of when the reaction was performed.

Listing 7: *An example of the contents found in the `reactions` field.*

```
{
  "reactions": [
    {
      "emoji": "[EMOJI]",
      "fromId": "dd52badb-b058-4830-a38a-98cb1352badb",
      "targetTimestamp": 1733871688069,
      "timestamp": 1733871766605
    }
  ]
}
```

[6] The user who performed said reaction.

6.4.7 Read Receipts and Send Status

It is important to clarify the difference between a "read" and a "viewed" message in the context of Signal. Both of these states only make sense when relating a user to a specific message; in group conversations, these states exist for each individual member.

A "read message" is any message — *e.g.*, a regular text message — that has been seen by the recipient. Distinctively, a "viewed message" refers to one of two situations — a voice message that has been seen in chat and played, or a view-once message that has been seen in chat and opened by the recipient. For example, if user A sends two messages to user B — a text message and a voice message — and user B only opens the conversation, both messages would be considered "read". However, as soon as user B plays the voice message, that specific message transitions to the "viewed" state. For simplicity, these two scenarios will collectively be referred to as "viewable messages".

Behind the scenes, Signal handles read receipts differently for `outgoing` messages and those received by the current user. Signal also tracks the read receipts of messages triggered by user actions, such as a call history message after initiating a call or a group change; however, this tracking applies exclusively to received messages.

Read Receipts for Received Messages.

For messages received by the user, the `readStatus` and `seenStatus` fields store integers that indicate whether the message has been seen or viewed by the current user. A `readStatus` and `seenStatus` value of `1` signifies that the message was neither read nor viewed by the current user. When the `readStatus` is `0` and the `seenStatus` is `2`, it indicates that the message has been read by the current user. Lastly, a `readStatus` and `seenStatus` value of `2` denotes that the message has been both read and viewed by the current user, a state that can only be observed in viewable messages. Table 6.7 summarizes these fields' values and their significance. Listing 8 displays an excerpt of the JSON object of an example text message that was received and read by the current user.

Table 6.7: *List of possible message read receipt states for messages received by the current user.*

State	readStatus	seenStatus
Unread	1	1
Read	0	2
Viewed	2	2

Send Status for Outgoing Messages.

For `outgoing` messages, the `sendStateByConversationId` field is key. The `sendStateByConversationId` field maps each user's conversation ID — including the current user, who sent the message, and all receiving users — to an object containing the message's send status for that specific user. This inner object contains two key elements: the

Listing 8: *An excerpt from a JSON object of an `incoming` message with read receipts information.*

```
{
  "type": "incoming",
  "readStatus": 0,
  "seenStatus": 2,
  "body": "A message received by the current user"
}
```

`status`, which denotes the current state of the message for the user, and the `updatedAt`, a timestamp indicating when the message send status was last updated for that user. The possible values for the `status` field are `Failed`, `Pending`, `Sent`, `Delivered`, `Read`, `Viewed`, and `Skipped`. As the names suggest, `Read` means that the respective user has read the message, while `Viewed` indicates that the user has played the viewable message. The combination of both of these elements, allows us to discern, for example, when a message was read or viewed by a certain user.

Listing 9 shows an excerpt of the JSON object for a voice message sent by the current user to a group conversation. In this example, from bottom to top, the first user sent the message; the second received it but has neither seen nor played it; the third has both seen and played it; while the fourth has seen the message but has not played it yet.

Listing 9: *An excerpt from a JSON object of an `outgoing` voice message's send state.*

```
{
  "type": "outgoing",
  "sendStateByConversationId": {
    "7fc993c9-439e-414d-b8d1-9d131493861c": {
      "status": "Read",
      "updatedAt": 1735790339678
    },
    "ec81695a-60d8-4910-86d7-d076cfd13983": {
      "status": "Viewed",
      "updatedAt": 1735790229545
    },
    "b7c18f3c-5739-45b9-83c7-d685cc611919": {
      "status": "Delivered",
      "updatedAt": 1735790227149
    },
    "6b0b2008-89b5-49d5-ab14-eb1ac562cb06": {
      "status": "Sent",
      "updatedAt": 1735790225007
    }
  }
}
```

6.4.8 Quote

In Signal, messages can be sent as direct replies to a previous message in the conversation — this is known as "quoting a message". When a message is a reply, information about the quoted message can be found in the `quote` field of the message's JSON object. The `quote` field — exemplified in Listing 10 — is an object containing several key fields:

- `messageId` – The UUID of the quoted message.
- `authorAci` – The ACI of the user who sent the quoted message.
- `referencedMessageNotFound` – A boolean that is `true` if the quoted message could not be found.

Additionally, the `quote` object may contain `bodyRanges`, `attachments`, and `text` fields, which can help reconstruct parts of the quoted message when the original message is missing from the database.

Listing 10: *An excerpt of a `quote` field in a message's JSON object.*

```
1  {
2    "quote": {
3      "authorAci": "7fc993c9-439e-414d-b8d1-9d131493861c",
4      "attachments": [],
5      "bodyRanges": [],
6      "isViewOnce": false,
7      "messageId": "01942478-3a94-749e-b70a-eb1ac820ca11",
8      "referencedMessageNotFound": false,
9      "text": "This message was quoted."
10   },
11 }
```

6.4.9 Stories

As explained in Section 3.2, stories in Signal are stored as regular messages, identifiable by their message type — `story`. Stories uploaded by the current user will share the same conversation ID as the "Notes to Self" chat, which is tied to the current user's Service ID. Additionally, stories introduces a new attachment type, identified by its `contentType` being `text/x-signal-story`; this attachment type is used for stories that consist of text displayed over a colored background. An example of a text attachment is shown in Listing 11.

6.4.10 Group Changes

Messages of type `group-v2-change` represent the notification messages sent in the group's conversation when a group-related event occurs; internally, this is known as a "group change". These events include changes to the group's profile — *i. e.*, name,

Listing 11: *An example of a Signal story consisting of a text attachment.*

```
{
  "contentType": "text/x-signal-story",
  "textAttachment": {
    "color": 4285041620,
    "text": "Test",
    "textStyle": 1,
    "textForegroundColor": 4294967295
  },
  "size": 4
}
```

description and avatar — changes to the group's configuration — *e. g.*, toggling join links — changes in the group's members, or the creation of the group itself.

These messages include a `groupV2Change` field, which is an object with two keys: `from`, identifying the Service ID of the user who performed the change or triggered the event, and `details`, a field containing a list. This list stores detail objects, each representing the specifics of a group change. A single message can contain multiple details objects, each representing a separate group event. Each detail object is composed of a `type` field, which identifies the type of change/event, along with auxiliary fields that provide additional information about the event. The structure and content of these auxiliary fields vary depending on the type of change described. Table C.1 in Appendix C presents all the currently possible group change types with a brief description of each, the corresponding details they include, and an example JSON illustrating their structure.

6.5 Redundant Tables

Even though the `json` column in the `messages` table contains all the information about a specific message, executing SQL searches on this field is neither efficient nor straightforward. For that reason, Signal's local database includes several tables that store redundant information to facilitate these operations. Among them, we found it relevant to describe the `mentions` and `reactions` tables.

Mentions Table. The `mentions` table stores information about mentions across all messages. The `messageId` column identifies the message containing the mention, while the `mentionAci`, `start`, and `length` columns provide the necessary metadata to render the mention as readable text, as described in Section 6.4.2.

Reactions Table. The `reactions` table stores information about reactions across all messages. The information contained within this table's columns should be processed as described in Section 6.4.6.

Chapter Summary

This chapter presented an in-depth analysis of the main forensic artifacts of Signal Desktop, focusing on the SQLite database where most relevant data is stored. Key tables were examined, highlighting user account information, conversation metadata, call logs, message content (including text, attachments, stories, and history), and reactions. Special attention was given to the relationships between tables, timestamp handling, and the cryptographic handling of attachments and avatars. Overall, the database provides a comprehensive view of user activity and interactions, making it a valuable source of forensic evidence.

7 The SignalForensics Script

The process described in the previous chapter is not overly complex; however, it is not entirely straightforward either. More importantly, it is time-consuming, especially when decrypting multiple attachments, as each attachment is encrypted with unique cryptographic data. With this in mind, we developed a script that automates this process and provides various modes of execution, allowing us to accommodate different operating systems and environments. We named the Python script `SignalForensics`. Beyond decrypting and exporting Signal artifacts, SignalForensics also generates CSV and HTML reports based on the gathered information, offering a structured and comprehensive overview of the decrypted data. This ensures flexibility for integration into various processing programs and adaptability for diverse analytical purposes. The reports also include SHA-256 hashes of all processed files, enabling forensic investigators to verify artifact integrity and ensure that the data remains unaltered during processing, thereby supporting chain-of-custody requirements. The upcoming subsections detail the different execution modes and available options. Listing 12 displays the syntax for the script. Listing 13 exemplifies the terminal output of the script. Figure 7.1 illustrates an example of an HTML report generated by the script. Please note that the screenshot is cropped and does not display the full dashboard, some fields are omitted from view to save space. SignalForensics is open-source and the source code can be obtained at `https://github.com/labcif/SignalForensics`.

7.1 Execution Modes

SignalForensics supports five execution modes to accommodate different forensic contexts and platforms. They are: Live, Forensic, Auxiliary Key Provided, SQLCipher Key Provided and Passphrase Provided. Each mode may require additional inputs and its functionality often depends on the environment from which the Signal data was acquired. Table 7.1 presents a brief description of each mode. Table 7.2 summarizes the compatibility of each mode across the supported environments.

Listing 12: *SignalForensics' syntax*

```
SignalForensics [-m live] [-e <environment>] -d <signal_dir> [-o <output_dir>] [OPTIONS]
SignalForensics -m forensic -e <environment> -d <signal_dir> [-o <output_dir>] [-p
↪  <password> | -pb <HEX> | -pbf <file>] -gkf <gnome_keyring_file> [OPTIONS]
SignalForensics -m aux [-e <environment>] -d <signal_dir> [-o <output_dir>] [-kf <file>
↪  | -k <HEX>] [OPTIONS]
SignalForensics -m key [-e <environment>] -d <signal_dir> -o <output_dir> [-kf <file> |
↪  -k <HEX>] [OPTIONS]
SignalForensics -m passphrase -e <environment> -d <signal_dir> [-p <passphrase> | -pb
↪  <HEX> | -pbf <file>] [-o <output_dir>] [OPTIONS]
```

Listing 13: *Example terminal output for SignalForensics.*

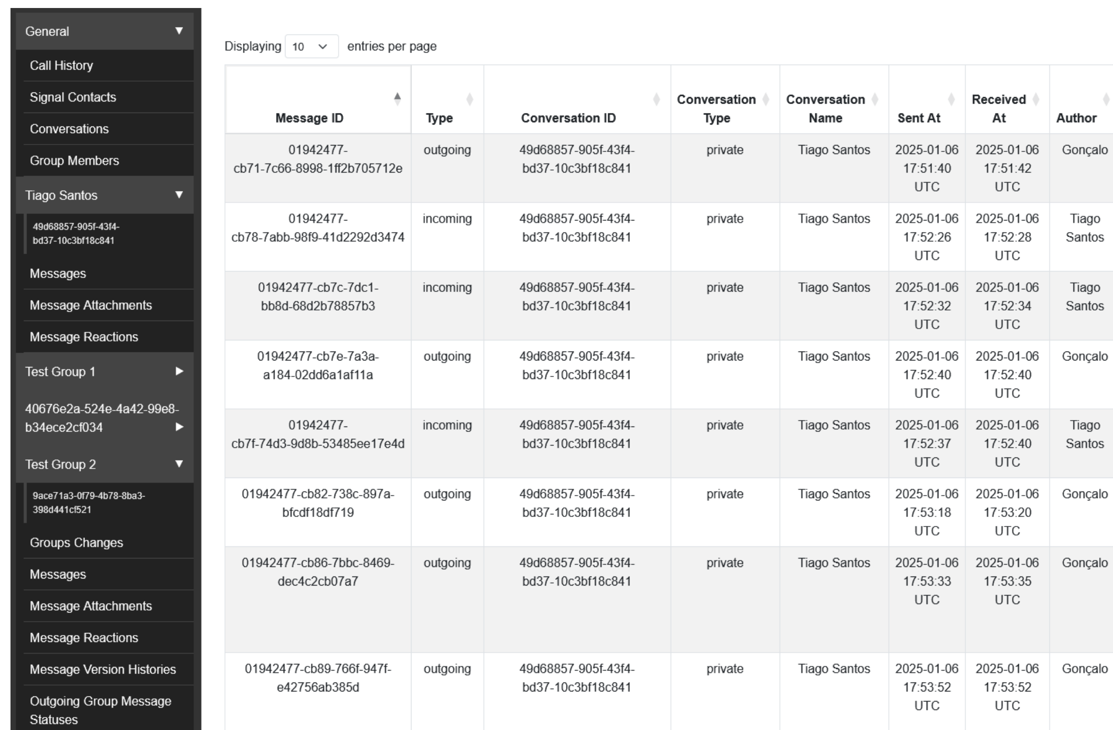
```
$ SignalForensics -d C:\Users\TheUser\AppData\Roaming\Signal -o output_dir
------------==<[ CONFIG ]>==-------------
Mode: live
Environment: windows
Signal Directory: C:\Users\TheUser\AppData\Roaming\Signal
Output Directory: output_dir
----------------------------------------
[i] SQLCipher Key: 9a325c733c1d19b840e765b5bafdee957797eea25b7aacae746f83f0c6e44479
[i] Opening SQLCipher database
[i] Exported the unencryted database
[i] Exporting attachments...
[i] Exported 192 attachments
[i] Writing reports...
```

Table 7.1: *Execution modes in SignalForensics.*

Execution Mode	Requirements
Live	Requires the path to the data directory. Must be run on the same environment as Signal.
Forensic	Requires the path to the data directory, path to the KWallet or Gnome Keyring file, and the wallet/keyring's password.
Auxiliary Key Provided	Requires the path to the data directory and the decrypted auxiliary key.
Decryption Key Provided	Requires the path to the data directory and the decrypted SQLCipher key.

Table 7.2: *Compatibility of execution modes across supported environments.*

Execution Mode	Windows	Gnome	KWallet	Linux w/o Keystore
Live	Yes	Yes	Yes	N/A
Forensic	No	Yes	Yes	Yes
Auxiliary Key Provided	Yes	Yes	Yes	Yes
SQLCipher Key Provided	Yes	Yes	Yes	Yes
Passphrase Provided	No	Yes	Yes	No



Message ID	Type	Conversation ID	Conversation Type	Conversation Name	Sent At	Received At	Author
01942477-cb71-7c66-8998-1ff2b705712e	outgoing	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:51:40 UTC	2025-01-06 17:51:42 UTC	Gonçalo
01942477-cb78-7abb-98f9-41d2292d3474	incoming	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:52:26 UTC	2025-01-06 17:52:28 UTC	Tiago Santos
01942477-cb7c-7dc1-bb8d-68d2b78857b3	incoming	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:52:32 UTC	2025-01-06 17:52:34 UTC	Tiago Santos
01942477-cb7e-7a3a-a184-02dd6a1af11a	outgoing	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:52:40 UTC	2025-01-06 17:52:40 UTC	Gonçalo
01942477-cb7f-74d3-9d8b-53485ee17e4d	incoming	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:52:37 UTC	2025-01-06 17:52:40 UTC	Tiago Santos
01942477-cb82-738c-897a-bfcdf18df719	outgoing	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:53:18 UTC	2025-01-06 17:53:20 UTC	Gonçalo
01942477-cb86-7bbc-8469-dec4c2cb07a7	outgoing	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:53:33 UTC	2025-01-06 17:53:35 UTC	Gonçalo
01942477-cb89-766f-947f-e42756ab385d	outgoing	49d68857-905f-43f4-bd37-10c3bf18c841	private	Tiago Santos	2025-01-06 17:53:52 UTC	2025-01-06 17:53:52 UTC	Gonçalo

Figure 7.1: *An example HTML report generated by SignalForensics (cropped to save space).*

> **Note**
>
> In Linux environments without an OS-level keystore library, `Live` mode behaves identically to `Forensic` mode, as no secure key retrieval via the operating system is required.

7.1.1 Live (`-m live`)

This mode takes Signal's data directory as input and fully automates the decryption process. The script decrypts the auxiliary key using the available operating system–specific services. For Windows environments, it uses the Windows Data Protection API (DPAPI) to decrypt the auxiliary key. For Linux environments, it accesses the keystore library using the currently logged-in user session. This mode requires execution within the original user account and environment within which the Signal artifacts were generated.

7.1.2 Forensic (`-m forensic`)

This mode takes Signal's data directory together with the relevant credential storage artifacts (*e. g.*, a Gnome Keyring or a KWallet file) and their associated secrets as inputs. It is designed to be run in a *post-mortem* context, where the artifacts were acquired from a Linux environment.

In this mode, the script uses the provided credential storage to extract the passphrase required to decrypt the auxiliary key, which is then used to obtain the SQLCipher key. The process by which this passphrase is extracted is described in detail in Section 5.3 and 5.4.

When using Gnome Keyring, the corresponding keyring file and its password must be supplied. When using KWallet, the required inputs depend on the wallet's protection method:

- For wallets encrypted with **Blowfish**, both the wallet file and its associated salt file (stored in the same directory with a `.salt` extension) are needed, along with the wallet's password.
- For wallets protected by a **GPG key**, the path to a file storing the GPG private key in ASCII-armored format must be provided, and instead of the wallet's password, the password of the GPG key itself is required.

For environments where neither Gnome Keyring nor KWallet is available — or being used by Signal Desktop — as described in Section 5.2.1, a hardcoded passphrase is utilized to derive the auxiliary key.

This mode is fully isolated from the original environment, allowing it to be executed in a forensic lab or any external setting. Forensic Mode is only available for artifacts generated in Linux environments.

Passphrase Provided (`-m passphrase`)

This mode takes Signal's data directory and the passphrase as inputs. It uses the passphrase to decrypt the auxiliary key, which is then used to obtain the SQLCipher key. This mode works identically to the Forensic Mode, but it assumes that the user has already extracted the passphrase from Gnome's Keyring, as such it suffers from the same environment limitations as Forensic Mode.

7.1.3 Auxiliary Key Provided (`-m aux`)

This mode takes Signal's data directory and the auxiliary key as inputs; it then uses the auxiliary key, as described in Section 5.2.1, to obtain the SQLCipher key. Since these environment-dependent steps are skipped, this mode works for artifacts generated on all operating systems supported by Signal Desktop and can be executed in an external environment.

7.1.4 Decryption Key Provided (`-m key`)

This mode takes Signal's data directory and the SQLCipher key as inputs. Since all necessary inputs are provided by the user, this mode, like the Auxiliary Key Provided mode, works on external environments and for all Desktop platforms (Windows, macOS, and Linux).

7.2 Additional Options

SignalForensics offers various options to customize its behavior, including flags to skip optional steps, controlling verbosity levels, and fine-tuning report generation.

7.2.1 Flags

No Decryption. The `--no-decryption` or `-nd` argument runs the key-derivation phase only, calculating the SQLCipher key without actually decrypting the local database or attachments. Because no decryption or report generation is performed, this mode executes significantly faster and can be useful when the goal is simply to obtain or verify the key. This flag is incompatible with the *Decryption Key Provided* mode.

Skip Database. The `--skip-database` or `-sD` argument disables the export of the SQLCipher database to an unencrypted SQLite database file. This option is intended for cases where an investigator is only interested in the processed reports and does not need a standalone decrypted database.

Skip Attachments. The `--skip-attachments` or `-sA` argument skips the decryption of attachments and avatars.

Skip Reports. The `--skip-reports` or `-sR` argument skips the generation of CSV and HTML reports.

Merge Conversations. The `--merge-conversations` or `-mc` argument merges message-related reports into single CSV files instead of separating them by conversation.

Convert Timestamps. The `--convert-timestamps` or `-t` argument converts all timestamps to a human-readable format in the generated reports. A specific timezone can be provided with this argument; if not specified, it defaults to the UTC timezone.

7.2.2 Output Management

SignalForensics includes a quiet mode (`--quiet` or `-q`) to suppress all log output and offers three verbosity levels (`-v` up to `-vvv`) that output progressively more detailed information during execution.

7.3 Input/Output Arguments

SignalForensics implements the arguments described in this subsection to handle the inputs and outputs of the script effectively. These include the path to Signal's data directory, the output directory, and optional keys for decryption.

7.3.1 Signal Data Directory

The `--dir` or `-d` argument specifies the path to Signal's data directory. On Windows, this would be the `Signal` directory in the user's Roaming folder.

7.3.2 Output Directory

The `--output` or `-o` argument specifies the directory where the decrypted files, extracted artifacts and generated reports will be saved. This argument is optional; if not provided, SignalForensics assumes that "No Decryption" flag is enabled, meaning that no artifacts will be decrypted or exported and no reports will be generated. For optimal results, it is recommended to provide an empty directory. If the output directory already contains a SQLite database, the script will not export a new one.

7.3.3 Key Input Arguments

The script offers two ways to provide keys for the Key Provided modes (Decryption Key Provided and Auxiliary Key Provided):

- **Key File (`--key-file` or `-kf`)**: Accepts the path to a plain text file containing the key. The key must be saved as a hexadecimal string representation of the bytes within the file (note: the script reads a string, not the file's binary contents).
- **Direct Key Input (`--key` or `-k`)**: Accepts the key directly as a hexadecimal string.

7.3.4 Passphrase and Password Input Arguments

The script provides three ways to input the passphrase or password for the Passphrase Provided and Forensic Modes:

- **Password Bytes File (`--password-bytes-file` or `-pbf`)**: Accepts the path to a plain text file containing the passphrase/password. The passphrase must be saved as a hexadecimal string representation of the bytes within the file (note: the script reads a string, not the file's binary contents).
- **Password Bytes (`--password-bytes` or `-pb`)**: Accepts the passphrase/password as a hexadecimal string.
- **Password (`--password` or `-p`)**: Accepts the passphrase/password directly as a string.

7.3.5 Gnome Keyring File

The `--gnome-keyring-file` or `-gkf` argument specifies the path to the Gnome Keyring file. When handling artifacts originating from a Linux environment using Gnome Keyring, this file is required by the Forensic Mode to extract the passphrase needed to decrypt the auxiliary key.

7.3.6 KWallet File

The `--kwallet-file` or `-kwf` argument specifies the path to the KWallet file. When handling artifacts originating from a Linux environment that uses KWallet, this file is required by Forensic Mode. Depending on the wallet's protection method, additional files or credentials may also be necessary.

7.3.7 KWallet Salt File

The `--kwallet-salt-file` or `-ksf` argument specifies the path to the salt file associated with a KWallet encrypted using Blowfish. In Forensic Mode, the salt is required to derive the key used to decrypt the wallet.

7.3.8 GPG Key File

The `--gpg-key-file` or `-gak` argument specifies the path to a file storing a GPG private key in ASCII-armored format. This key is required by Forensic Mode when the KWallet is protected using GPG. In such cases, the password provided must correspond to the GPG key rather than the wallet itself.

7.4 Usage Examples

The following command-line examples illustrate how SignalForensics can be configured in different scenarios depending on the execution mode, platform and encryption mechanisms used.

7.4.1 Live Mode

In this example, Live Mode is used to extract artifacts from a Windows environment. Since the environment is Windows by default, no additional parameter is required to specify it:

```
1 SignalForensics -m live -d "C:\Users\TheUser\AppData\Roaming\Signal" -o output_folder
```

7.4.2 Forensic Mode (Gnome Keyring)

This example demonstrates the use of Forensic Mode for a Linux environment utilizing the Gnome Keyring for credential storage:

```
1 SignalForensics -m forensic -e gnome -d ~/.config/Signal
2 -p 123456
3 -gkf ~/.local/share/keyrings/login.keyring
4 -o output_dir
```

7.4.3 Forensic Mode (KWallet, Blowfish)

In this case, Forensic Mode is executed for a Linux environment where KWallet is used with Blowfish encryption. The wallet's salt file is also required for decryption:

```
SignalForensics -m forensic -e kwallet_bf -d ~/.config/Signal
-p 123456
-kwf ~/.local/share/kwalletd/wallet.kwl
-ksf ~/.local/share/kwalletd/wallet.salt
-o output_dir
```

7.4.4 Forensic Mode (KWallet, GPG)

This example illustrates the use of Forensic Mode with a GPG-protected KWallet. The GPG private key is provided for decryption:

```
SignalForensics -m forensic -e kwallet_gpg -d ~/.config/Signal
-p 123456
-kwf ~/.local/share/kwalletd/wallet.kwl
-gak gpg_private_key.asc
-o output_dir
```

7.4.5 Auxiliary Key Mode

In this example, the auxiliary key is provided directly:

```
SignalForensics -m aux -d signal_data/ -kf aux_key.txt -o output_folder
```

7.4.6 SQLCipher Key Mode

In this example, the SQLCipher key is provided directly:

```
SignalForensics -m key -d signal_data/ -k 9a325c73... -o output_folder
```

7.4.7 Passphrase Mode

In this example, a passphrase is provided to derive the auxiliary key:

```
SignalForensics -m passphrase -d signal_data/ -p "vEq+XKoPekTsiU+nciF4" -o output_folder
```

7.5 Relevant Updates

SignalForensics was actively maintained throughout the entire duration of this project, with new features and improvements added over time. Many of these updates have already been described in previous sections; however, certain functionalities had to be revised specifically to ensure compatibility with newer versions of Signal Desktop. One notable example is the `messages` table, which underwent a structural change. As briefly mentioned in Section 6.4, initially, all relevant data was stored within the `json` field, while the remaining columns merely duplicated selected values to facilitate filtering and SQL operations — a structure similar to that of the `conversations` table. Over the course of this work, Signal Desktop began to transition away from this approach. In recent versions, some fields are no longer included in the JSON object and are instead stored exclusively in their respective columns. To accommodate this change, SignalForensics was updated accordingly. Backward compatibility was preserved, allowing the script to continue supporting earlier versions of Signal Desktop where this was not the case.

Chapter Summary

This chapter introduced the SignalForensics Python script, developed to automate the decryption and extraction of Signal Desktop artifacts. The script supports multiple execution modes, allowing flexible operation across different operating systems and environments. SignalForensics generates structured outputs, including CSV and HTML reports, and calculates SHA-256 hashes for integrity verification. Additional options and flags enable customization of the decryption process, report generation, and artifact handling. Further updates ensured compatibility with evolving Signal Desktop versions while maintaining backward support, making the script a versatile tool for forensic investigations.

8
Conclusion

This investigation had two primary objectives: decrypting Signal Desktop's locally stored data and analyzing its forensic artifacts in Windows environments. Our results confirmed that these artifacts contain valuable information, shedding light on the application's storage structure. To ensure reproducibility, we developed a detailed methodology and an automated script – SignalForensics – to streamline data extraction and forensic analysis operations on Signal Desktop.

While full decryption in Windows requires access to the original execution environment — due to the use of DPAPI. Without this access, the auxiliary key must be manually extracted and provided as an input to the script. This requirement may restrict the applicability of the methodology in Windows environments where access to the user's original environment — or to a virtual machine emulating it — is not possible. As explained in earlier chapters, this limitation does not apply in Linux environments. There, it is possible to recover the necessary decryption keys without activating the original system, using only the contents of the acquired file system and the wallet or keyring's master key.

Despite these limitations, we consider this work a significant contribution to the field of digital forensic analysis. To our knowledge, no previous work has mapped the forensic artifacts present in Signal Desktop in such detail, nor has the Electron safeStorage mechanism been explored in the context of this application. Our findings provide a comprehensive guideline for forensic investigators, aiding in the interpretation of artifacts and facilitating the construction and validation of hypotheses during an investigation. Moreover, both the methodology and the developed script remain useful even in cases where encryption is not a barrier. The tool enables the examination of Signal's local data without opening the application, preventing potential alterations or contamination of files — an essential factor in preserving evidence integrity, particularly for expired messages that may still reside in the database until Signal is launched. Additionally, exporting results in CSV format ensures compatibility with various tools, allowing forensic investigators to manage and analyze the data flexibly. Finally, this work also introduces a method for forensically extracting and decrypting data from

the Gnome Keyring and KWallet, allowing for full decryption of Signal Desktop data in Linux environments — even in offline scenarios where the original operating system is no longer available.

As future work, one goal is to develop the Forensic Mode for Windows in SignalForensics, allowing it to bypass DPAPI using only the user's password, security identifier, and local DPAPI files, without requiring the original Windows system, enabling decryption in external environments. Furthermore, future research could explore the different security measures applied to the auxiliary key in macOS's Keychain, to assess the feasibility of this methodology in macOS environments.

Another promising avenue for future work is the integration of SignalForensics with broader digital forensic platforms. In particular, developing a dedicated module for Exhume[1], a modular forensic toolkit designed to support step-by-step analysis workflows, could enable deeper correlation of Signal Desktop artifacts with evidence from other sources. This would also make SignalForensics available within Thanatology[2] — a modern, cross-platform forensic interface built on top of Exhume — bringing its capabilities to a more user-friendly desktop environment and bridging the gap between low-level analysis tools and accessible interfaces tailored to investigative workflows. We consider Exhume and Thanatology particularly promising despite the availability of more established forensic frameworks such as Autopsy. While Autopsy offers maturity, community support, and a wide range of extensions, Exhume is being developed with different design priorities that align well with our goals. Implemented in Rust, it combines memory safety with high performance, lowering the risk of low-level vulnerabilities without sacrificing efficiency. Moreover, its modular, multi-platform design makes it well suited for integrating custom components and potentially scaling into distributed workflows, an aspect that can be especially valuable for large-scale or cloud-based investigations. For these reasons, Exhume and Thanatology are an appealing choice for future integration with SignalForensics, despite their relative novelty.

[1] https://www.forensicxlab.com/docs/exhume

[2] https://www.forensicxlab.com/docs/thanatology

This page intentionally left blank.

Bibliography

A.P. Taneva G. Smaragdakis, S. Picek (2023). *Reverse Engineering of Web Cookies | TU Delft Repository*. URL: https://repository.tudelft.nl/record/uuid:8089d4d3-8e36-4b11-a9fc-17b43ebceb9a (visited on 2024-12-11).

Abegaz, Tamirat, Boone Phaxai, and Bryson Payne (2024). "Investigating Ephemeral Messaging Apps on iPhone 14: Insight into Signal, WhatsApp, Snapchat, Instagram, and Confide". In: *Proceedings of the Future Technologies Conference (FTC) 2024, Volume 4*. Ed. by Kohei Arai. Cham: Springer Nature Switzerland, pp. 312–331. ISBN: 978-3-031-73128-0. DOI: 10.1007/978-3-031-73128-0_19.

Abrams, Lawrence (2024). *Signal downplays encryption key flaw, fixes it after X drama*. URL: https://www.bleepingcomputer.com/news/security/signal-downplays-encryption-key-flaw-fixes-it-after-x-drama/ (visited on 2024-12-11).

Andriotis, Panagiotis, Atsuhiro Takasu, and Theo Tryfonas (Jan. 2014). "Smartphone Message Sentiment Analysis". In: *IFIP Advances in Information and Communication Technology*. Vol. 433. ISBN: 978-3-319-12567-1. DOI: 10.1007/978-3-662-44952-3_17.

Auxier, Brooke et al. (Nov. 2019). *Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information*. en-US. URL: https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/ (visited on 2025-05-20).

Bilz, Alexander (2021). *A Forensic Gold Mine II: Forensic Analysis of Signal Messenger on Windows 10*. URL: https://www.alexbilz.com/post/2021-06-07-forensic-artifacts-signal-desktop/ (visited on 2024-12-11).

Bowling, Herschel et al. (Feb. 2023). "We are meeting on Microsoft Teams: Forensic analysis in Windows, Android, and iOS operating systems". In: *Journal of Forensic Sciences* 68.2, pp. 434–460. ISSN: 1556-4029. DOI: 10.1111/1556-4029.15208. URL: http://dx.doi.org/10.1111/1556-4029.15208.

Casey, Eoghan (May 2014a). *Digital evidence and computer crime*. 3rd ed. London, England: Elsevier Science.

Casey, Eoghan (2014b). *Handbook of Digital Forensics and Investigation*. 1st. Amsterdam: Academic Press. URL: https://www.elsevier.com/books/handbook-of-digital-forensics-and-investigation/casey/978-0-12-374267-4.

Cohn-Gordon, Katriel et al. (Sept. 2020). "A Formal Security Analysis of the Signal Messaging Protocol". In: *Journal of Cryptology* 33.4, pp. 1914–1983. ISSN: 1432-1378. DOI: 10.1007/s00145-020-09360-1. URL: http://dx.doi.org/10.1007/s00145-020-09360-1.

Gonçalo Paulino, Miguel Negrão, Miguel Frade and Patrício Domingues (2025). "Decrypting messages: Extracting digital evidence from signal desktop for windows". In: *Forensic Science International: Digital Investigation* 54, p. 301941. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2025.301941. URL: https://www.sciencedirect.com/science/article/pii/S2666281725000800.

Gupta, Khushi, Phani Lanka, and Cihan Varol (May 2024). "A holistic digital forensic analysis of Discord – Storage, memory, and network perspectives". In: *Journal of Forensic Sciences* 69.4, pp. 1320–1333. ISSN: 1556-4029. DOI: 10.1111/1556-4029.15548. URL: http://dx.doi.org/10.1111/1556-4029.15548.

Heath, Howard, Áine MacDermott, and Alex Akinbi (2023). "Forensic analysis of ephemeral messaging applications: Disappearing messages or evidential data?" In: *Forensic Science International: Digital Investigation* 46, p. 301585. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2023.301585. URL: https://www.sciencedirect.com/science/article/pii/S266628172300094X.

Karo Karo, Grace Friscilla Margaretha, Muhammad Hilal, and Dimas Wahyu Utomo (Jan. 2024). "Analisis Bukti Digital Aplikasi Pesan Terenkripsi Signal Desktop pada Windows 11 Menggunakan Metode IDFIF v2". In: *JRIIN :Jurnal Riset Informatika dan Inovasi* 1.7. (Written in Indonesian, automated translation was used). URL: http://jurnalmahasiswa.com/index.php/jriin/article/view/828.

Kent, K et al. (2006). *Guide to integrating forensic techniques into incident response*. National Institute of Standards and Technology. DOI: 10.6028/nist.sp.800-86. URL: http://dx.doi.org/10.6028/nist.sp.800-86.

Kim, Giyoon et al. (2025). "Analyzing the Web and UWP versions of WhatsApp for digital forensics". In: *Forensic Science International: Digital Investigation* 52, p. 301861. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2024.301861. URL: https://www.sciencedirect.com/science/article/pii/S2666281724001884.

Kohn, M.D., M.M. Eloff, and J.H.P. Eloff (2013). "Integrated digital forensic process model". In: *Computers & Security* 38. Cybercrime in the Digital Economy, pp. 103–115. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2013.05.001. URL: https://www.sciencedirect.com/science/article/pii/S0167404813000849.

Kret, Ehren and Rolfe Schmidt (2024). *Signal >> Specifications >> The PQXDH Key Agreement Protocol*. URL: https://signal.org/docs/specifications/pqxdh/ (visited on 2024-12-11).

Langley, Adam, Mike Hamburg, and Sean Turner (Jan. 2016). *Elliptic Curves for Security*. RFC 7748. Internet Engineering Task Force. URL: http://www.ietf.org/rfc/rfc7748.txt.

Madden, Mary (Nov. 2014). *Public Perceptions of Privacy and Security in the Post-Snowden Era*. en-US. URL: https://www.pewresearch.org/internet/2014/11/12/public-privacy-perceptions/ (visited on 2025-05-20).

Marlinspike, Moxie (2016). *WhatsApp's Signal Protocol integration is now complete*. en. URL: https://signal.org/blog/whatsapp-complete/ (visited on 2025-03-04).

NIST (2001). *FIPS 197. Advanced Encryption Standard*. Tech. rep. Gaithersburg, MD, United States: National Institute of Standards & Technology. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

OpenJS Foundation and Electron contributors (2023). *Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron*. URL: https://www.electronjs.org/ (visited on 2024-12-11).

Perrin, Trevor and Moxie Marlinspike (Nov. 2016). *The Double Ratchet Algorithm*. Tech. rep. Version Revision 1. Signal. URL: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

Signal Foundation (2025a). *Group chats – Signal Support*. URL: https://support.signal.org/hc/en-us/articles/360007319331-Group-chats (visited on 2025-02-11).

Signal Foundation (2025b). *Signal >> Documentation*. URL: https://signal.org/docs/ (visited on 2024-12-11).

Signal Foundation (2025c). *Signal-Desktop/ts/model-types.d.ts · signalapp/Signal-Desktop*. URL: https://github.com/signalapp/Signal-Desktop/blob/ca1d17354db10a14f9f5558dcb546af9f3bba578/ts/model-types.d.ts#L148 (visited on 2025-01-21).

Son, Jihun et al. (2022). "Forensic analysis of instant messengers: Decrypt Signal, Wickr, and Threema". In: *Forensic Science International: Digital Investigation* 40, p. 301347. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2022.301347. URL: https://www.sciencedirect.com/science/article/pii/S2666281722000166.

Steel, Chad (Mar. 2014). "Idiographic Digital Profiling: Behavioral Analysis Based on Digital Footprints". In: *Journal of Digital Forensics, Security and Law* 9, pp. 7–18. DOI: 10.15394/jdfsl.2014.1160.

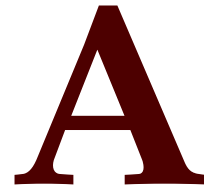
The Chromium Project (2024). *Chromium >> Home*. URL: https://www.chromium.org/chromium-projects/ (visited on 2024-12-11).

Zetetic, LLC (2024). *SQLCipher - Full Database Encryption for SQLite*. en. URL: https://www.zetetic.net/sqlcipher/ (visited on 2024-12-11).

This page intentionally left blank.

Appendices

This page intentionally left blank.



Gnome Keyring Data Extraction: Example Implementation

```
1  with keyring_path.open("rb") as f:
2      data = f.read()
3
4  # Check if the file prefix is correct
5  GNOME_KEYRING_PREFIX = b"GnomeKeyring\n\r\0\n"
6  idx = len(GNOME_KEYRING_PREFIX)
7  if data[:idx] != GNOME_KEYRING_PREFIX:
8      raise ValueError("Invalid keyring file format.")
9
10 # Get the keyring name length
11 idx += 4
12 idx = skip_string(data, idx)
13 idx += 24
14
15 # Get the keyring hash_iterations
16 hash_iterations = struct.unpack(">I", data[idx : idx + 4])[0]
17 idx += 4
18
19 # Get the keyring salt
20 salt = data[idx : idx + 8]
21 idx += 8 + 4 * 4
22
23 # Skipping the Attributes' Metadata section
24 # Get num items
25 num_items = struct.unpack(">I", data[idx : idx + 4])[0]
26 idx += 4
27 for i in range(num_items):
28     idx += 8
29
30     # Get num attributes
31     num_attributes = struct.unpack(">I", data[idx : idx + 4])[0]
```

```
32         idx += 4
33 
34         for j in range(num_attributes):
35             # Get attribute name length
36             idx = skip_string(data, idx)
37 
38             # Get attribute type
39             attr_type = struct.unpack(">I", data[idx : idx + 4])[0]
40             idx += 4
41 
42             if attr_type == 0:
43                 # Skip string hash
44                 idx = skip_string(data, idx)
45             else:
46                 # Skip guint32 hash
47                 idx += 4
48 
49     # Get number of encrypted bytes
50     num_encrypted_bytes = struct.unpack(">I", data[idx : idx + 4])[0]
51     idx += 4
52 
53     # Get encrypted data
54     encrypted_data = data[idx : idx + num_encrypted_bytes]
55 
56     # Use a custom implementation of EVP_BytesToKey to derive the key
57     keyring_key = derive_evp_key(password=password, salt=salt, key_len=16,
       ↪   iterations=hash_iterations)
58 
59     # Decrypt the keyring data
60     keyring = decrypt_keyring_data(encrypted_keyring_data, keyring_key)
61 
62     # Reset the index to the start of the decrypted payload
63     idx = 16
64     passphrase = None
65     for _ in range(num_items):
66         # Get the item display name length
67         idx = skip_string(keyring, idx) + 4
68 
69         # Extract the secret
70         secret_len = 0
71         secret = None
72         if keyring[idx - 4 : idx] != bytes.fromhex("FFFFFFFF"):
73             secret_len = struct.unpack(">I", keyring[idx - 4 : idx])[0]
74             secret = keyring[idx : idx + secret_len]
75         idx += secret_len + 16
76 
77         # Skip the reserved string and guint32[4]
78         idx = skip_string(keyring, idx)
79         idx += 4 * 4
80 
81         # Get num attributes
82         num_attributes = struct.unpack(">I", keyring[idx : idx + 4])[0]
```

```
83        idx += 4
84        for _ in range(num_attributes):
85            # Extract attribute name
86            name_len = struct.unpack(">I", keyring[idx : idx + 4])[0]
87            idx += 4
88            name = keyring[idx : idx + name_len]
89            idx += name_len
90
91            # Get attribute type
92            attr_type = struct.unpack(">I", keyring[idx : idx + 4])[0]
93            idx += 4
94            if attr_type != 0:
95                # Not a string, skip the value
96                idx += 4
97                continue
98
99            val_len = struct.unpack(">I", keyring[idx : idx + 4])[0]
100           idx += 4
101           val = keyring[idx : idx + val_len]
102           idx += val_len
103
104           if name == b"application" and val == b"Signal":
105               passphrase = secret
106               break
107
108       acl_len = struct.unpack(">I", keyring[idx : idx + 4])[0]
109       idx += 4
110       for _ in range(acl_len):
111           idx += 4  # Skip types_allowed
112           idx = skip_string(keyring, idx)  # Skip display_name
113           idx = skip_string(keyring, idx)  # Skip pathname
114           idx = skip_string(keyring, idx)  # Skip reserved_str
115           idx += 4  # Skip reserved_uint
116
117   if passphrase is None:
118       raise ValueError("Signal's auxiliary key not found in the keyring.")
```

B
Signal Desktop's SQLCipher Database Schema

Tables (49)

Name	Type	Schema
attachment_backup_jobs		`CREATE TABLE attachment_backup_jobs ( mediaName TEXT NOT NULL PRIMARY KEY, type TEXT NOT NULL, data TEXT NOT NULL, receivedAt INTEGER NOT NULL, -- job manager fields attempts INTEGER NOT NULL, active INTEGER NOT NULL, retryAfter INTEGER, lastAttemptTimestamp INTEGER ) STRICT`
mediaName	TEXT	"mediaName" TEXT NOT NULL
type	TEXT	"type" TEXT NOT NULL
data	TEXT	"data" TEXT NOT NULL
receivedAt	INTEGER	"receivedAt" INTEGER NOT NULL
attempts	INTEGER	"attempts" INTEGER NOT NULL
active	INTEGER	"active" INTEGER NOT NULL
retryAfter	INTEGER	"retryAfter" INTEGER
lastAttemptTimestamp	INTEGER	"lastAttemptTimestamp" INTEGER
attachment_downloads		`CREATE TABLE "attachment_downloads" (messageId TEXT NOT NULL REFERENCES messages(id) ON DELETE CASCADE, attachmentType TEXT NOT NULL, attachmentSignature TEXT NOT NULL, receivedAt INTEGER NOT NULL, sentAt INTEGER NOT NULL, contentType TEXT NOT NULL, size INTEGER NOT NULL, attachmentJson TEXT NOT NULL, active INTEGER NOT NULL, attempts INTEGER NOT NULL, retryAfter INTEGER, lastAttemptTimestamp INTEGER, source TEXT NOT NULL DEFAULT standard, ciphertextSize INTEGER NOT NULL DEFAULT 0, PRIMARY KEY (messageId, attachmentType, attachmentSignature)) STRICT`
messageId	TEXT	"messageId" TEXT NOT NULL
attachmentType	TEXT	"attachmentType" TEXT NOT NULL
attachmentSignature	TEXT	"attachmentSignature" TEXT NOT NULL
receivedAt	INTEGER	"receivedAt" INTEGER NOT NULL
sentAt	INTEGER	"sentAt" INTEGER NOT NULL
contentType	TEXT	"contentType" TEXT NOT NULL
size	INTEGER	"size" INTEGER NOT NULL
attachmentJson	TEXT	"attachmentJson" TEXT NOT NULL
active	INTEGER	"active" INTEGER NOT NULL
attempts	INTEGER	"attempts" INTEGER NOT NULL
retryAfter	INTEGER	"retryAfter" INTEGER
lastAttemptTimestamp	INTEGER	"lastAttemptTimestamp" INTEGER
source	TEXT	"source" TEXT NOT NULL DEFAULT standard
ciphertextSize	INTEGER	"ciphertextSize" INTEGER NOT NULL DEFAULT 0
backup_cdn_object_metadata		`CREATE TABLE backup_cdn_object_metadata ( mediaId TEXT NOT NULL PRIMARY KEY, cdnNumber INTEGER NOT NULL, sizeOnBackupCdn INTEGER ) STRICT`
mediaId	TEXT	"mediaId" TEXT NOT NULL
cdnNumber	INTEGER	"cdnNumber" INTEGER NOT NULL
sizeOnBackupCdn	INTEGER	"sizeOnBackupCdn" INTEGER
badgeImageFiles		`CREATE TABLE badgeImageFiles( badgeId TEXT REFERENCES badges(id) ON DELETE CASCADE ON UPDATE CASCADE, 'order' INTEGER NOT NULL, url TEXT NOT NULL, localPath TEXT, theme TEXT NOT NULL )`
badgeId	TEXT	"badgeId" TEXT
order	INTEGER	"order" INTEGER NOT NULL
url	TEXT	"url" TEXT NOT NULL
localPath	TEXT	"localPath" TEXT
theme	TEXT	"theme" TEXT NOT NULL
badges		`CREATE TABLE badges( id TEXT PRIMARY KEY, category TEXT NOT NULL, name TEXT NOT NULL, descriptionTemplate TEXT NOT NULL )`

Name	Type	Schema
id	TEXT	"id" TEXT
category	TEXT	"category" TEXT NOT NULL
name	TEXT	"name" TEXT NOT NULL
descriptionTemplate	TEXT	"descriptionTemplate" TEXT NOT NULL
callLinks		`CREATE TABLE callLinks (roomId TEXT NOT NULL PRIMARY KEY, rootKey BLOB NOT NULL, adminKey BLOB, name TEXT NOT NULL, -- Enum which stores CallLinkRestrictions from ringrtc restrictions INTEGER NOT NULL, revoked INTEGER NOT NULL, expiration INTEGER , deleted INTEGER NOT NULL DEFAULT 0, storageID TEXT, storageVersion INTEGER, storageUnknownFields BLOB, storageNeedsSync INTEGER NOT NULL DEFAULT 0, deletedAt INTEGER) STRICT`
roomId	TEXT	"roomId" TEXT NOT NULL
rootKey	BLOB	"rootKey" BLOB NOT NULL
adminKey	BLOB	"adminKey" BLOB
name	TEXT	"name" TEXT NOT NULL
restrictions	INTEGER	"restrictions" INTEGER NOT NULL
revoked	INTEGER	"revoked" INTEGER NOT NULL
expiration	INTEGER	"expiration" INTEGER
deleted	INTEGER	"deleted" INTEGER NOT NULL DEFAULT 0
storageID	TEXT	"storageID" TEXT
storageVersion	INTEGER	"storageVersion" INTEGER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER NOT NULL DEFAULT 0
deletedAt	INTEGER	"deletedAt" INTEGER
callsHistory		`CREATE TABLE callsHistory (callId TEXT PRIMARY KEY, peerId TEXT NOT NULL, -- conversation id (legacy) | uuid | groupId | roomId ringerId TEXT DEFAULT NULL, -- ringer uuid mode TEXT NOT NULL, -- enum "Direct" | "Group" type TEXT NOT NULL, -- enum "Audio" | "Video" | "Group" direction TEXT NOT NULL, -- enum "Incoming" | "Outgoing -- Direct: enum "Pending" | "Missed" | "Accepted" | "Deleted" -- Group: enum "GenericGroupCall" | "OutgoingRing" | "Ringing" | "Joined" | "Missed" | "Declined" | "Accepted" | "Deleted" status TEXT NOT NULL, timestamp INTEGER NOT NULL, startedById TEXT DEFAULT NULL, endedTimestamp INTEGER DEFAULT NULL, UNIQUE (callId, peerId) ON CONFLICT FAIL)`
callId	TEXT	"callId" TEXT
peerId	TEXT	"peerId" TEXT NOT NULL
ringerId	TEXT	"ringerId" TEXT DEFAULT NULL
mode	TEXT	"mode" TEXT NOT NULL
type	TEXT	"type" TEXT NOT NULL
direction	TEXT	"direction" TEXT NOT NULL
status	TEXT	"status" TEXT NOT NULL
timestamp	INTEGER	"timestamp" INTEGER NOT NULL
startedById	TEXT	"startedById" TEXT DEFAULT NULL
endedTimestamp	INTEGER	"endedTimestamp" INTEGER DEFAULT NULL
conversations		`CREATE TABLE conversations(id STRING PRIMARY KEY ASC, json TEXT, active_at INTEGER, type STRING, members TEXT, name TEXT, profileName TEXT , profileFamilyName TEXT, profileFullName TEXT, e164 TEXT, serviceId TEXT, groupId TEXT, profileLastFetchedAt INTEGER, expireTimerVersion INTEGER NOT NULL DEFAULT 1)`
id	STRING	"id" STRING
json	TEXT	"json" TEXT
active_at	INTEGER	"active_at" INTEGER
type	STRING	"type" STRING

Name	Type	Schema
members	TEXT	"members" TEXT
name	TEXT	"name" TEXT
profileName	TEXT	"profileName" TEXT
profileFamilyName	TEXT	"profileFamilyName" TEXT
profileFullName	TEXT	"profileFullName" TEXT
e164	TEXT	"e164" TEXT
serviceId	TEXT	"serviceId" TEXT
groupId	TEXT	"groupId" TEXT
profileLastFetchedAt	INTEGER	"profileLastFetchedAt" INTEGER
expireTimerVersion	INTEGER	"expireTimerVersion" INTEGER NOT NULL DEFAULT 1
defunctCallLinks		`CREATE TABLE defunctCallLinks ( roomId TEXT NOT NULL PRIMARY KEY, rootKey BLOB NOT NULL, adminKey BLOB , storageID TEXT, storageVersion INTEGER, storageUnknownFields BLOB, storageNeedsSync INTEGER NOT NULL DEFAULT 0) STRICT`
roomId	TEXT	"roomId" TEXT NOT NULL
rootKey	BLOB	"rootKey" BLOB NOT NULL
adminKey	BLOB	"adminKey" BLOB
storageID	TEXT	"storageID" TEXT
storageVersion	INTEGER	"storageVersion" INTEGER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER NOT NULL DEFAULT 0
donationReceipts		`CREATE TABLE donationReceipts( id TEXT NOT NULL PRIMARY KEY, currencyType TEXT NOT NULL, paymentAmount INTEGER NOT NULL, timestamp INTEGER NOT NULL ) STRICT`
id	TEXT	"id" TEXT NOT NULL
currencyType	TEXT	"currencyType" TEXT NOT NULL
paymentAmount	INTEGER	"paymentAmount" INTEGER NOT NULL
timestamp	INTEGER	"timestamp" INTEGER NOT NULL
edited_messages		`CREATE TABLE edited_messages( messageId STRING REFERENCES messages(id) ON DELETE CASCADE, sentAt INTEGER, readStatus INTEGER , conversationId STRING)`
messageId	STRING	"messageId" STRING
sentAt	INTEGER	"sentAt" INTEGER
readStatus	INTEGER	"readStatus" INTEGER
conversationId	STRING	"conversationId" STRING
emojis		`CREATE TABLE emojis( shortName TEXT PRIMARY KEY, lastUsage INTEGER )`
shortName	TEXT	"shortName" TEXT
lastUsage	INTEGER	"lastUsage" INTEGER
groupCallRingCancellations		`CREATE TABLE groupCallRingCancellations( ringId INTEGER PRIMARY KEY, createdAt INTEGER NOT NULL )`
ringId	INTEGER	"ringId" INTEGER
createdAt	INTEGER	"createdAt" INTEGER NOT NULL
groupSendCombinedEndorsement		`CREATE TABLE groupSendCombinedEndorsement ( groupId TEXT NOT NULL PRIMARY KEY, -- Only one endorsement per group expiration INTEGER NOT NULL, -- Unix timestamp in seconds endorsement BLOB NOT NULL ) STRICT`
groupId	TEXT	"groupId" TEXT NOT NULL
expiration	INTEGER	"expiration" INTEGER NOT NULL
endorsement	BLOB	"endorsement" BLOB NOT NULL
groupSendMemberEndorsement		`CREATE TABLE groupSendMemberEndorsement ( groupId TEXT NOT NULL, memberAci TEXT NOT NULL, expiration INTEGER NOT NULL, -- Unix timestamp in seconds endorsement BLOB NOT NULL, PRIMARY KEY (groupId, memberAci) -- Only one endorsement per group member ) STRICT`

Name	Type	Schema
groupId	TEXT	"groupId" TEXT NOT NULL
memberAci	TEXT	"memberAci" TEXT NOT NULL
expiration	INTEGER	"expiration" INTEGER NOT NULL
endorsement	BLOB	"endorsement" BLOB NOT NULL
identityKeys		`CREATE TABLE identityKeys( id STRING PRIMARY KEY ASC, json TEXT )`
id	STRING	"id" STRING
json	TEXT	"json" TEXT
items		`CREATE TABLE items( id STRING PRIMARY KEY ASC, json TEXT )`
id	STRING	"id" STRING
json	TEXT	"json" TEXT
jobs		`CREATE TABLE jobs( id TEXT PRIMARY KEY, queueType TEXT STRING NOT NULL, timestamp INTEGER NOT NULL, data STRING TEXT )`
id	TEXT	"id" TEXT
queueType	TEXT STRING	"queueType" TEXT STRING NOT NULL
timestamp	INTEGER	"timestamp" INTEGER NOT NULL
data	STRING TEXT	"data" STRING TEXT
kyberPreKeys		`CREATE TABLE kyberPreKeys( id STRING PRIMARY KEY NOT NULL, json TEXT NOT NULL, ourServiceId NUMBER GENERATED ALWAYS AS (json_extract(json, '$.ourServiceId')))`
id	STRING	"id" STRING NOT NULL
json	TEXT	"json" TEXT NOT NULL
ourServiceId	NUMBER	"ourServiceId" NUMBER GENERATED ALWAYS AS (json_extract("json", '$.ourServiceId')) VIRTUAL
mentions		`CREATE TABLE mentions ( messageId REFERENCES messages(id) ON DELETE CASCADE, mentionAci STRING, start INTEGER, length INTEGER )`
messageId		"messageId"
mentionAci	STRING	"mentionAci" STRING
start	INTEGER	"start" INTEGER
length	INTEGER	"length" INTEGER
message_attachments		`CREATE TABLE message_attachments (messageId TEXT NOT NULL REFERENCES messages(id) ON DELETE CASCADE, -- For editHistoryIndex to be part of the primary key, it cannot be NULL in strict tables. -- For that reason, we use a value of -1 to indicate that it is the root message (not in editHistory) editHistoryIndex INTEGER NOT NULL, attachmentType TEXT NOT NULL, -- 'long-message' | 'quote' | 'attachment' | 'preview' | 'contact' | 'sticker' orderInMessage INTEGER NOT NULL, conversationId TEXT NOT NULL, sentAt INTEGER NOT NULL, clientUuid TEXT, size INTEGER NOT NULL, contentType TEXT NOT NULL, path TEXT, plaintextHash TEXT, localKey TEXT, caption TEXT, fileName TEXT, blurHash TEXT, height INTEGER, width INTEGER, digest TEXT, key TEXT, downloadPath TEXT, version INTEGER, incrementalMac TEXT, incrementalMacChunkSize INTEGER, transitCdnKey TEXT, transitCdnNumber INTEGER, transitCdnUploadTimestamp INTEGER, backupCdnNumber INTEGER, thumbnailPath TEXT, thumbnailSize INTEGER, thumbnailContentType TEXT, thumbnailLocalKey TEXT, thumbnailVersion INTEGER, screenshotPath TEXT, screenshotSize INTEGER, screenshotContentType TEXT, screenshotLocalKey TEXT, screenshotVersion INTEGER, backupThumbnailPath TEXT, backupThumbnailSize INTEGER, backupThumbnailContentType TEXT, backupThumbnailLocalKey TEXT, backupThumbnailVersion INTEGER, storyTextAttachmentJson TEXT, localBackupPath TEXT, flags INTEGER, error INTEGER, wasTooBig INTEGER, isCorrupted INTEGER, copiedFromQuotedAttachment INTEGER, pending INTEGER, backfillError INTEGER, PRIMARY KEY (messageId, editHistoryIndex,`

Name	Type	Schema
		attachmentType, orderInMessage)) STRICT
messageId	TEXT	"messageId" TEXT NOT NULL
editHistoryIndex	INTEGER	"editHistoryIndex" INTEGER NOT NULL
attachmentType	TEXT	"attachmentType" TEXT NOT NULL
orderInMessage	INTEGER	"orderInMessage" INTEGER NOT NULL
conversationId	TEXT	"conversationId" TEXT NOT NULL
sentAt	INTEGER	"sentAt" INTEGER NOT NULL
clientUuid	TEXT	"clientUuid" TEXT
size	INTEGER	"size" INTEGER NOT NULL
contentType	TEXT	"contentType" TEXT NOT NULL
path	TEXT	"path" TEXT
plaintextHash	TEXT	"plaintextHash" TEXT
localKey	TEXT	"localKey" TEXT
caption	TEXT	"caption" TEXT
fileName	TEXT	"fileName" TEXT
blurHash	TEXT	"blurHash" TEXT
height	INTEGER	"height" INTEGER
width	INTEGER	"width" INTEGER
digest	TEXT	"digest" TEXT
key	TEXT	"key" TEXT
downloadPath	TEXT	"downloadPath" TEXT
version	INTEGER	"version" INTEGER
incrementalMac	TEXT	"incrementalMac" TEXT
incrementalMacChunkSize	INTEGER	"incrementalMacChunkSize" INTEGER
transitCdnKey	TEXT	"transitCdnKey" TEXT
transitCdnNumber	INTEGER	"transitCdnNumber" INTEGER
transitCdnUploadTimestamp	INTEGER	"transitCdnUploadTimestamp" INTEGER
backupCdnNumber	INTEGER	"backupCdnNumber" INTEGER
thumbnailPath	TEXT	"thumbnailPath" TEXT
thumbnailSize	INTEGER	"thumbnailSize" INTEGER
thumbnailContentType	TEXT	"thumbnailContentType" TEXT
thumbnailLocalKey	TEXT	"thumbnailLocalKey" TEXT
thumbnailVersion	INTEGER	"thumbnailVersion" INTEGER
screenshotPath	TEXT	"screenshotPath" TEXT
screenshotSize	INTEGER	"screenshotSize" INTEGER
screenshotContentType	TEXT	"screenshotContentType" TEXT
screenshotLocalKey	TEXT	"screenshotLocalKey" TEXT
screenshotVersion	INTEGER	"screenshotVersion" INTEGER
backupThumbnailPath	TEXT	"backupThumbnailPath" TEXT
backupThumbnailSize	INTEGER	"backupThumbnailSize" INTEGER
backupThumbnailContentType	TEXT	"backupThumbnailContentType" TEXT
backupThumbnailLocalKey	TEXT	"backupThumbnailLocalKey" TEXT
backupThumbnailVersion	INTEGER	"backupThumbnailVersion" INTEGER
storyTextAttachmentJson	TEXT	"storyTextAttachmentJson" TEXT
localBackupPath	TEXT	"localBackupPath" TEXT
flags	INTEGER	"flags" INTEGER
error	INTEGER	"error" INTEGER
wasTooBig	INTEGER	"wasTooBig" INTEGER
isCorrupted	INTEGER	"isCorrupted" INTEGER
copiedFromQuotedAttachment	INTEGER	"copiedFromQuotedAttachment" INTEGER

Name	Type	Schema
pending	INTEGER	"pending" INTEGER
backfillError	INTEGER	"backfillError" INTEGER
messages		`CREATE TABLE messages(rowid INTEGER PRIMARY KEY ASC, id STRING UNIQUE, json TEXT, readStatus INTEGER, expires_at INTEGER, sent_at INTEGER, schemaVersion INTEGER, conversationId STRING, received_at INTEGER, hasAttachments INTEGER, hasFileAttachments INTEGER, hasVisualMediaAttachments INTEGER, expireTimer INTEGER, expirationStartTimestamp INTEGER, type STRING, body TEXT, messageTimer INTEGER, messageTimerStart INTEGER, messageTimerExpiresAt INTEGER, isErased INTEGER, isViewOnce INTEGER, sourceServiceId TEXT, serverGuid STRING NULL, sourceDevice INTEGER, storyId STRING, isStory INTEGER GENERATED ALWAYS AS (type IS 'story'), isChangeCreatedByUs INTEGER NOT NULL DEFAULT 0, isTimerChangeFromSync INTEGER GENERATED ALWAYS AS (json_extract(json, '$.expirationTimerUpdate.fromSync') IS 1), seenStatus NUMBER default 0, storyDistributionListId STRING, expiresAt INT GENERATED ALWAYS AS (ifnull(expirationStartTimestamp + (expireTimer * 1000), 9007199254740991)), isUserInitiatedMessage INTEGER GENERATED ALWAYS AS (type IS NULL OR type NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'group-v2-change', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change')), mentionsMe INTEGER NOT NULL DEFAULT 0, isGroupLeaveEvent INTEGER GENERATED ALWAYS AS (type IS 'group-v2-change' AND json_array_length(json_extract(json, '$.groupV2Change.details')) IS 1 AND json_extract(json, '$.groupV2Change.details[0].type') IS 'member-remove' AND json_extract(json, '$.groupV2Change.from') IS NOT NULL AND json_extract(json, '$.groupV2Change.from') IS json_extract(json, '$.groupV2Change.details[0].aci')), isGroupLeaveEventFromOther INTEGER GENERATED ALWAYS AS (isGroupLeaveEvent IS 1 AND isChangeCreatedByUs IS 0), callId TEXT GENERATED ALWAYS AS (json_extract(json, '$.callId')), shouldAffectPreview INTEGER GENERATED ALWAYS AS (type IS NULL OR type NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change') AND NOT (type IS 'message-request-response-event' AND json_extract(json, '$.messageRequestResponseEvent') IN ('ACCEPT', 'BLOCK', 'UNBLOCK'))), shouldAffectActivity INTEGER GENERATED ALWAYS AS (type IS NULL OR type NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change') AND NOT (type IS 'message-request-response-event' AND json_extract(json, '$.messageRequestResponseEvent') IN ('ACCEPT', 'BLOCK', 'UNBLOCK'))), isAddressableMessage INTEGER GENERATED ALWAYS AS (type IS NULL OR type IN ('incoming', 'outgoing')), timestamp INTEGER, received_at_ms INTEGER, unidentifiedDeliveryReceived INTEGER, serverTimestamp INTEGER, source TEXT)`
rowid	INTEGER	"rowid" INTEGER
id	STRING	"id" STRING UNIQUE
json	TEXT	"json" TEXT
readStatus	INTEGER	"readStatus" INTEGER
expires_at	INTEGER	"expires_at" INTEGER
sent_at	INTEGER	"sent_at" INTEGER
schemaVersion	INTEGER	"schemaVersion" INTEGER

Name	Type	Schema
conversationId	STRING	"conversationId" STRING
received_at	INTEGER	"received_at" INTEGER
hasAttachments	INTEGER	"hasAttachments" INTEGER
hasFileAttachments	INTEGER	"hasFileAttachments" INTEGER
hasVisualMediaAttachments	INTEGER	"hasVisualMediaAttachments" INTEGER
expireTimer	INTEGER	"expireTimer" INTEGER
expirationStartTimestamp	INTEGER	"expirationStartTimestamp" INTEGER
type	STRING	"type" STRING
body	TEXT	"body" TEXT
messageTimer	INTEGER	"messageTimer" INTEGER
messageTimerStart	INTEGER	"messageTimerStart" INTEGER
messageTimerExpiresAt	INTEGER	"messageTimerExpiresAt" INTEGER
isErased	INTEGER	"isErased" INTEGER
isViewOnce	INTEGER	"isViewOnce" INTEGER
sourceServiceId	TEXT	"sourceServiceId" TEXT
serverGuid	STRING	"serverGuid" STRING
sourceDevice	INTEGER	"sourceDevice" INTEGER
storyId	STRING	"storyId" STRING
isStory	INTEGER	"isStory" INTEGER GENERATED ALWAYS AS ("type" IS 'story') VIRTUAL
isChangeCreatedByUs	INTEGER	"isChangeCreatedByUs" INTEGER NOT NULL DEFAULT 0
isTimerChangeFromSync	INTEGER	"isTimerChangeFromSync" INTEGER GENERATED ALWAYS AS (json_extract("json", '$.expirationTimerUpdate.fromSync') IS 1) VIRTUAL
seenStatus	NUMBER	"seenStatus" NUMBER DEFAULT 0
storyDistributionListId	STRING	"storyDistributionListId" STRING
expiresAt	INT	"expiresAt" INT GENERATED ALWAYS AS (ifnull("expirationStartTimestamp" + ("expireTimer" * 1000), 9007199254740991)) VIRTUAL
isUserInitiatedMessage	INTEGER	"isUserInitiatedMessage" INTEGER GENERATED ALWAYS AS ("type" IS NULL OR "type" NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'group-v2-change', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change')) VIRTUAL
mentionsMe	INTEGER	"mentionsMe" INTEGER NOT NULL DEFAULT 0
isGroupLeaveEvent	INTEGER	"isGroupLeaveEvent" INTEGER GENERATED ALWAYS AS ("type" IS 'group-v2-change' AND json_array_length(json_extract("json", '$.groupV2Change.details')) IS 1 AND json_extract("json", '$.groupV2Change.details[0].type') IS 'member-remove' AND json_extract("json", '$.groupV2Change.from') IS NOT NULL AND json_extract("json", '$.groupV2Change.from') IS json_extract("json", '$.groupV2Change.details[0].aci')) VIRTUAL
isGroupLeaveEventFromOther	INTEGER	"isGroupLeaveEventFromOther" INTEGER GENERATED ALWAYS AS ("isGroupLeaveEvent" IS 1 AND "isChangeCreatedByUs" IS 0) VIRTUAL
callId	TEXT	"callId" TEXT GENERATED ALWAYS AS (json_extract("json", '$.callId')) VIRTUAL
shouldAffectPreview	INTEGER	"shouldAffectPreview" INTEGER GENERATED ALWAYS AS ("type" IS NULL OR "type" NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change') AND NOT ("type" IS 'message-request-response-event' AND json_extract("json", '$.messageRequestResponseEvent') IN ('ACCEPT', 'BLOCK', 'UNBLOCK'))) VIRTUAL
shouldAffectActivity	INTEGER	"shouldAffectActivity" INTEGER GENERATED ALWAYS AS ("type" IS NULL OR "type" NOT IN ('change-number-notification', 'contact-removed-notification', 'conversation-merge', 'group-v1-migration', 'keychange', 'message-history-unsynced', 'profile-change', 'story', 'universal-timer-notification', 'verified-change') AND NOT ("type" IS 'message-request-response-event' AND json_extract("json", '$.messageRequestResponseEvent') IN ('ACCEPT', 'BLOCK', 'UNBLOCK')))

Name	Type	Schema
		VIRTUAL
isAddressableMessage	INTEGER	"isAddressableMessage" INTEGER GENERATED ALWAYS AS ("type" IS NULL OR "type" IN ('incoming', 'outgoing')) VIRTUAL
timestamp	INTEGER	"timestamp" INTEGER
received_at_ms	INTEGER	"received_at_ms" INTEGER
unidentifiedDeliveryReceived	INTEGER	"unidentifiedDeliveryReceived" INTEGER
serverTimestamp	INTEGER	"serverTimestamp" INTEGER
source	TEXT	"source" TEXT
messages_fts		`CREATE VIRTUAL TABLE messages_fts USING fts5( body, tokenize = 'signal_tokenizer' )`
messages_fts_config		`CREATE TABLE 'messages_fts_config'(k PRIMARY KEY, v) WITHOUT ROWID`
k		"k"
v		"v"
messages_fts_content		`CREATE TABLE 'messages_fts_content'(id INTEGER PRIMARY KEY, c0)`
id	INTEGER	"id" INTEGER
c0		"c0"
messages_fts_data		`CREATE TABLE 'messages_fts_data'(id INTEGER PRIMARY KEY, block BLOB)`
id	INTEGER	"id" INTEGER
block	BLOB	"block" BLOB
messages_fts_docsize		`CREATE TABLE 'messages_fts_docsize'(id INTEGER PRIMARY KEY, sz BLOB)`
id	INTEGER	"id" INTEGER
sz	BLOB	"sz" BLOB
messages_fts_idx		`CREATE TABLE 'messages_fts_idx'(segid, term, pgno, PRIMARY KEY(segid, term)) WITHOUT ROWID`
segid		"segid"
term		"term"
pgno		"pgno"
notificationProfiles		`CREATE TABLE notificationProfiles(id TEXT PRIMARY KEY NOT NULL, name TEXT NOT NULL, emoji TEXT, /* A numeric representation of a color, like 0xAARRGGBB */ color INTEGER NOT NULL, createdAtMs INTEGER NOT NULL, allowAllCalls INTEGER NOT NULL, allowAllMentions INTEGER NOT NULL, /* A JSON array of conversationId strings */ allowedMembersJson TEXT NOT NULL, scheduleEnabled INTEGER NOT NULL, /* 24-hour clock int, 0000-2359 (e.g., 15, 900, 1130, 2345) */ scheduleStartTime INTEGER, scheduleEndTime INTEGER, /* A JSON object with true/false for each of the numbers in the Protobuf enum */ scheduleDaysEnabledJson TEXT, deletedAtTimestampMs INTEGER, storageID TEXT, storageVersion INTEGER, storageUnknownFields BLOB, storageNeedsSync INTEGER NOT NULL DEFAULT 0) STRICT`
id	TEXT	"id" TEXT NOT NULL
name	TEXT	"name" TEXT NOT NULL
emoji	TEXT	"emoji" TEXT
color	INTEGER	"color" INTEGER NOT NULL
createdAtMs	INTEGER	"createdAtMs" INTEGER NOT NULL
allowAllCalls	INTEGER	"allowAllCalls" INTEGER NOT NULL
allowAllMentions	INTEGER	"allowAllMentions" INTEGER NOT NULL
allowedMembersJson	TEXT	"allowedMembersJson" TEXT NOT NULL
scheduleEnabled	INTEGER	"scheduleEnabled" INTEGER NOT NULL
scheduleStartTime	INTEGER	"scheduleStartTime" INTEGER
scheduleEndTime	INTEGER	"scheduleEndTime" INTEGER
scheduleDaysEnabledJson	TEXT	"scheduleDaysEnabledJson" TEXT

Name	Type	Schema
deletedAtTimestampMs	INTEGER	"deletedAtTimestampMs" INTEGER
storageID	TEXT	"storageID" TEXT
storageVersion	INTEGER	"storageVersion" INTEGER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER NOT NULL DEFAULT 0
preKeys		`CREATE TABLE preKeys( id STRING PRIMARY KEY ASC, json TEXT , ourServiceId NUMBER GENERATED ALWAYS AS (json_extract(json, '$.ourServiceId')))`
id	STRING	"id" STRING
json	TEXT	"json" TEXT
ourServiceId	NUMBER	"ourServiceId" NUMBER GENERATED ALWAYS AS (json_extract("json", '$.ourServiceId')) VIRTUAL
reactions		`CREATE TABLE reactions( conversationId STRING, emoji STRING, fromId STRING, messageReceivedAt INTEGER, targetAuthorAci STRING, targetTimestamp INTEGER, unread INTEGER , messageId STRING, timestamp NUMBER)`
conversationId	STRING	"conversationId" STRING
emoji	STRING	"emoji" STRING
fromId	STRING	"fromId" STRING
messageReceivedAt	INTEGER	"messageReceivedAt" INTEGER
targetAuthorAci	STRING	"targetAuthorAci" STRING
targetTimestamp	INTEGER	"targetTimestamp" INTEGER
unread	INTEGER	"unread" INTEGER
messageId	STRING	"messageId" STRING
timestamp	NUMBER	"timestamp" NUMBER
recentGifs		`CREATE TABLE recentGifs (id TEXT NOT NULL PRIMARY KEY, title TEXT NOT NULL, description TEXT NOT NULL, previewMedia_url TEXT NOT NULL, previewMedia_width INTEGER NOT NULL, previewMedia_height INTEGER NOT NULL, attachmentMedia_url TEXT NOT NULL, attachmentMedia_width INTEGER NOT NULL, attachmentMedia_height INTEGER NOT NULL, lastUsedAt INTEGER NOT NULL) STRICT`
id	TEXT	"id" TEXT NOT NULL
title	TEXT	"title" TEXT NOT NULL
description	TEXT	"description" TEXT NOT NULL
previewMedia_url	TEXT	"previewMedia_url" TEXT NOT NULL
previewMedia_width	INTEGER	"previewMedia_width" INTEGER NOT NULL
previewMedia_height	INTEGER	"previewMedia_height" INTEGER NOT NULL
attachmentMedia_url	TEXT	"attachmentMedia_url" TEXT NOT NULL
attachmentMedia_width	INTEGER	"attachmentMedia_width" INTEGER NOT NULL
attachmentMedia_height	INTEGER	"attachmentMedia_height" INTEGER NOT NULL
lastUsedAt	INTEGER	"lastUsedAt" INTEGER NOT NULL
sendLogMessageIds		`CREATE TABLE sendLogMessageIds( payloadId INTEGER NOT NULL, messageId STRING NOT NULL, PRIMARY KEY (payloadId, messageId), CONSTRAINT sendLogMessageIdsForeignKey FOREIGN KEY (payloadId) REFERENCES sendLogPayloads(id) ON DELETE CASCADE )`
payloadId	INTEGER	"payloadId" INTEGER NOT NULL
messageId	STRING	"messageId" STRING NOT NULL
sendLogPayloads		`CREATE TABLE sendLogPayloads( id INTEGER PRIMARY KEY ASC, timestamp INTEGER NOT NULL, contentHint INTEGER NOT NULL, proto BLOB NOT NULL , urgent INTEGER, hasPniSignatureMessage INTEGER DEFAULT 0 NOT NULL)`
id	INTEGER	"id" INTEGER
timestamp	INTEGER	"timestamp" INTEGER NOT NULL
contentHint	INTEGER	"contentHint" INTEGER NOT NULL
proto	BLOB	"proto" BLOB NOT NULL

Name	Type	Schema
urgent	INTEGER	"urgent" INTEGER
hasPniSignatureMessage	INTEGER	"hasPniSignatureMessage" INTEGER NOT NULL DEFAULT 0
sendLogRecipients		`CREATE TABLE sendLogRecipients( payloadId INTEGER NOT NULL, recipientServiceId STRING NOT NULL, deviceId INTEGER NOT NULL, PRIMARY KEY (payloadId, recipientServiceId, deviceId), CONSTRAINT sendLogRecipientsForeignKey FOREIGN KEY (payloadId) REFERENCES sendLogPayloads(id) ON DELETE CASCADE )`
payloadId	INTEGER	"payloadId" INTEGER NOT NULL
recipientServiceId	STRING	"recipientServiceId" STRING NOT NULL
deviceId	INTEGER	"deviceId" INTEGER NOT NULL
senderKeys		`CREATE TABLE senderKeys( id TEXT PRIMARY KEY NOT NULL, senderId TEXT NOT NULL, distributionId TEXT NOT NULL, data BLOB NOT NULL, lastUpdatedDate NUMBER NOT NULL )`
id	TEXT	"id" TEXT NOT NULL
senderId	TEXT	"senderId" TEXT NOT NULL
distributionId	TEXT	"distributionId" TEXT NOT NULL
data	BLOB	"data" BLOB NOT NULL
lastUpdatedDate	NUMBER	"lastUpdatedDate" NUMBER NOT NULL
sessions		`CREATE TABLE sessions ( id TEXT NOT NULL PRIMARY KEY, ourServiceId TEXT NOT NULL, serviceId TEXT NOT NULL, conversationId TEXT NOT NULL, deviceId INTEGER NOT NULL, record BLOB NOT NULL ) STRICT`
id	TEXT	"id" TEXT NOT NULL
ourServiceId	TEXT	"ourServiceId" TEXT NOT NULL
serviceId	TEXT	"serviceId" TEXT NOT NULL
conversationId	TEXT	"conversationId" TEXT NOT NULL
deviceId	INTEGER	"deviceId" INTEGER NOT NULL
record	BLOB	"record" BLOB NOT NULL
signedPreKeys		`CREATE TABLE signedPreKeys( id STRING PRIMARY KEY ASC, json TEXT , ourServiceId NUMBER GENERATED ALWAYS AS (json_extract(json, '$.ourServiceId')))`
id	STRING	"id" STRING
json	TEXT	"json" TEXT
ourServiceId	NUMBER	"ourServiceId" NUMBER GENERATED ALWAYS AS (json_extract("json", '$.ourServiceId')) VIRTUAL
sqlite_stat1		`CREATE TABLE sqlite_stat1(tbl,idx,stat)`
tbl		"tbl"
idx		"idx"
stat		"stat"
sqlite_stat4		`CREATE TABLE sqlite_stat4(tbl,idx,neq,nlt,ndlt,sample)`
tbl		"tbl"
idx		"idx"
neq		"neq"
nlt		"nlt"
ndlt		"ndlt"
sample		"sample"
sticker_packs		`CREATE TABLE sticker_packs(id TEXT PRIMARY KEY, key TEXT NOT NULL, author STRING, coverStickerId INTEGER, createdAt INTEGER, downloadAttempts INTEGER, installedAt INTEGER, lastUsed INTEGER, status STRING, stickerCount INTEGER, title STRING , attemptedStatus STRING, position INTEGER DEFAULT 0 NOT NULL, storageID STRING, storageVersion INTEGER, storageUnknownFields BLOB, storageNeedsSync INTEGER DEFAULT 0 NOT NULL)`
id	TEXT	"id" TEXT
key	TEXT	"key" TEXT NOT NULL

Name	Type	Schema
author	STRING	"author" STRING
coverStickerId	INTEGER	"coverStickerId" INTEGER
createdAt	INTEGER	"createdAt" INTEGER
downloadAttempts	INTEGER	"downloadAttempts" INTEGER
installedAt	INTEGER	"installedAt" INTEGER
lastUsed	INTEGER	"lastUsed" INTEGER
status	STRING	"status" STRING
stickerCount	INTEGER	"stickerCount" INTEGER
title	STRING	"title" STRING
attemptedStatus	STRING	"attemptedStatus" STRING
position	INTEGER	"position" INTEGER NOT NULL DEFAULT 0
storageID	STRING	"storageID" STRING
storageVersion	INTEGER	"storageVersion" INTEGER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER NOT NULL DEFAULT 0
sticker_references		`CREATE TABLE sticker_references( messageId STRING, packId TEXT, stickerId INTEGER NOT NULL DEFAULT -1, isUnresolved INTEGER NOT NULL DEFAULT 0, CONSTRAINT sticker_references_fk FOREIGN KEY(packId) REFERENCES sticker_packs(id) ON DELETE CASCADE )`
messageId	STRING	"messageId" STRING
packId	TEXT	"packId" TEXT
stickerId	INTEGER	"stickerId" INTEGER NOT NULL DEFAULT -1
isUnresolved	INTEGER	"isUnresolved" INTEGER NOT NULL DEFAULT 0
stickers		`CREATE TABLE stickers(id INTEGER NOT NULL, packId TEXT NOT NULL, emoji STRING, height INTEGER, isCoverOnly INTEGER, lastUsed INTEGER, path STRING, width INTEGER, version INTEGER NOT NULL DEFAULT 1, localKey TEXT, size INTEGER, PRIMARY KEY (id, packId), CONSTRAINT stickers_fk FOREIGN KEY (packId) REFERENCES sticker_packs(id) ON DELETE CASCADE)`
id	INTEGER	"id" INTEGER NOT NULL
packId	TEXT	"packId" TEXT NOT NULL
emoji	STRING	"emoji" STRING
height	INTEGER	"height" INTEGER
isCoverOnly	INTEGER	"isCoverOnly" INTEGER
lastUsed	INTEGER	"lastUsed" INTEGER
path	STRING	"path" STRING
width	INTEGER	"width" INTEGER
version	INTEGER	"version" INTEGER NOT NULL DEFAULT 1
localKey	TEXT	"localKey" TEXT
size	INTEGER	"size" INTEGER
storyDistributionMembers		`CREATE TABLE storyDistributionMembers( listId STRING NOT NULL REFERENCES storyDistributions(id) ON DELETE CASCADE ON UPDATE CASCADE, serviceId STRING NOT NULL, PRIMARY KEY (listId, serviceId) )`
listId	STRING	"listId" STRING NOT NULL
serviceId	STRING	"serviceId" STRING NOT NULL
storyDistributions		`CREATE TABLE storyDistributions( id STRING PRIMARY KEY NOT NULL, name TEXT, senderKeyInfoJson STRING , deletedAtTimestamp INTEGER, allowsReplies INTEGER, isBlockList INTEGER, storageID STRING, storageVersion INTEGER, storageUnknownFields BLOB, storageNeedsSync INTEGER)`
id	STRING	"id" STRING NOT NULL
name	TEXT	"name" TEXT
senderKeyInfoJson	STRING	"senderKeyInfoJson" STRING

Name	Type	Schema
deletedAtTimestamp	INTEGER	"deletedAtTimestamp" INTEGER
allowsReplies	INTEGER	"allowsReplies" INTEGER
isBlockList	INTEGER	"isBlockList" INTEGER
storageID	STRING	"storageID" STRING
storageVersion	INTEGER	"storageVersion" INTEGER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER
storyReads		`CREATE TABLE storyReads ( authorId STRING NOT NULL, conversationId STRING NOT NULL, storyId STRING NOT NULL, storyReadDate NUMBER NOT NULL, PRIMARY KEY (authorId, storyId) )`
authorId	STRING	"authorId" STRING NOT NULL
conversationId	STRING	"conversationId" STRING NOT NULL
storyId	STRING	"storyId" STRING NOT NULL
storyReadDate	NUMBER	"storyReadDate" NUMBER NOT NULL
syncTasks		`CREATE TABLE syncTasks( id TEXT PRIMARY KEY NOT NULL, attempts INTEGER NOT NULL, createdAt INTEGER NOT NULL, data TEXT NOT NULL, envelopeId TEXT NOT NULL, sentAt INTEGER NOT NULL, type TEXT NOT NULL ) STRICT`
id	TEXT	"id" TEXT NOT NULL
attempts	INTEGER	"attempts" INTEGER NOT NULL
createdAt	INTEGER	"createdAt" INTEGER NOT NULL
data	TEXT	"data" TEXT NOT NULL
envelopeId	TEXT	"envelopeId" TEXT NOT NULL
sentAt	INTEGER	"sentAt" INTEGER NOT NULL
type	TEXT	"type" TEXT NOT NULL
uninstalled_sticker_packs		`CREATE TABLE uninstalled_sticker_packs ( id STRING NOT NULL PRIMARY KEY, uninstalledAt NUMBER NOT NULL, storageID STRING, storageVersion NUMBER, storageUnknownFields BLOB, storageNeedsSync INTEGER NOT NULL )`
id	STRING	"id" STRING NOT NULL
uninstalledAt	NUMBER	"uninstalledAt" NUMBER NOT NULL
storageID	STRING	"storageID" STRING
storageVersion	NUMBER	"storageVersion" NUMBER
storageUnknownFields	BLOB	"storageUnknownFields" BLOB
storageNeedsSync	INTEGER	"storageNeedsSync" INTEGER NOT NULL
unprocessed		`CREATE TABLE unprocessed(id TEXT NOT NULL PRIMARY KEY ASC, type INTEGER NOT NULL, timestamp INTEGER NOT NULL, attempts INTEGER NOT NULL, receivedAtCounter INTEGER NOT NULL, urgent INTEGER NOT NULL, story INTEGER NOT NULL, serverGuid TEXT NOT NULL, serverTimestamp INTEGER NOT NULL, isEncrypted INTEGER NOT NULL, content BLOB NOT NULL, messageAgeSec INTEGER NOT NULL, destinationServiceId TEXT NOT NULL, -- Not present for 1:1 messages and not sealed messages groupId TEXT, -- Not present for sealed envelopes reportingToken BLOB, source TEXT, sourceServiceId TEXT, updatedPni TEXT , sourceDevice INTEGER, receivedAtDate INTEGER DEFAULT 0 NOT NULL) STRICT`
id	TEXT	"id" TEXT NOT NULL
type	INTEGER	"type" INTEGER NOT NULL
timestamp	INTEGER	"timestamp" INTEGER NOT NULL
attempts	INTEGER	"attempts" INTEGER NOT NULL
receivedAtCounter	INTEGER	"receivedAtCounter" INTEGER NOT NULL
urgent	INTEGER	"urgent" INTEGER NOT NULL
story	INTEGER	"story" INTEGER NOT NULL
serverGuid	TEXT	"serverGuid" TEXT NOT NULL

Name	Type	Schema
serverTimestamp	INTEGER	"serverTimestamp" INTEGER NOT NULL
isEncrypted	INTEGER	"isEncrypted" INTEGER NOT NULL
content	BLOB	"content" BLOB NOT NULL
messageAgeSec	INTEGER	"messageAgeSec" INTEGER NOT NULL
destinationServiceId	TEXT	"destinationServiceId" TEXT NOT NULL
groupId	TEXT	"groupId" TEXT
reportingToken	BLOB	"reportingToken" BLOB
source	TEXT	"source" TEXT
sourceServiceId	TEXT	"sourceServiceId" TEXT
updatedPni	TEXT	"updatedPni" TEXT
sourceDevice	INTEGER	"sourceDevice" INTEGER
receivedAtDate	INTEGER	"receivedAtDate" INTEGER NOT NULL DEFAULT 0

Indices (80)

Name	Type	Schema
attachment_backup_jobs_receivedAt		`CREATE INDEX attachment_backup_jobs_receivedAt ON attachment_backup_jobs ( receivedAt )`
receivedAt		"receivedAt"
attachment_backup_jobs_type_receivedAt		`CREATE INDEX attachment_backup_jobs_type_receivedAt ON attachment_backup_jobs ( type, receivedAt )`
type		"type"
receivedAt		"receivedAt"
attachment_downloads_active_messageId		`CREATE INDEX attachment_downloads_active_messageId ON attachment_downloads ( active, messageId )`
active		"active"
messageId		"messageId"
attachment_downloads_active_receivedAt		`CREATE INDEX attachment_downloads_active_receivedAt ON attachment_downloads ( active, receivedAt )`
active		"active"
receivedAt		"receivedAt"
attachment_downloads_active_source_receivedAt		`CREATE INDEX attachment_downloads_active_source_receivedAt ON attachment_downloads ( active, source, receivedAt )`
active		"active"
source		"source"
receivedAt		"receivedAt"
attachment_downloads_messageId		`CREATE INDEX attachment_downloads_messageId ON attachment_downloads ( messageId )`
messageId		"messageId"
attachment_downloads_source_ciphertextSize		`CREATE INDEX attachment_downloads_source_ciphertextSize ON attachment_downloads ( source, ciphertextSize )`
source		"source"
ciphertextSize		"ciphertextSize"
callLinks_adminKey		`CREATE INDEX callLinks_adminKey ON callLinks (adminKey)`
adminKey		"adminKey"
callLinks_deleted		`CREATE INDEX callLinks_deleted ON callLinks (deleted, roomId)`
deleted		"deleted"

Name	Type	Schema
roomId		"roomId"
callsHistory_byConversation_order		`CREATE INDEX callsHistory_byConversation_order ON callsHistory (peerId, timestamp DESC, callId)`
peerId		"peerId"
timestamp	DESC	"timestamp" DESC
callId		"callId"
callsHistory_callAndGroupInfo_optimize		`CREATE INDEX callsHistory_callAndGroupInfo_optimize on callsHistory ( direction, peerId, timestamp DESC, status )`
direction		"direction"
peerId		"peerId"
timestamp	DESC	"timestamp" DESC
status		"status"
callsHistory_incoming_missed		`CREATE INDEX callsHistory_incoming_missed ON callsHistory (callId, status, direction) WHERE status IS 'Missed' AND direction IS 'Incoming'`
callId		"callId"
status		"status"
direction		"direction"
callsHistory_order		`CREATE INDEX callsHistory_order ON callsHistory (timestamp DESC, callId, peerId)`
timestamp	DESC	"timestamp" DESC
callId		"callId"
peerId		"peerId"
conversations_active		`CREATE INDEX conversations_active ON conversations ( active_at ) WHERE active_at IS NOT NULL`
active_at		"active_at"
conversations_e164		`CREATE INDEX conversations_e164 ON conversations(e164)`
e164		"e164"
conversations_groupId		`CREATE INDEX conversations_groupId ON conversations(groupId)`
groupId		"groupId"
conversations_serviceId		`CREATE INDEX conversations_serviceId ON conversations(serviceId)`
serviceId		"serviceId"
conversations_type		`CREATE INDEX conversations_type ON conversations ( type ) WHERE type IS NOT NULL`
type		"type"
donationReceipts_byTimestamp		`CREATE INDEX donationReceipts_byTimestamp on donationReceipts(timestamp)`
timestamp		"timestamp"
edited_messages_messageId		`CREATE INDEX edited_messages_messageId ON edited_messages(messageId)`
messageId		"messageId"
edited_messages_sent_at		`CREATE INDEX edited_messages_sent_at ON edited_messages (sentAt)`
sentAt		"sentAt"
edited_messages_unread		`CREATE INDEX edited_messages_unread ON edited_messages (readStatus, conversationId)`
readStatus		"readStatus"
conversationId		"conversationId"
		`CREATE INDEX emojis_lastUsage ON emojis ( lastUsage )`

Name	Type	Schema
emojis_lastUsage		
lastUsage		"lastUsage"
expiring_message_by_conversation_and_received_at		`CREATE INDEX expiring_message_by_conversation_and_received_at ON messages ( conversationId, storyId, expirationStartTimestamp, expireTimer, received_at ) WHERE isStory IS 0 AND type IS 'incoming'`
conversationId		"conversationId"
storyId		"storyId"
expirationStartTimestamp		"expirationStartTimestamp"
expireTimer		"expireTimer"
received_at		"received_at"
jobs_timestamp		`CREATE INDEX jobs_timestamp ON jobs (timestamp)`
timestamp		"timestamp"
kyberPreKeys_ourServiceId		`CREATE INDEX kyberPreKeys_ourServiceId ON kyberPreKeys (ourServiceId)`
ourServiceId		"ourServiceId"
mentions_aci		`CREATE INDEX mentions_aci ON mentions (mentionAci)`
mentionAci		"mentionAci"
mentions_messageId		`CREATE INDEX mentions_messageId ON mentions(messageId)`
messageId		"messageId"
message_user_initiated		`CREATE INDEX message_user_initiated ON messages (conversationId, isUserInitiatedMessage)`
conversationId		"conversationId"
isUserInitiatedMessage		"isUserInitiatedMessage"
messages_activity		`CREATE INDEX messages_activity ON messages (conversationId, shouldAffectActivity, isTimerChangeFromSync, isGroupLeaveEventFromOther, received_at, sent_at)`
conversationId		"conversationId"
shouldAffectActivity		"shouldAffectActivity"
isTimerChangeFromSync		"isTimerChangeFromSync"
isGroupLeaveEventFromOther		"isGroupLeaveEventFromOther"
received_at		"received_at"
sent_at		"sent_at"
messages_by_date_addressable		`CREATE INDEX messages_by_date_addressable ON messages ( conversationId, isAddressableMessage, received_at, sent_at )`
conversationId		"conversationId"
isAddressableMessage		"isAddressableMessage"
received_at		"received_at"
sent_at		"sent_at"
messages_by_date_addressable_nondisappearing		`CREATE INDEX messages_by_date_addressable_nondisappearing ON messages ( conversationId, isAddressableMessage, received_at, sent_at ) WHERE expireTimer IS NULL`
conversationId		"conversationId"
isAddressableMessage		"isAddressableMessage"
received_at		"received_at"
sent_at		"sent_at"
messages_by_distribution_list		`CREATE INDEX messages_by_distribution_list ON messages(storyDistributionListId, received_at) WHERE storyDistributionListId IS NOT NULL`
storyDistributionListId		"storyDistributionListId"

Name	Type	Schema
received_at		"received_at"
messages_by_storyId		`CREATE INDEX messages_by_storyId ON messages (storyId)`
storyId		"storyId"
messages_call		`CREATE INDEX messages_call ON messages (type, conversationId, callId, sent_at) WHERE type IS 'call-history'`
type		"type"
conversationId		"conversationId"
callId		"callId"
sent_at		"sent_at"
messages_callHistory_markReadBefore		`CREATE INDEX messages_callHistory_markReadBefore ON messages (type, seenStatus, received_at DESC) WHERE type IS 'call-history'`
type		"type"
seenStatus		"seenStatus"
received_at	DESC	"received_at" DESC
messages_callHistory_markReadByConversationBefore		`CREATE INDEX messages_callHistory_markReadByConversationBefore ON messages (type, conversationId, seenStatus, sent_at DESC) WHERE type IS 'call-history'`
type		"type"
conversationId		"conversationId"
seenStatus		"seenStatus"
sent_at	DESC	"sent_at" DESC
messages_callHistory_seenStatus		`CREATE INDEX messages_callHistory_seenStatus ON messages (type, seenStatus) WHERE type IS 'call-history'`
type		"type"
seenStatus		"seenStatus"
messages_conversation		`CREATE INDEX messages_conversation ON messages (conversationId, isStory, storyId, received_at, sent_at)`
conversationId		"conversationId"
isStory		"isStory"
storyId		"storyId"
received_at		"received_at"
sent_at		"sent_at"
messages_conversation_no_story_id		`CREATE INDEX messages_conversation_no_story_id ON messages (conversationId, isStory, received_at, sent_at)`
conversationId		"conversationId"
isStory		"isStory"
received_at		"received_at"
sent_at		"sent_at"
messages_expires_at		`CREATE INDEX messages_expires_at ON messages ( expiresAt )`
expiresAt		"expiresAt"
messages_hasAttachments		`CREATE INDEX messages_hasAttachments ON messages (conversationId, hasAttachments, received_at) WHERE type IS NOT 'story' AND storyId IS NULL`
conversationId		"conversationId"
hasAttachments		"hasAttachments"
received_at		"received_at"
messages_hasFileAttachments		`CREATE INDEX messages_hasFileAttachments ON messages (conversationId, hasFileAttachments, received_at) WHERE type IS NOT 'story' AND storyId IS NULL`

Name	Type	Schema
conversationId		"conversationId"
hasFileAttachments		"hasFileAttachments"
received_at		"received_at"
messages_hasVisualMediaAttachments		`CREATE INDEX messages_hasVisualMediaAttachments ON messages ( conversationId, isStory, storyId, hasVisualMediaAttachments, received_at, sent_at ) WHERE hasVisualMediaAttachments IS 1`
conversationId		"conversationId"
isStory		"isStory"
storyId		"storyId"
hasVisualMediaAttachments		"hasVisualMediaAttachments"
received_at		"received_at"
sent_at		"sent_at"
messages_id		`CREATE INDEX messages_id ON messages (id ASC)`
id	ASC	"id" ASC
messages_isStory		`CREATE INDEX messages_isStory ON messages(received_at, sent_at) WHERE isStory = 1`
received_at		"received_at"
sent_at		"sent_at"
messages_preview		`CREATE INDEX messages_preview ON messages (conversationId, shouldAffectPreview, isGroupLeaveEventFromOther, received_at, sent_at)`
conversationId		"conversationId"
shouldAffectPreview		"shouldAffectPreview"
isGroupLeaveEventFromOther		"isGroupLeaveEventFromOther"
received_at		"received_at"
sent_at		"sent_at"
messages_preview_without_story		`CREATE INDEX messages_preview_without_story ON messages (conversationId, shouldAffectPreview, isGroupLeaveEventFromOther, received_at, sent_at) WHERE storyId IS NULL`
conversationId		"conversationId"
shouldAffectPreview		"shouldAffectPreview"
isGroupLeaveEventFromOther		"isGroupLeaveEventFromOther"
received_at		"received_at"
sent_at		"sent_at"
messages_receipt		`CREATE INDEX messages_receipt ON messages (sent_at)`
sent_at		"sent_at"
messages_schemaVersion		`CREATE INDEX messages_schemaVersion ON messages (schemaVersion)`
schemaVersion		"schemaVersion"
messages_searchOrder		`CREATE INDEX messages_searchOrder on messages(received_at, sent_at)`
received_at		"received_at"
sent_at		"sent_at"
messages_sourceServiceId		`CREATE INDEX messages_sourceServiceId on messages(sourceServiceId)`
sourceServiceId		"sourceServiceId"
messages_story_replies		`CREATE INDEX messages_story_replies ON messages (storyId, received_at, sent_at) WHERE isStory IS 0`
storyId		"storyId"
received_at		"received_at"
sent_at		"sent_at"

Name	Type	Schema
messages_unexpectedly_missing_expiration_start_timestamp		`CREATE INDEX messages_unexpectedly_missing_expiration_start_timestamp ON messages ( expireTimer, expirationStartTimestamp, type ) WHERE expireTimer IS NOT NULL AND expirationStartTimestamp IS NULL`
expireTimer		"expireTimer"
expirationStartTimestamp		"expirationStartTimestamp"
type		"type"
messages_unread		`CREATE INDEX messages_unread ON messages (conversationId, readStatus, isStory, storyId, received_at, sent_at) WHERE readStatus IS NOT NULL`
conversationId		"conversationId"
readStatus		"readStatus"
isStory		"isStory"
storyId		"storyId"
received_at		"received_at"
sent_at		"sent_at"
messages_unread_mentions		`CREATE INDEX messages_unread_mentions ON messages (conversationId, readStatus, mentionsMe, isStory, storyId, received_at, sent_at) WHERE readStatus IS NOT NULL`
conversationId		"conversationId"
readStatus		"readStatus"
mentionsMe		"mentionsMe"
isStory		"isStory"
storyId		"storyId"
received_at		"received_at"
sent_at		"sent_at"
messages_unread_mentions_no_story_id		`CREATE INDEX messages_unread_mentions_no_story_id ON messages (conversationId, readStatus, mentionsMe, isStory, received_at, sent_at) WHERE isStory IS 0 AND readStatus IS NOT NULL`
conversationId		"conversationId"
readStatus		"readStatus"
mentionsMe		"mentionsMe"
isStory		"isStory"
received_at		"received_at"
sent_at		"sent_at"
messages_unread_no_story_id		`CREATE INDEX messages_unread_no_story_id ON messages (conversationId, readStatus, isStory, received_at, sent_at) WHERE readStatus IS NOT NULL`
conversationId		"conversationId"
readStatus		"readStatus"
isStory		"isStory"
received_at		"received_at"
sent_at		"sent_at"
messages_unseen_no_story		`CREATE INDEX messages_unseen_no_story ON messages (conversationId, seenStatus, isStory, received_at, sent_at) WHERE seenStatus IS NOT NULL`
conversationId		"conversationId"
seenStatus		"seenStatus"
isStory		"isStory"
received_at		"received_at"
sent_at		"sent_at"
messages_unseen_with_story		`CREATE INDEX messages_unseen_with_story ON messages (conversationId, seenStatus, isStory, storyId, received_at, sent_at) WHERE seenStatus IS NOT NULL`

Name	Type	Schema
conversationId		"conversationId"
seenStatus		"seenStatus"
isStory		"isStory"
storyId		"storyId"
received_at		"received_at"
sent_at		"sent_at"
messages_view_once		`CREATE INDEX messages_view_once ON messages (isErased) WHERE isViewOnce = 1`
isErased		"isErased"
preKeys_ourServiceId		`CREATE INDEX preKeys_ourServiceId ON preKeys (ourServiceId)`
ourServiceId		"ourServiceId"
reaction_identifier		`CREATE INDEX reaction_identifier ON reactions ( emoji, targetAuthorAci, targetTimestamp )`
emoji		"emoji"
targetAuthorAci		"targetAuthorAci"
targetTimestamp		"targetTimestamp"
reactions_byTimestamp		`CREATE INDEX reactions_byTimestamp ON reactions (fromId, timestamp)`
fromId		"fromId"
timestamp		"timestamp"
reactions_messageId		`CREATE INDEX reactions_messageId ON reactions (messageId)`
messageId		"messageId"
reactions_unread		`CREATE INDEX reactions_unread ON reactions ( conversationId, unread )`
conversationId		"conversationId"
unread		"unread"
recentGifs_order		`CREATE INDEX recentGifs_order ON recentGifs ( lastUsedAt DESC )`
lastUsedAt	DESC	"lastUsedAt" DESC
sendLogMessageIdsByMessage		`CREATE INDEX sendLogMessageIdsByMessage ON sendLogMessageIds (messageId)`
messageId		"messageId"
sendLogPayloadsByTimestamp		`CREATE INDEX sendLogPayloadsByTimestamp ON sendLogPayloads (timestamp)`
timestamp		"timestamp"
sendLogRecipientsByRecipient		`CREATE INDEX sendLogRecipientsByRecipient ON sendLogRecipients (recipientServiceId, deviceId)`
recipientServiceId		"recipientServiceId"
deviceId		"deviceId"
signedPreKeys_ourServiceId		`CREATE INDEX signedPreKeys_ourServiceId ON signedPreKeys (ourServiceId)`
ourServiceId		"ourServiceId"
sticker_packs_by_position_and_id		`CREATE INDEX sticker_packs_by_position_and_id ON sticker_packs ( position ASC, id ASC )`
position	ASC	"position" ASC
id	ASC	"id" ASC
stickers_recents		`CREATE INDEX stickers_recents ON stickers ( lastUsed ) WHERE lastUsed IS NOT NULL`
lastUsed		"lastUsed"
		`CREATE INDEX storyReads_data ON storyReads ( storyReadDate,`

Name	Type	Schema
storyReads_data		authorId, conversationId)
storyReadDate		"storyReadDate"
authorId		"authorId"
conversationId		"conversationId"
storyReads_storyId		CREATE INDEX storyReads_storyId ON storyReads (storyId)
storyId		"storyId"
syncTasks_delete		CREATE INDEX syncTasks_delete ON syncTasks (attempts DESC)
attempts	DESC	"attempts" DESC
syncTasks_type		CREATE INDEX syncTasks_type ON syncTasks (type)
type		"type"
unprocessed_byReceivedAtCounter		CREATE INDEX unprocessed_byReceivedAtCounter ON unprocessed (receivedAtCounter)
receivedAtCounter		"receivedAtCounter"
unprocessed_byReceivedAtDate		CREATE INDEX unprocessed_byReceivedAtDate ON unprocessed (receivedAtDate)
receivedAtDate		"receivedAtDate"
unresolved_sticker_refs		CREATE INDEX unresolved_sticker_refs ON sticker_references (packId, stickerId) WHERE isUnresolved IS 1
packId		"packId"
stickerId		"stickerId"

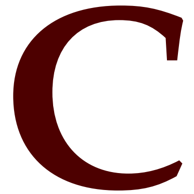
Views (0)

Name	Type	Schema

Triggers (6)

Name	Type	Schema
messages_on_delete		CREATE TRIGGER messages_on_delete AFTER DELETE ON messages BEGIN DELETE FROM messages_fts WHERE rowid = old.rowid; DELETE FROM sendLogPayloads WHERE id IN (SELECT payloadId FROM sendLogMessageIds WHERE messageId = old.id); DELETE FROM reactions WHERE rowid IN (SELECT rowid FROM reactions WHERE messageId = old.id); DELETE FROM storyReads WHERE storyId = old.storyId; END
messages_on_insert		CREATE TRIGGER messages_on_insert AFTER INSERT ON messages WHEN new.isViewOnce IS NOT 1 AND new.storyId IS NULL BEGIN INSERT INTO messages_fts (rowid, body) VALUES (new.rowid, new.body); END
messages_on_insert_insert_mentions		CREATE TRIGGER messages_on_insert_insert_mentions AFTER INSERT ON messages BEGIN INSERT INTO mentions (messageId, mentionAci, start, length) SELECT messages.id, bodyRanges.value ->> 'mentionAci' as mentionAci, bodyRanges.value ->> 'start' as start, bodyRanges.value ->> 'length' as length FROM messages, json_each(messages.json ->> 'bodyRanges') as bodyRanges WHERE bodyRanges.value ->> 'mentionAci' IS NOT NULL AND messages.id = new.id; END
messages_on_update		CREATE TRIGGER messages_on_update AFTER UPDATE ON messages WHEN (new.body IS NULL OR old.body IS NOT new.body) AND new.isViewOnce IS NOT 1 AND new.storyId IS NULL BEGIN DELETE FROM messages_fts WHERE rowid = old.rowid; INSERT INTO messages_fts (rowid, body) VALUES (new.rowid, new.body); END
messages_on_update_update_mentions		CREATE TRIGGER messages_on_update_update_mentions AFTER UPDATE ON messages BEGIN DELETE FROM mentions WHERE messageId = new.id; INSERT INTO mentions (messageId, mentionAci, start, length) SELECT messages.id, bodyRanges.value ->> 'mentionAci' as mentionAci, bodyRanges.value ->> 'start' as start, bodyRanges.value ->> 'length' as length FROM messages, json_each(messages.json -

Name	Type	Schema
		>> 'bodyRanges') as bodyRanges WHERE bodyRanges.value ->> 'mentionAci' IS NOT NULL AND messages.id = new.id; END
messages_on_view_once_update		CREATE TRIGGER messages_on_view_once_update AFTER UPDATE ON messages WHEN new.body IS NOT NULL AND new.isViewOnce = 1 BEGIN DELETE FROM messages_fts WHERE rowid = old.rowid; END



Possible Group Changes In Signal Desktop

Table C.1: *List of possible group changes.*

Type	Description	Details	Example
`create`	Group Created	—	`"type": "create"`
`title`	Group Name Changed	• **newTitle**: A string containing the new group name.	`"type": "title",` `"newTitle": "A new group name"`

`description`	Group Description Changed	• **description**: A string containing the new group description.
		• **removed**: A boolean flag indicating whether the group's description was removed.
			`"type": "description",`
			`"description": "A new group description",`
			`"removed": false`

`avatar`	Group Avatar Changed	• **removed**: A boolean flag indicating whether the group's avatar was removed.	`"type": "avatar",` `"removed": true`
`group-link-add`	Group Link Enabled	• **privilege**: An integer denoting if admin approval is required for members joining through the group link. 1 if yes, 3 if not.	`"type": "group-link-add",` `"privilege": 1`
`group-link-remove`	Group Link Disabled	—	`"type": "group-link-remove"`
`group-link-reset`	Group Link Reset	—	`"type": "group-link-reset"`

Continued on the next page.

Type	Description	Details	Example
`access-invite-link`	Group Link Admin Approval Toggled	• **newPrivilege**: An integer denoting if admin approval was enabled or disabled for members joining through the group link. 1 if admin approval was disabled, 3 if admin approval was enabled.	`"type": "access-invite-link",` `"newPrivilege": 3`
`access-members`	Add Members Permission Changed	• **newPrivilege**: An integer denoting who can add members to the group. 2 if everyone, 3 if only group administrators.	`"type": "access-members",` `"newPrivilege": 2`
`access-attributes`	Modify Group Profile Permission Changed	• **newPrivilege**: An integer denoting who can modify the group's profile. 2 if everyone, 3 if only group administrators.	`"type": "access-attributes",` `"newPrivilege": 2`
`announcements-only`	Announcements Only Mode Toggled	• **announcementsOnly**: A boolean flag indicating if the announcement only mode was enabled or disabled.	`"type": "announcements-only",` `"announcementsOnly": true`
`member-add`	Member Added	• **aci**: A string containing the ACI of the added user.	`"type": "member-add",` `"aci": "82d105f9-0b12-4557-8acb-ab7b1cb98a5f"`
`member-remove`	Member Removed	• **aci**: A string containing the ACI of the removed user.	`"type": "member-remove",` `"aci": "82d105f9-0b12-4557-8acb-ab7b1cb98a5f"`

`member-privilege`	Member Role Updated	• **aci**: A string containing the ACI of the target user.
		• **newPrivilege**: An integer denoting the target user's new role. 2 if Administrator, 1 if a regular member.
			`"type": "member-privilege",`
			`"aci": "82d105f9-0b12-4557-8acb-ab7b1cb98a5f",`
			`"newPrivilege": 2`

This page intentionally left blank.