Forensic Science International: Digital Investigation 52 (2025) 301866

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS EU 2025 - Selected Papers from the 12th Annual Digital Forensics Research Conference Europe

Forensic analysis of Telegram Messenger on iOS smartphones

Lukas Jaeckel *, Michael Spranger, Dirk Labudde

University of Applied Sciences Mittweida, Technikumplatz 17, 09648 Mittweida, Germany

ARTICLE INFO

Keywords:
Mobile forensics
Telegram
Telegram messenger
Instant messaging
iOS
Smartphones
SQLite database

ABSTRACT

As mobile messengers have dominated and penetrated our daily communication and activities, the odds of them being involved in criminal activities have increased. Since each messenger usually uses its own proprietary data schema (including encoding, encryption and frequent updates) to store communication data, with a pressing demand, investigative authorities require a solution to transfer the data in a processable structure to analyse it efficiently, especially in a forensic context. Therefore, this work identifies and examines locally stored data of the Telegram Messenger with high forensic value on iOS devices. In particular, this work deals with extracting contact and communication data to link and analyse it. For this purpose, artificially generated test data, as well as the open source code of the Telegram Messenger under iOS, are analysed. The main focus of this work lies on the primary database in which a large part of data is coded and, therefore, needs to be transferred into an interpretable form. In summary, this work enables a manual or automated analysis of Messenger data for investigative authorities and IT companies with forensic reference. The proposed method can also be adapted in research to analyse further instant messaging services.

1. Introduction

With more than 900 million active users per month, Telegram is one of the most popular messenger services worldwide (We Are Social et al., 2024). Due to its security features, such as secret chat with end-to-end encryption, the service is also popular among criminals (Moreb, 2022a). As a result, the messenger is used to plan, control and commit offences ranging from cybercrime to terrorism (Anglano et al., 2017). Therefore, analysing locally stored messenger data can help to clarify criminally relevant issues if a device has been confiscated and seized by a corresponding intelligence service. The analysis provides information on who communicated with whom at what time and on what topic. From this information, for example, it is possible to derive clues as to the motive for the crime, the form of the perpetration or crimes planned for the future. However, before a more in-depth data analysis can be carried out, law enforcement authorities face the challenge of preparing the secured data accordingly (Moreb, 2022b). That is because the locally stored messenger data is in an encoded form. Consequently, the data has to be decoded and converted into an analyseable format. In many cases, law enforcement agencies overcome this challenge by using special programs in which a parser is integrated for each messenger. However, the data preparation process within these programs is not transparent and understandable.

Consequently, there is a need for detailed research and documentation of Telegram regarding storage and coding on iOS devices. This work aims to explore as much forensically relevant data as possible from the Telegram messenger on iOS so that it can be decoded and converted into a form that can be analysed. In this way, the work enables a deep analysis of the communications conducted in Telegram. In order to derive further insights, additional contact and communication networks can thus be created and linked. The comprehensive analysis of Telegram Messenger on iOS devices can be carried out manually or automatically using any programming language.

In the following, section 2 provides an overview of similar literature. Then, the unique features of Telegram are explained in section 3, such as the different types of contacts and chats. Subsequently, section 4 shows the methodological approach. The data structures of Telegram on iOS devices are described in section 5. The analysis results obtained are discussed in section 6. Finally, section 7 summarises the work and discusses its applicability in practice.

2. Related work

A systematic review by Sihombing et al. (2018) emphasised the forensic value of IM (Instant Messaging) data. It showed that from 2014 to 2017, only one scientific paper was published in the IM field for iOS

* Corresponding author.
E-mail address: jaeckel1@hs-mittweida.de (L. Jaeckel).

https://doi.org/10.1016/j.fsidi.2025.301866

Available online 24 March 2025

devices. The article dealt with the messenger service Kik (Kenneth and Ovens, 2016). Six years earlier, Husain and Sridhar (Husain et al., 2010) covered three other IM services (AIM, Yahoo! Messenger, Google Talk) on the iOS operating system, all of which have since been discontinued. On the other hand, IM services on Android devices have often been investigated. For example, Mahajan et al. (2013) dealt with the forensic analysis of the messengers Viber and WhatsApp. They generated messenger data on five different Android devices and analysed them manually. As a result, their work showed where and which forensic artefacts of the applications can be found. Anglano (2014) extended the analysis of WhatsApp, where the author used virtualised devices to generate the data. In his work, he examined other forensically relevant files and documented how the locally stored data of the WhatsApp messenger can be linked and interpreted.

Telegram Messenger version 3.4.2 on Android devices has already been examined by Satrya et al. (2016). For this purpose, the authors carried out 17 different test scenarios using three smartphones, which were subsequently analysed using online and offline forensics methods. In particular, the paper showed where and in which format forensically relevant data of the Telegram Messenger are created and stored. However, the authors did not describe how the data could be interpreted and correlated. For example, the evaluation of the Binary Large Objects (BLOBs) within the central database cache4.db was missing, although these objects contain valuable information in coded form. Anglano et al. (2017) further analysed Telegram Messenger versions 3.15 to 3.18 on Android devices regarding offline forensics. For this, they used three virtualised Android devices and a real smartphone in their experiments to validate the results obtained. In addition to examining the resulting data, the authors analysed the public source code of the Android version of Telegram Messenger to interpret data from the central database better. In addition, they used parts of the code written in Java to deserialise BLOBs. In doing so, they did not have to go into the exact coding of such objects.

In conclusion, the authors made clear that the methods used could also be extended to Telegram versions of other platforms, such as iOS and Windows Phone. However, they recognised that the difficulty with iOS devices is that no virtualisation platform exists. Therefore, physical devices were used to generate Messenger data in this work.

Furthermore, Gregorio et al. (2017) researched the Telegram Messenger for Windows Phone. For this purpose, the authors combined public knowledge, analysing the messenger's generated data and the freely accessible source code of Telegram for Windows Phone. They used two physical smartphones with Telegram versions 1.12.1 and 1.27 to generate the data. In summary, the paper demonstrated where and how locally stored Telegram Messenger information can be obtained. In addition, Gregorio et al. (2018) examined the desktop version of Telegram Messenger on MacOS. In particular, the authors described methods for finding relevant data and deriving their meanings concerning users' communication.

Bhatt et al. (2018) analysed the data from the network traffic of 20 different iOS applications. Among them was the iOS version of Telegram Messenger, whose data is, in principle, transmitted in encrypted form. Nevertheless, the authors succeeded in recording location data, third-party domains, device details of the communication participants, and encrypted text from the messenger's network traffic using online forensics methods. However, Telegram Messenger was not investigated in the area of offline forensics.

Salamh et al. (2021) examined a large number of apps on Android and iOS from a forensic perspective. Telegram on iOS was also included. The authors generated data to locate and document using Autopsy and Magnet AXIOM. That made it possible to determine where Telegram stores its database, media and other files on iOS. However, this did not involve a deep analysis of the data, especially the central database. In their literature review, the authors noted a lack of research regarding Telegram Messenger on iOS devices. This work thus closes a gap in the existing literature in iOS forensics.

Furthermore, Moreb (2022a) dealt with Telegram Messenger on iOS devices. For the data acquisition, the authors used Belkasoft and FINALMobile. Then, they analysed the data using FINALMobile, Magnet AXIOM, and Elcomsoft Phone Viewer. All the generated data could be retrieved. However, the generated files by Telegram were not analysed in detail. In contrast, an analysis of the Telegram Messenger independent of tools is carried out in the context of this work.

3. Communication structures in telegram

Telegram is a free instant messaging service that allows users to send and receive unlimited text messages and media files. A user can communicate with another person registered with Telegram if their contact information is stored in the mobile device's phone book or their public user name is known. A particular form of a user is a programmable bot, which performs specific tasks automatically. A bot can usually see a user's public name and profile picture and exchange messages. The usual communication between two users occurs via *Cloud Chats*, which use client-server/server-client encryption (Telegram, 2024b). Such chats are encrypted on the end devices and in the Telegram cloud. In addition to the standard chat function, Telegram offers voice and video calls and secret chats. Secret chats in Telegram are the only type of communication that uses end-to-end encryption. Therefore, this communication is only stored locally on the sender's and recipient's end devices. Secret chats are also deleted from a device when the user logs out of Telegram. As a result, their existence is always tied to the current session. In addition, such chats offer a self-destruct mode, whereby each message is deleted after a specified time once the recipient has read it.

Up to 200,000 members can communicate in a group, whereby the creator and administrators have special rights (Telegram, 2024b). The chat messages are stored on the group members' end devices and in the Telegram cloud. Members who see each other within a group can also start a regular or secret chat in pairs. A group is either public or private, depending on the setting of the creator. Public groups have a unique group name and, like public users, can be found via the search function in Telegram. Alternatively, a user can join a public group via an invitation link. Private groups, on the other hand, do not have a public group name and cannot be found via the search function. Joining can be done via an invitation link.

Furthermore, members can add new users depending on their authorisation, irrespective of the group's visibility. In principle, it is also possible to invite bots, who can only see messages intended for them by default. However, a bot can be configured to read all messages of a group.

Unlike groups, the number of channel members is unlimited (Telegram, 2024b). However, only the creator of the channel and the administrators have writing access, while all other members have only read access to the messages. Chat messages are saved on the end devices of the channel members as well as in the Telegram cloud and, if necessary, automatically forwarded to a defined group. Furthermore, users without special rights cannot view the member list of a channel. Thus, the creator and the administrators are anonymous to ordinary members. By default, the name of the sender of a message is not displayed. Analogous to groups, channels can be either public or private and are identified by a unique channel name. The invitation links for channels work in the same way as links for groups. However, regular members in private channels cannot invite other users.

4. Methodology

This work followed the workflow described by Anglano et al. (2017) for analysing Telegram on Android devices. However, physical iOS devices were used in this work. To recognise and subsequently analyse forensically relevant data of Telegram Messenger, the official open source code of Telegram under iOS was examined in particular. The code is freely available and constantly updated at https://github.com/Tel

egramMessenger/Telegram-iOS (Telegram, 2024a). The main aim of the source code analysis was to draw conclusions about the structure of the primary database and, thus, to identify relevant tables. Subsequently, these tables were examined and documented concerning their structure to extract information with forensic value from them in the future. Of particular interest was whether each table had specific signatures and a fixed or dynamic structure. To understand the coding of specific classes, first, the corresponding source code file was searched and then the functions init() and encode() were considered.

In addition, test data was generated according to specific requirements. Thus, the code itself could be better understood, and the insights gained from it could be validated. The test data was generated using Telegram Messenger version 11.1.1 on an iPhone SE, iPhone 7 and a Samsung Galaxy S9. The generation process implemented the following steps:

Preparation and installation. First, SIM cards were obtained for each device, and the contact details of the devices were entered into each other's phone books. Then, Telegram Messenger had to be installed on each device.

Creating user profiles. A new Telegram account was created on each device. After that, the profiles of the accounts were each given a profile picture, a user name, and a bio.

Creating groups, channels and secret chats. Groups and channels were created in which the users had different roles (creator, administrator, member). In addition, such chats differed regarding their visibility (private, public). Some groups and channels also had a profile picture and info. Some channels are also linked to a discussion group. Furthermore, secret chats were created for two users at a time. Some of these chats had the self-destruct mode enabled.

Communication in chats of different types. In ordinary chats (cloud chats), groups, channels and secret chats, users now exchanged messages, whereby the communication was as far as possible, not one-sided. Some messages were longer than 255 characters and had emojis and special characters. They also included surveys and media. The latter included pictures, audio files, videos, documents, stickers, locations and contacts. In some cases, users replied directly to messages or forwarded them. In addition to messages, users made, declined or cancelled voice and video calls.

After the test data generation according to the requirements was completed, UFED Touch (Cellebrite DI Ltd, 2024) was used for the full file system extraction. The folder in which the Telegram Messenger application data was stored was backed up from each device. That included the folder telegram-data, which contained the local central database for the test account created. The exported database was copied to a separate folder in each case, as no files such as db_sqlite-wal were to be changed during further analysis. That was followed by investigating the extracted primary databases to validate the findings gained from the code analysis. The primary databases were examined using the DB Browser for SQLite (Clift, 2024) for a general overview. Thus, each database could be opened, and the content of the individual tables could be analysed. The HxD Hex Editor (Hörz, 2024) was used for a deeper analysis of the respective table content. It facilitated the finding of offsets and signatures on a binary level. It also automatically converted bytes into specific number formats when necessary.

5. Telegram data structures on iOS devices

In October 2018, Telegram X replaced the C-based iOS variant of Telegram Messenger with the release of version 5.0 while retaining the original name *Telegram Messenger* (Telegram, 2018). Telegram X for iOS is an optimised variant of the old C-based messenger rewritten in Swift. An optimised *Telegram X* variant also exists for Android but is currently being developed and offered in parallel to the conventional variant.

Due to the fundamental change with version 5.0 on iOS, the structure of the databases has completely changed compared to old Telegram versions. Furthermore, the directories have been restructured. Most Telegram files are stored in the directory telegram-data, which is why it has a high forensic relevance. It is located under the path \private\var\mobile\Containers\Shared\AppGroup*x*, where *x* stands for an app-specific ID that is not constant. General information is kept in the subdirectory accounts-metadata. Each user account used to log on to the device has a separate directory in telegram-data containing important user-specific data. Within it is a directory postbox, which contains a media directory media and a database directory db. The latter contains the central database db_sqlite (Fig. 1), which stores messages and contacts of the user account. For this reason, this work focuses on the primary database analysis.

In addition, the db_sqlite-wal file in the database directory may contain important information that has been deleted from or is not yet stored in the central database. This file is created by Write-Ahead Logging (WAL), an optional mode of an SQLite database (SQLite Documentation, 2024). Database transactions are not written directly to the central database but are stored temporarily in a WAL file. As a result, the two files can each have a different status, which can be of significant forensic relevance.

6. Forensic analysis of the telegram main database

The local main database of a specific account mainly contains several tables whose names are each composed of the prefix t and an identification number from the range of natural numbers. Each table with the prefix t has, in principle, two columns. The content of the first column

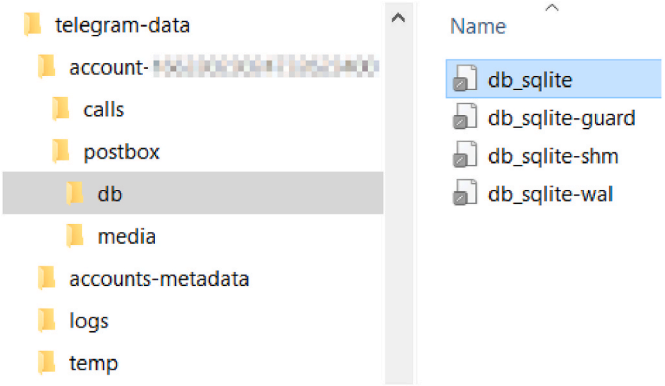


Fig. 1. Telegram directory structure. On the left is the directory hierarchy starting with the directory telegram-data, with the currently open database directory highlighted in grey. The right-hand side shows the contents of this directory. db_sqlite is the local central database of the Telegram account.

key is either of the type Integer or Binary Large Object (BLOB). The second column is called value and always contains a BLOB, which is a long sequence of bytes [(Fehily, 2020), p. 60]. As a result, most data is encoded in a serialised binary format. They must, therefore, be decoded before the actual examination can be carried out. None of the analysed data was encrypted.

Furthermore, a separate.swift file exists for each table in the Telegram source project under submodules/Postbox/Sources/. After analysing the corresponding source files, the most relevant information about contacts and chats was found in the PeerTable t2. This information complements entries from the CachedPeerDataTable t18. In addition, the MessageHistoryTable t7 contains forensically valuable information about messages, calls and media files. Using the init() function of the Postbox class in the Postbox.swift file under submodules/Postbox/Sources/, the respective identification number of the tables could be inferred.

6.1. Coding of objects

To evaluate the respective table contents, it is necessary in many cases to know how individual objects and their attributes are coded in a BLOB. Hierarchical structures are formed at the top level, of which there are usually one or more RootObjects. A RootObject is coded as follows:

```
0x015F05 + 4 byte hash of object class name +
4 byte attribute length n + n byte attributes
```

An object encoded this way usually contains several class-specific attributes that follow one another in a well-defined order. Each attribute is encoded according to this scheme:

```
1 byte key lenght n + n byte key + 1 byte
data type + coded attribute value
```

Here, the key indicates the kind of class-specific attribute. Furthermore, the coding of the attribute value depends directly on the data type of the corresponding attribute. The coding can be derived from the class PostboxEncoder in the source file Coding.swift under submodules/Postbox/Sources/and is summarised in Table 11. The class name of an object is always stored as a hash value. In detail, this is a 32-bit value generated from the class name using the MurmurHash3 with the seed value −137723950. Since the hash values are unique for each class, they each represent a 4-byte signature for an object of a specific class. The exact hash algorithm can be found in the MurMurHash32.m file of the Telegram source code at submodules/MurMurHash32/Sources/.

6.2. Extraction of the peer ID of the local user account

In order to extract information about the local user account, the matching peer ID must first be determined. This number, which is unique in Telegram, is stored in the MetadataTable t0, which is defined in the Telegram source file MetadataTable.swift under submodules/Postbox/Sources/. The corresponding table contains up to six line entries. The peer ID is in the State entry, where the key is 2 and the value contains a RootObject of the class AuthorizedAccountState if the local user account is authorised. The structure of such an object can be derived from the file SyncCore_AuthorizedAccountState.swift available at submodules/TelegramCore/Sources/SyncCore/. In order to determine the peer ID, it is usually sufficient to search for the byte sequence 0 × 06 70 65 65 72 49 64 01, which includes the key length (0 × 06), the key of the attribute (0 × 70 65 65 72 49 64) and its data type (0 × 01). That is followed by the eight-byte peer ID of the local user account in little-endian format. To illustrate, Fig. 2 shows an example State entry from the value column of the MetadataTable. In this case, the peer ID of the local user is decimal 36513321142 or 0 × 08 80 5D 10 B6 in hexadecimal (little-endian format).

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   01 5F 05 06 03 71 5F 8F 00 00 00 14 69 73 54 65  ._...q_.....isTe
00000010   73 74 69 6E 67 45 6E 76 69 72 6F 6E 6D 65 6E 74  stingEnvironment
00000020   00 00 00 00 00 12 6D 61 73 74 65 72 44 61 74 61  ......masterData
00000030   63 65 6E 74 65 72 49 64 00 02 00 00 00 06 70 65  centerId......pe
00000040   65 72 49 64 01 B6 10 5D 80 08 00 00 00 05 73 74  erId.¶.]€.....st
00000050   61 74 65 05 7C B1 A3 D8 25 00 00 00 03 70 74 73  ate.|±£Ø%....pts
00000060   00 25 00 00 00 03 71 74 73 00 12 69 51 3C 04 64  .%....qts..iQ<.d
00000070   61 74 65 00 DA EF FF 66 03 73 65 71 00 06 02 00  ate.Úïÿf.seq....
00000080   00 13 69 6E 76 61 6C 69 64 61 74 65 64 43 68 61  ..invalidatedCha
00000090   6E 6E 65 6C 73 08 00 00 00 00                    nnels.....
```

Fig. 2. Shown is the content of a State entry of the column value, which comes from the table t0 of the local main database of a Telegram account. Here, the entry is encoded as a BLOB. The uniform signature (solid line) is followed by a unique peer ID of the local user (dashed line).

6.3. Extraction of contact and chat data

The PeerTable, referred to as t2 in the main database, stores important information about users, groups, channels and secret chats. The table's structure can be derived from the PeerTable.swift file of the Telegram source code at submodules/Postbox/Sources/. For each peer instance, there is exactly one entry in the table. The key column contains the peer ID of the respective instance. In the column value of the PeerTable, the object belonging to a peer instance is stored as RootObject in little-endian format. As already discussed, the encoding as a BLOB depends on the exact class of the object and the respective attributes.

The CachedPeerDataTable t18 of the main database contains additional information about individual peer instances and is defined in the CachedPeerDataTable.swift file under submodules/Postbox/Sources/. However, an entry does not have to exist for each instance. Analogous to the PeerTable, the key column of t18 contains the peer ID of the corresponding peer instance. In the column value, a RootObject encoded as a BLOB is stored in little-endian format. Table 1 summarises the possible RootObject classes.

6.3.1. Extraction of users and bots

The RootObject in value entries of table t2 is for ordinary users and bots from the class TelegramUser, which is defined in the SyncCore_TelegramUser.swift file under submodules/TelegramCore/Sources/SyncCore/. The 32-bit MurmurHash3 value for the corresponding class name is 2657658155. Converted as a hexadecimal number in little-endian format, this results in the specific signature 0 × 2B A5 68 9E. For example, Fig. 3 from the generated test data shows a column entry containing information about a Telegram user. An overview of the attributes of the class TelegramUser is given in Table 2.

The most relevant information that can be extracted from such a class is the first name (key: 0 × 66 6E), last name (key: 0 × 6C 6E), user name (key: 0 × 75 6E) and phone number (key: 0 × 70) of a user. However, not every object of the class TelegramUser contains the mentioned attributes because they depend on the voluntary specification of the user. The user name and phone number can uniquely identify a Telegram account.

Table 1
Possible RootObject classes of the value entries in table t18 with associated MurmurHash3 and the corresponding source file of Telegram iOS from the open source project at submodules/TelegramCore/Sources/SyncCore/.

Class (source file)	Hash value
CachedUserData (SyncCore_CachedUserData.swift)	0 × 91 9D 1F 76
CachedGroupData (SyncCore_CachedGroupData.swift)	0 × 21 E8 2D A9
CachedChannelData (SyncCore_CachedChannelData.swift)	0 × 55 15 C2 16
CachedSecretChatData (SyncCore_TelegramSecretChat.swift)	0 × 4B D0 6F 4E

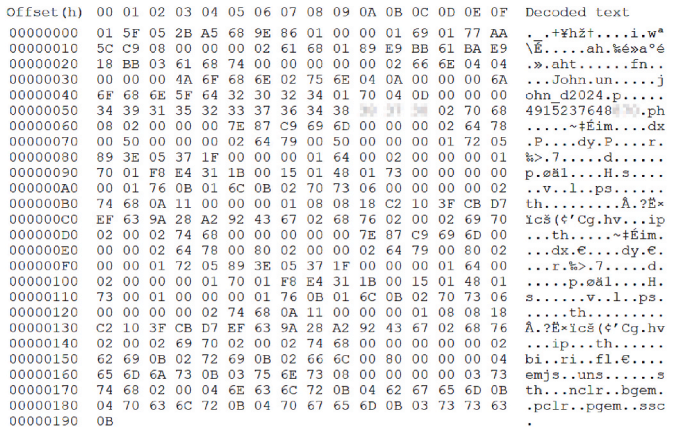


Fig. 3. The content of an entry in the value column from the t2 table of the local main database of a Telegram account is opened in a hex editor because it is coded as a BLOB.

Table 2
Attributes of the class TelegramUser.

Attribute	Key	Data type
id	0 × 69	Int64
accessHash	0 × 61 68	Int64
accessHashType	0 × 61 68 74	Int32
firstName	0 × 66 6E	String
lastName	0 × 6C 6E	String
username	0 × 75 6E	String
phone	0x70	String
photo	0 × 70 68	ObjectArray
botInfo	0 × 62 69	Object
restrictionInfo	0 × 72 69	Object
flags	0 × 66 6C	Int32
emojiStatus	0 × 65 6D 6A 73	Object
usernames	0 × 75 6E 73	ObjectArray
storiesHidden	0 × 73 74 68	boolean
nameColor	0 × 6E 63 6C 72	Int32
backgroundEmojiId	0 × 62 67 65 6D	Int64
profileColor	0 × 70 63 6C 72	Int32
profileBackground-EmojiId	0 × 70 67 65 6D	Int64
subscriberCount	0 × 73 73 63	Int32

Usually, a profile picture of a user is stored in low as well as in full resolution as an object of the class TelegramMediaImageRepresentation in the photo-ObjectArray with the key 0 × 70 68. The specific signature derived from the MurmurHash3 is 0 × 7E 87 C9 69 for objects of the corresponding class. The definition of TelegramMediaImageRepresentation is in the source file SyncCore_TelegramMediaImage.swift at submodules/TelegramCore/Sources/SyncCore/.

Whether a user is a bot can be determined by evaluating the object with the key 0 × 62 69. If the key value is followed by the NULL value 0 × 0B, it is an ordinary user. Otherwise, an object of the structure BotUserInfo with the specific signature 0 × 55 7F AA 56 follows. The source file SyncCore_TelegramUser.swift defines the structure.

Further information can also be extracted for the local account, as a corresponding user entry exists in the PeerTable. The entry has the peer ID, the determination of which has already been discussed. Finally, the findings about users and bots derived from the Telegram source code were validated using the test data. The decoded attributes of the entry shown in Fig. 3 are summarised in Table 3.

Table t18 stores additional information about users and bots, where the RootObject of the value column is of the class CachedUserData. Below this is the about attribute with the key 0 × 61, which refers to the value info given by the user in the form of a string. Additional

Table 3
Decoded attributes of the TelegramUser object from Fig. 3.

Offset	Attribute	Decoded value
0 × 0B to 0 × 15	id	37738031735
0 × 16 to 0 × 21	accessHash	−4964961602463078007
0 × 22 to 0 × 2A	accessHash-Type	0 → personal
0 × 2B to 0 × 36	firstName	John
0 × 37 to 0 × 48	username	john_d2024
0 × 49 to 0 × 5C	phone	4915237648 …
0x5D to 0 × 14E	photo	Array including two objects of the class TelegramMedia-ImageRepresentation
0 × 14F to 0 × 152	botInfo	NIL/NULL → User is not a bot.
0 × 153 to 0 × 156	restriction-Info	NIL/NULL
0 × 157 to 0 × 15E	flags	128 → mutualContact
0 × 15F to 0 × 164	emojiStatus	NIL/NULL
0 × 165 to 0 × 16D	usernames	Empty Array
0 × 16E to 0 × 173	stories-Hidden	false
0 × 174 to 0 × 190	Further attributes	in each case: NIL/NULL

information about bots can be found in the 0 × 62 69 attribute marked by the key botInfo. Provided it is a bot, the value of the attribute is an object of the class BotInfo. Otherwise, its value is NIL/NULL. The BotInfo object stores a description (key: 0 × 64) and an object array of BotCommands (key: 0 × 63) containing a text (key: 0 × 74) and a description (key: 0 × 64). In chats with users and bots, exactly one message can be pinned to keep it in view. The ID of such a message is stored as the attribute's value with the key 0 × 70 6D 2E 69. If no message is pinned, the value is NIL/NULL. Furthermore, the attribute is Blocked identified by the key 0 × 62 shows whether a user/bot is blocked by the local user (0 × 01) or not (0 × 00).

6.3.2. *Extraction of groups*

Telegram groups are encoded in the column value in the table t2 as objects of the class TelegramGroup, which is defined in the SyncCore_TelegramGroup.swift file under submodules/TelegramCore/Sources/SyncCore/. The hexadecimal MurmurHash3 value in little-endian format is 0 × 51 7D A1 66. The specific group attributes and their keys are summarised in Table 4.

In addition to the title of the group (key: 0 × 74), the class TelegramGroup has a ObjectArray (key: 0 × 70 68) in which profile pictures can be located. In addition, the attribute with the key 0 × 70 63 contains the current number of group members. The creation date is stored as Unix time (key: 0 × 64). The Int32 value assigned to the key 0 × 6D can be used to determine the current status of the device owner within the respective group. He can be a current member (0 × 00), have left the group (0 × 01) or have been removed from it (0 × 02). The group role of the local user can be determined by the object with the key 0 × 72 76. In order to do this, the object's attribute with the key 0 × 5F 76 must be

Table 4
Attributes of the class TelegramGroup.

Attribute	Key	Data type
id	0 × 69	Int64
title	0 × 74	String
photo	0 × 70 68	ObjectArray
participantCount	0 × 70 63	Int32
role	0 × 72 76	Object
membership	0 × 6D	Int32
flags	0 × 66	Int32
defaultBannedRights	0 × 64 62 72	Object
migrationReference (id)	0 × 6D 72 2E 69	Int64
migrationReference (accessHash)	0 × 6D 72 2E 61	Int64
creationDate	0 × 64	Int32
version	0 × 76	Int32

evaluated. This object is from the enumeration TelegramGroupRole, which can also be derived from the file SyncCore_TelegramGroup.swift. The corresponding Int32 value indicates whether the user is the creator (0 × 00), administrator (0 × 01) or a normal member (0 × 02) of the group. Furthermore, a group can be represented in an additional entry by a class object TelegramChannel. In this case, it is called a migrated group. That occurs in public groups and discussion groups. If such an object exists, its peer ID is stored in an attribute with the key 0 × 6D 72 2E 69 in the TelegramGroup object.

More information about a group can be found in table t18. The key column contains the peer ID of the group, whereas the additional information in the value column is stored as a BLOB in little-endian format. The corresponding RootObject is of class CachedGroupData. Analogous to the class CachedUserData, objects of the class CachedGroupData have the about attribute (key: 0 × 61 62) as well as the ID of a pinned message (key: 0 × 70 6D 2E 69). Furthermore, the key 0 × 62 is followed by an attribute containing an array of class objects CachedPeerBotInfo. Such an object represents a bot within the group and has its peer ID (key: 0 × 70) and an object of the class BotInfo (key: 0 × 69). The evaluation of BotInfo objects has already been discussed in subsubsection 6.3.1. In addition, the attribute with the key 0 × 69 6E 76 42 79 stores the peer ID of the user who invited the local user to the group. The attribute identified by the key 0 × 70 is of particular relevance, as it stores information about the group members. The attribute's value is an object of the class CachedGroupParticipants, whose MurmurHash3 value is 0 × 5F B5 3B 79 in little-endian format. The source code defines the class under the path submodules/TelegramCore/Sources/SyncCore/ in the file SyncCore_CachedGroupParticipants.swift. It has an array of objects of the enumeration GroupParticipant (key: 0 × 70) and a four-byte version number (key: 0 × 76) as attributes. Each group member is represented by an object of the enumeration GroupParticipant, which is also defined in SyncCore_CachedGroupParticipants.swift. The MurmurHash3 value 0 × DE E4 05 56 is formed from the enumeration name in little-endian format. Group members can thus be quickly found by searching for the special hash value in a value entry of t18. Table 5 gives a chronological overview of all attributes of the enumeration GroupParticipant.

6.3.3. Extraction of channels and migrated groups

Channels and migrated groups are coded as objects of the class TelegramChannel in the table t2 of the local main database. A group is migrated if its visibility is set to public or used as a channel discussion group. In these cases, table t2 usually has an additional entry for the group, where the peer ID in the key column is different from the ID from the regular group entry (see subsubsection 6.3.2). The signature of an class object TelegramChannel is 0 × DA 11 6B 63. Table 6 shows the attributes of the corresponding class, which is defined in the file SyncCore_TelegramChannel.swift under the path submodules/TelegramCore/Sources/SyncCore/.

Just like TelegramGroup objects, TelegramChannel objects have a title (key: 0 × 74), a creation date in Unix time (key: 0 × 64) as well as an ObjectArray for profile pictures (key: 0 × 70 68), the evaluation of which can be done analogously. The attribute marked by the key 0 × 70 73 also indicates the current status of the local user in the channel or group. He can be a current member (0 × 00), have left the channel or group (0 × 01) or have been removed by another user with administrative rights (0 × 02). Whether an object of the class TelegramChannel represents a channel or a group can be recognised by the attribute with the key 0 × 69 2E 74. The value 0 × 00 indicates a channel (broadcast), whereas 0 × 01 stands for a migrated group. If there is no username attribute (key: 0 × 75 6E), it is a private group or a private channel. Otherwise, the visibility is set to public. By evaluating the info flag, which is marked by the key 0 × 69 2E 66, further information can be obtained. From a forensic point of view, it is particularly relevant whether the first bit of the info flag is set for channels. In this case, the sender's name is given for each channel message. Otherwise, they are anonymous. Another flag, marked with the key 0 × 66 6C, provides further information about the user to whom the local account is assigned. In particular, if set, the second bit of the flag shows that the user is the creator of the channel or group.

Additional information about a channel or a migrated group can be stored in table t18. The key column contains the corresponding peer ID, whereas a RootObject of the class CachedChannelData is encoded in the value column in little-endian format. Analogous to CachedGroupData, the class has an about attribute (key: 0 × 61), the ID of a pinned message (key: 0 × 70 6D 2E 69), the attribute botInfos (key: 0 × 62) and the invitedBy attribute (key: 0 × 69 6E 76 42 79). Furthermore, a CachedChannelData object stores general statistical information about all members. This includes the number of all members (key: 0 × 70 2E 6D), administrators (key: 0 × 70 2E 61), banned (key: 0 × 70 2E 62) and removed (key: 0 × 70 2E 6B) users. However, the object does not contain

Table 5
Attributes of the enumeration GroupParticipant. Members and admins have attributes marked with an asterisk (*).

Attribute	Key	Data type
variant	0 × 76	Int32 (0 → Member; 1 → Creator; 2 → Admin)
id	0 × 69	Int64 (Peer ID)
invitedBy*	0 × 62	Int64 (Peer ID)
invitedAt*	0 × 74	Int32 (Unix timestamp)

Table 6
Attributes of the class TelegramChannel.

Attribute	Key	Data type
id	0 × 69	Int64
accessHash	0 × 61 68	Int64
accessHashType	0 × 61 68 74	Int32
title	0 × 74	String
username	0 × 75 6E	String
photo	0 × 70 68	ObjectArray
creationDate	0 × 64	Int32
version	0 × 76	Int32
participationStatus	0 × 70 73	Int32
info (type)	0 × 69 2E 74	Int32
info (flag)	0 × 69 2E 66	Int32
flags	0 × 66 6C	Int32
restrictionInfo	0 × 72 69	Object
adminRights	0 × 61 72	Object
bannedRights	0 × 62 72	Object
defaultBannedRights	0 × 64 62 72	Object
usernames	0 × 75 6E 73	ObjectArray
storiesHidden	0 × 73 74 68	boolean
nameColor	0 × 6E 63 6C 72	Int32
backgroundEmojiId	0 × 62 67 65 6D	Int64
profileColor	0 × 70 63 6C 72	Int32
profileBackground-EmojiId	0 × 70 67 65 6D	Int64
emojiStatus	0 × 65 6D 6A 73	Object
approximateBoost-Level	0 × 61 62 6C	Int32
subscriptionUntil-Date	0 × 73 75 64	Int32

Table 7
Attributes of the class TelegramSecretChat.

Attribute	Key	Data type
id	0 × 69	Int64
regularPeerId	0 × 72	Int64
accessHash	0 × 68	Int64
creationDate	0 × 64	Int32
role	0 × 6F	Int32
embeddedState	0 × 73	Int32
messageAutoremoveTimeout	0 × 61 74	Int32

specific information about individual members. Provided it is a representation of a migrated group, the peer ID of the original group and thus the key value of the ordinary group entry from table t2 can be extracted from the ChannelMigrationReference object, which is stored as the value of the attribute with key 0x6D 72. The peer ID of the group corresponds to the Int64 value of the attribute of the structure ChannelMigrationReference identified by the key 0 × 70. If a channel has a discussion group, the attribute's value with the key 0 × 64 67 69 is the peer ID of the migrated group.

6.3.4. Extraction of secret chats

SyncCore_TelegramSecretChat.swift defines the TelegramSecretChat class. The 32-bit hexadecimal value 0 × 5A 6E C9 21 is formed as the signature from the MurmurHash3 of the class name. An overview of the specific class attributes is given in Table 7.

Since the peer ID of the other chat participant is stored in each entry of such a chat (key: 0 × 72), each secret chat can be directly assigned to the respective users. The value of the attribute with the key 0 × 6F additionally indicates whether the local user is the creator (0 × 00) or participant (0 × 01) of the secret chat. Furthermore, the embeddedState attribute (key:0 × 73) shows the current status of the chat, which can be finished (0 × 00), under construction (0 × 01) or active (0 × 02). In addition, the attribute with the key 0 × 64 stores the creation date as Unix time. The time after which a message is automatically deleted after the recipient has read it can be changed at any time by any chat participant. The currently set time is stored in seconds as the attribute's value with the key 0 × 61 74. An assigned NIL/NULL value indicates that the *self-destruct mode* is disabled for the corresponding secret chat.

In table t18, entries for Secret Chats are less informative than for the other peer instances. For Secret Chats, the key column in t18 contains the corresponding peer ID, while the value entry stores a BLOB-encoded RootObject of the class CachedSecretChatData in little-endian format. An object of this class has only one attribute identified by the key 0 × 70 73 73 named peerStatusSettings whose value is either NIL/NULL or a PeerStatusSettings object. That is defined in SyncCore_PeerStatusSettings.swift. No forensically relevant information was identified in these objects.

6.4. Extraction of communication data

Messages and calls, as well as related metadata, are stored in the MessageHistoryTable, which can be found in the db_sqlite under the table name t7. The table structure can be derived from the file MessageHistoryTable.swift under submodules/Postbox/Sources/.

The entries within the key column are each encoded as a Binary Large Object in big-endian format. However, this binary object contains no RootObjects or attributes but four consecutive integer values whose offsets and meaning are listed in Table 8. The structure can be derived from the key() function of the source file already mentioned.

Based on the peer ID at offset 0 × 00 to 0 × 07, it is possible to specifically identify from which individual chat (cloud chat), secret chat, channel or group a message originates by searching for the matching peer entry in table t2 and evaluating it accordingly according to subsection 6.3. However, this information does not indicate whether the local user is the sender or receiver of the message. Fig. 4 shows such a key entry in table t7 from the generated test data. The message matching

Table 8
Decoding a Binary Large Object in the key column of table t7 containing messages and associated metadata.

Offset	Value	Meaning
0 × 00 to 0 × 07	Int64	Peer ID of the other instance
0 × 08 to 0 × 0B	Int32	namespace of the other instance
0 × 0C to 0 × 0F	Int32	Unix timestamp of the message
0 × 10 to 0 × 13	Int32	Unique message ID within a chat

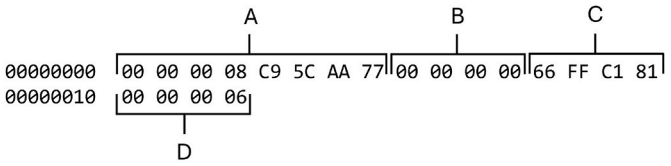


Fig. 4. The key entry is situated in the table t7 of the main database. Here, message information is encoded as a BLOB in big-endian format. The entry starts with the peer ID of the instance (A) with which the message was exchanged. It is followed by namespace (B), timestamp (C) and message ID (D).

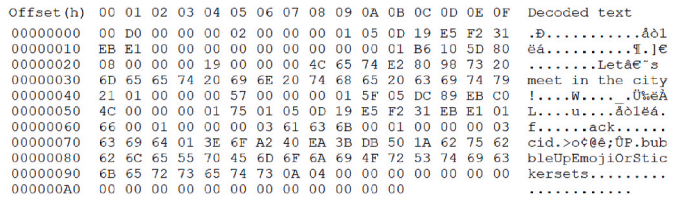
Table 9
The possible attributes of each value entry in table t7, describing a Telegram message, are listed chronologically with their data types.

Attribute	Data type	Condition
type	Int8	–
stableId	UInt32	–
stableVersion	UInt32	–
dataFlags	Int8	–
		dataFlags:
globallyUniqueId	Int64	1st bit set
globalTags	UInt32	2nd bit set
groupingKey	Int64	3rd bit set
groupInfo.stableId	UInt32	4th bit set
localTags	UInt32	5th bit set
threadId	Int64	6th bit set
flags	UInt32	–
tags	UInt32	–
forwardInfoFlags	Int8	–
		forward-InfoFlags:
forwardAuthorId	Int64	1st bit set
forwardDate	Int32	1st bit set
sourceId	Int64	2nd bit set
sourceMessageIdPeerId	Int64	3rd bit set
sourceMessageId-Namespace	Int32	3rd bit set
sourceMessageIdId	Int32	3rd bit set
authorSignature.length	Int32	4th bit set
authorSignature	String	4th bit set
psaType.length	Int32	5th bit set
psaType	String	5th bit set
forwardInfo.flags	Int32	6th bit set
hasAuthor	Int8	–
authorId	Int64	hasAuthor = 1
data.length	Int32	–
data	String	–
attributeCount	Int32	–
attributesBuffer	ObjectArray	–
embeddedMediaCount	Int32	–
embeddedMediaBuffer	ObjectArray	–
referencedMediaCount	Int32	–
Per referenced media file:		
mediaId.namespace	Int32	–
mediaId.id	Int64	–
customTagCount	Int32	–
Per custom tag:		
customTagLength	Int32	–
customTag	Object	–

the entry is from a chat with the peer ID 37738031735 and namespace 0 and is timestamped 04.10.2024 10:20:49 and the chat internal message ID 6. By decoding the peer ID and the timestamp of each entry, the message chronology for each chat can be reconstructed. The respective message content and other valuable information are coded in the corresponding value entries.

Table 9 summarises all attribute values and their data types as they occur chronologically in the value entry from table t7. The respective condition for an optional attribute value is also listed in the table.

The value entry from table t7 corresponding to the key entry from Fig. 4 is shown in Fig. 5. The figure shows that such a value entry



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   00 D0 00 00 00 02 00 00 00 01 05 0D 19 E5 F2 31  .Ð...........åò1
00000010   EB E1 00 00 00 00 00 00 00 00 01 B6 10 5D 80     ëá..........¶.]€
00000020   08 00 00 00 19 00 00 00 4C 65 74 E2 80 98 73 20  ........Letâ€˜s
00000030   6D 65 65 74 20 69 6E 20 74 68 65 20 63 69 74 79  meet in the city
00000040   21 01 00 00 00 57 00 00 00 01 5F 05 DC 89 EB C0  !....W...._.Ü‰ëÀ
00000050   4C 00 00 00 01 75 01 05 0D 19 E5 F2 31 EB E1 01  L....u....åò1ëá.
00000060   66 00 01 00 00 00 03 61 63 6B 00 01 00 00 00 03  f......ack......
00000070   63 69 64 01 3E 6F A2 40 EA 3B DB 50 1A 62 75 62  cid.>o¢@ê;ÛP.bub
00000080   62 6C 65 55 70 45 6D 6F 6A 69 4F 72 53 74 69 63  bleUpEmojiOrStic
00000090   6B 65 72 73 65 74 73 0A 04 00 00 00 00 00 00 00  kersets.........
000000A0   00 00 00 00 00 00 00 00 00 00 00 00              ............
```

Fig. 5. The content of an entry of the column value opened in a hex editor is shown, which originates from the table t7 of the local central database of a Telegram account. This column encodes information about an exchanged message as a BLOB in the little-endian format.

contains a BLOB encoded in little-endian format. Unlike many other table entries in the local main database db_sqlite, the BLOB starts directly with attribute values instead of a RootObjects without specifying keys or data types. The structure of a value entry in table t7 can be reconstructed from the justInsertMessage() function within the open source file MessageHistoryTable.swift, where the occurrence of specific optional attributes depends on the value of a previous flag (dataFlags or forwardInfoFlags).

Optionally, a message may have a unique identification number stored as a globallyUniqueId value. In this case, the first bit of the dataFlags is set.

The message has not been sent if the first bit of the flags attribute is set. If sending a message fails, a second bit is set. A set third bit of the flags value indicates an incoming message. The message is in the Sending process if the fifth bit is set. A set seventh bit of the flags value indicates a scheduled message. If a corresponding bit is not set, the negated statement applies.

For forwarded messages, the peer ID of the original sender (forwardAuthorId) and the original timestamp may optionally appear in the entry if the first bit of the forward-InfoFlag is set. The value of the optional attribute source-MessageIdPeerId contains the peer ID of the chat from which the message originated. The ID of a message within the originating chat is optionally stored as sourceMessageIdId. Messages can, in principle, be forwarded from any type of chat, with the exception of Secret Chats. For a message forwarded from a channel, the forwardAuthorId and sourceMessageIdPeerId correspond to the peer ID of the channel.

If a message was signed with the original sender's name, the entry optionally has this signature as authorSignature. If the value of the attribute hasAuthor is one, it is followed by the peer ID of the sender (authorId). The textual content of a message is contained in the data value.

Further meta information is optionally stored in message attributes, which are located in an attributesBuffer. Each type of message attribute has exactly one class with an individual MurmurHash3 value, which can be derived from the class name. Such classes always inherit from MessageAttribute and are located in the open source project of Telegram iOS, each in a separate.swift file under submodules/TelegramCore/Sources/SyncCore/. However, we will not go into more detail due to the scope, complexity and usually low forensic added value of these classes.

Information about media files sent in messages is either in a ObjectArray called embeddedMediaBuffer or listed as referenced media. If media files are referenced in the value entry, their namespace and id are listed consecutively at the end of the entry. The embeddedMediaBuffer can contain one or more encoded objects for a media file. Here, an object that inherits from the structure Media represents a media file. The respective class of the object depends on the type and origin of the media file. Media exchanged in Telegram can be found in the media folder of the local user account. In addition, the class TelegramMediaAction represents system messages or actions within a chat. The class has the signature 0 × 81 07 78 BC (little-endian format), which results from the application of the MurmurHash3 algorithm. Objects of this class also inherit from the structure Media and are stored in the embeddedMediaBuffer if applicable. Actions include, for example, creating a group and adding or removing members. Furthermore, an object of the class TelegramMediaAction, whose _raw-value is 14, characterises a voice or video call in the embeddedMediaBuffer of the value entry.

Table 10
Decoded message entry from Fig. 5.

Offset	Attribute	Value
0 × 00	type	0
0 × 01 to 0 × 04	stableId	208
0 × 05 to 0 × 08	stableVersion	2
0 × 09	dataFlags	1
0 × 0A to 0 × 11	globally-UniqueId	−2167583876353291003
0 × 12 to 0 × 15	flags	0 × 0 → outgoing
0 × 16 to 0 × 19	tags	0
0 × 1A	forwardInfo-Flags	0 → Message was not forwarded.
0 × 1B	hasAuthor	1
0 × 1C to 0 × 23	authorId	36513321142
0 × 24 to 0 × 27	data.length	19 (characters)
0 × 28 to 0 × 40	data	Let's meet in the city!
0 × 41 to 0 × 44	attributeCount	1
0 × 45 to 0 × 9F	attributes-Buffer	Contains a OutgoingMessageInfo-Attribute object:
0 × 54 to 0x5E	uniqueId	−2167583876353291003
0 × 5F to 0 × 65	flags	1
0 × 66 to 0 × 6E	acknowledged	1 ≅ true
0 × 6F to 0 × 7B	correlationId	5826316420226641726
0 × 7C to 0 × 9F	bubbleUpEmoji-OrStickersets	Empty array
0 × A0 to 0 × A3	embeddedMedia-Count	0 → No embedded media
0 × A4 to 0 × A7	referenced-MediaCount	0 → No referenced media
0 × A8 to 0 × AB	custom-TagCount	0 → No custom tags

Table 10 gives an overview of the decoded message entry already shown in Fig. 5. That is an outgoing text message sent from a cloud chat by the local user with peer ID 1116960140.

7. Conclusion

When analysing Telegram Messenger on iOS, it became apparent that there are fundamental differences to the Messenger version on Android described by Anglano et al. (2017) in terms of structures. The most likely reason is that the iOS version of the messenger was switched to Telegram X in October 2018. BLOBs must first be decoded in both messenger versions before a thorough analysis. Anglano et al. (2017) used parts of Telegram's open-source code for this without going into coding the data within BLOBs. This work goes one step further for the iOS version of the messenger and analyses the BLOBs of the most relevant tables in detail. That enables a forensic evaluation independent of the Telegram code written in Swift. The insights gained were successfully validated using the generated test data. However, further tests should ideally be carried out using actual mass data to identify any unrecognised exceptional cases in the database structure. The findings from this work refer to version 11.1.1 of Telegram Messenger, although a large part can be transferred to versions from version 5.0 onwards. However, certain deviations are possible in different versions concerning the database and table structures, as the development team of Telegram often updates the messenger, adjusting the coding within the central local database if necessary. Therefore, it is urgently necessary to regularly check the publicly accessible source code of the messenger for changes.

In future work, further tables of the db_sqlite database could be examined and documented regarding their forensic relevance. The recovery options for deleted data should also be investigated. In addition,

the analysis of further files would complete the forensic documentation of Telegram Messenger on iOS devices. For example, the evaluation of log files could allow conclusions to be drawn about deleted data.

Appendix A

Table A.11
Data type specific coding of the attribute values

Data type (key)	Coding of the attribute value
Int32 (0 × 00)	4 byte value
Int64 (0 × 01)	8 byte value
Boolean (0 × 02)	1 byte value = 0 × 00 (false) or 0 × 01 (true)
Double (0 × 03)	8 byte value
String (0 × 04)	4 byte lenght n + n byte value
Object (0 × 05)	4 byte hash of the object class name + 4 byte lenght n + n byte coded attributes
Int32Array (0 × 06)	4 byte size n + n · 4 byte value
Int64Array (0 × 07)	4 byte size n + n · 8 byte value
ObjectArray (0 × 08)	4 byte size n + n · (hash of the object class name + 4 byte lenght i + i byte coded attributes)
ObjectDictionary (0 × 09)	4 byte size n + n · (4 byte hash of the key class name + 4 byte key lenght i + i byte key content + 4 byte hash of the value class name + 4 byte value lenght j + j byte value content)
Byte (0 × 0A)	4 byte lenght n + n byte value
NIL/NULL (0 × 0B)	–
StringArray (0 × 0C)	4 byte size n + n · (lenght i + i byte value)
ByteArray (0 × 0D)	4 byte size n + n · (lenght i + i byte value)

References

Anglano, C., 2014. Forensic analysis of WhatsApp messenger on android smartphones. Digit. Invest. 11, 201–213. https://doi.org/10.1016/j.diin.2014.04.003.

Anglano, C., Canonico, M., Guazzone, M., 2017. Forensic analysis of telegram messenger on android smartphones. Digit. Invest. 23, 31–49. https://doi.org/10.1016/j.diin.2017.09.002. http://www.sciencedirect.com/science/article/pii/S1742287617301767.

Bhatt, A.J., Gupta, C., Mittal, S., 2018. Network forensics analysis of iOS social networking and messaging apps. In: 2018 Eleventh International Conference on Contemporary Computing (IC3), pp. 1–6.

Cellebrite DI Ltd, 2024. Cellebrite UFED. URL: https://cellebrite.com/en/ufed/.

Clift, J., 2024. DB Browser for SQLite Wiki. URL: https://github.com/sqlitebrowser/sqlitebrowser/wiki.

Fehily, C., 2020. SQL Database Programming, 5 ed. Questing Vole Press.

Gregorio, J., Alarcos, B., Gardel, A., 2018. Forensic analysis of telegram messenger desktop on MacOS. Int. J. Res. Eng. Sci. 6, 39–48.

Gregorio, J., Gardel, A., Alarcos, B., 2017. Forensic analysis of telegram messenger for Windows phone. Digit. Invest. 22, 88–106. https://doi.org/10.1016/j.diin.2017.07.004.

Husain, M.I., Sridhar, R., 2010. iForensics: forensic analysis of instant messaging on smart phones. In: Goel, S. (Ed.), Digital Forensics and Cyber Crime. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 9–18.

Hörz, M., 2024. HxD - Freeware Hex Editor and Disk. URL: https://mh-nexus.de/en/hxd/.

Kenneth, M., Ovens, G.M., 2016. Forensic analysis of Kik messenger on iOS devices. Digit. Invest. 17, 40–52. https://doi.org/10.1016/j.diin.2016.04.001.

Mahajan, A., Dahiya, M., Sanghvi, H., 2013. Forensic analysis of instant messenger applications on android devices. Int. J. Comput. Appl. 68. https://doi.org/10.5120/11602-6965.

Moreb, M., 2022a. Forensic analysis of telegram messenger on iOS and android smartphones case study. In: Practical Forensic Analysis of Artifacts on iOS and Android Devices: Investigating Complex Mobile Devices. Apress, Berkeley, CA, pp. 151–193. https://doi.org/10.1007/978-1-4842-8026-3_5.

Moreb, M., 2022b. Introduction to mobile forensic analysis. In: Practical Forensic Analysis of Artifacts on iOS and Android Devices: Investigating Complex Mobile Devices. Apress, Berkeley, CA, pp. 1–36. https://doi.org/10.1007/978-1-4842-8026-3_1.

Salamh, F.E., Mirza, M.M., Hutchinson, S., Yoon, Y.H., Karabiyik, U., 2021. What's on the horizon? An in-depth forensic analysis of android and iOS applications. IEEE Access 9, 99421–99454. https://doi.org/10.1109/ACCESS.2021.3095562. https://ieeexplore.ieee.org/document/9477591/.

Satrya, G.B., Daely, P.T., Nugroho, M.A., 2016. Digital forensic analysis of telegram messenger on android devices. In: 2016 International Conference on Information Communication Technology and Systems (ICTS), pp. 1–7.

Sihombing, H., Fajar, A., Utama, D., 2018. Instant messaging as information goldmines to digital forensic. Syst. Rev. https://doi.org/10.1109/ICIMTech.2018.8528089.

SQLite Documentation, 2024. WAL-Mode File Format. URL: https://www.sqlite.org/walformat.html.

Telegram, 2018. Telegram X: Progress through Competition. URL: https://telegram.org/blog/telegram-x.

Telegram, 2024a. Telegram Applications: Source Code. URL: https://telegram.org/apps#source-code.

Telegram, 2024b. Telegram FAQ. URL: https://telegram.org/faq.

We Are Social, DataReportal, Meltwater, 2024. Most Popular Global Mobile Messenger Apps as of April 2024, Based on Number of Monthly Active Users (In Millions). https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.