Retrieving deleted records from Telegram

By:
Alexandros Vasilaras, Donatos Dosis, Michael Kotsis and Panagiotis Rizomiliotis

From the proceedings of
The Digital Forensic Research Conference
DFRWS APAC 2022
Sept 28-30, 2022

Forensic Science International: Digital Investigation 43 (2022) 301447

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2022 APAC - Proceedings of the Second Annual DFRWS APAC

Retrieving deleted records from Telegram

Alexandros Vasilaras [a, *], Donatos Dosis [b, **], Michael Kotsis [b], Panagiotis Rizomiliotis [a]

[a] Department of Informatics and Telematics, Harokopio University of Athens, 9 Omirou str., Athens, 17778, Greece
[b] Department of Informatics and Computer Engineering, University of West Attica, 28 Ag. Spyridonos str., Athens, 12243, Greece

ARTICLE INFO

Article history:

Keywords:
Telegram
SQLite forensics
Android forensics
Mobile forensics

ABSTRACT

One of the most popular instant messaging applications and platforms is Telegram. In this paper, an investigation into digital forensics on Telegram is provided, deployed on contemporary android mobile phones. Due to its structure complexity, it is very challenging for forensic tools and software to properly decode its data during a forensic examination, and, especially, when it comes to deleted records retrieval. For the analysis, a realistic scenario was implemented by exchanging thousands of messages and media files among four active Telegram users, while trying to recover content from deleted text messages and exchanged picture files. Abundance of modern and up to date forensic software and tools were utilized to verify and crosscheck the results.

1. Introduction

Telegram is a cloud-based open-source social media application launched in 2013. Telegram users can exchange not only messages and media files, but also create and run chat rooms, and private channels. Telegram protects end-users' privacy, offering, among others the ability to create secret, end-to-end encrypted and self-destructing messages and chats (Telegram, 2021). Today, Telegram is available in 155 countries with more than 550 million active users per month. The country with the most installations (over 210 million) is India, followed by Russia and Indonesia (Dean, 2021).

By default, it does not enable end-to-end (E2E) encryption in normal chats and once a message arrives at the Telegram servers, it is encrypted by using its own MTProto (v2) encryption protocol, while at rest on the servers. However, Telegram could potentially read chat data since it handles encryption and decryption of the messages at its servers. E2E encryption is supported only if the secret chat feature is chosen. These two different approaches of messages handling (normal - secret) vary the structure of the data inside the Telegram android application. By extension, this affects the analysis by most forensic tools and software.

In this paper, the ability of retrieving deleted records from contemporary android devices using Telegram application is investigated. Even though Telegram supports many different types of interactions among its users (chat groups, channels, social media) this Paper focuses only on the "normal" and "secret" instant messaging between two users. The main difference between these two chat modes is the E2E encryption implementation. To the best of this paper's researchers' knowledge, this is the first systematic research focused on deleted chat records retrieval with Telegram. The analysis assumes a realistic and demanding scenario among four different active users exchanging thousands of chat messages, both text and media files.

Telegram's android application utilizes SQLite database for storing its data. Forensically the most interesting parameters of an SQLite database are the mode of journaling, the Secure Delete, the Auto Vacuum and the presence of encryption.

The examined Telegram application version (7.9.3) released on August 2021 used WAL (Write-Ahead-Log) as Journaling mode with a checkpoint set to 1.000 pages and has the Auto Vacuum and Secure Delete modes disabled.

By using the WAL as Journaling mode, the main database is left intact by committing the changes to the separate file until a checkpoint is reached. The examination of the WAL file provides information about the most recent state of the database. For instance, recently deleted data cannot be found in the main database but might be located in the database's WAL file. Furthermore, the lack of a Vacuum mode usually means that records after their deletion remain inside the database untouched forever.

In Table 1 the vital data regarding Telegram text and picture messages are shown.

* Corresponding author.
** Corresponding author.
E-mail addresses: vasilaras@hua.gr (A. Vasilaras), cscyb2010@uniwa.gr (D. Dosis), cscyb2015@uniwa.gr (M. Kotsis), prizomil@hua.gr (P. Rizomiliotis).

https://doi.org/10.1016/j.fsidi.2022.301447

Table 1
Telegram Artifacts' Directories/Files.

Content	Directory/ File Name
Telegram Database	/data/org.telegram.messenger/files/**cache4.db**
Copies of exchanged picture-media files in normal chat	/media/0/Telegram/Telegram Images
Copies of outgoing picture-media files in normal chat	/media/0/Pictures/Telegram
Picture-media files and their thumbnails	/media/0/Android/data/org.telegram.messenger/cache

1.1. Research purposes

This research was focused on whether and under which conditions is possible to retrieve deleted records from Telegram message chats, including exchanged picture-media files. The volatility or the persistence of the dataset was investigated under three different variables/conditions: (1) the elapsed time until device's forensic acquisition, (2) the device's status (power on/off, airplane mode) and (3) the interaction with the messaging application itself (e.g., creating, reading, deleting messages). In addition, several well-known forensic tools and software were evaluated, regarding their ability and accuracy of the analyzed data.

1.2. How this paper contributes to digital forensics

This research focuses on the possibility of retrieving deleted records from contemporary android devices using Telegram application. The creation of a scenario which simulates, as closely as possible, realistic situations by exchanging thousands of messages among four active Telegram users and leads to an integrated view productions and more accurate results than other related theoretical works. The main findings of the forensic analysis of the Telegram application's artifacts on modern Android devices are summarized below:

- Data volatility and how a user's actions on both Android device and the Telegram application would affect them.
- Artifacts that could be retrieved from Android OS regarding Telegram application existing apart from application's directories.
- The differences between normal and secret chat feature.
- The behavior of the SQLite database.
- The importance of validating examinations results.

1.3. How this paper is organized

The rest of the paper is structured as follows. In Section 2, related work is referenced. In Section 3, the workflow of the experiments, the followed methodology, as well as the forensic and non-forensic tools and software used for extracting and analyzing the devices, the Telegram application and SQLite databases, are presented. Section 4 contains the Analysis of Telegram application's artifacts and the ones discovered in Android devices' OS related to deleted chat records. In Section 5, artifacts related to media files are analyzed. In Section 6, variety of other findings are reported. More precisely, in Section 6.1 the Dual Application Mode is analyzed, in Section 6.2 the findings regarding the application's cloud servers and stored data are given. In Section 6.3, software and tools capabilities in application's data Decoding/Parsing are presented. Finally, in Section 7, the paper is concluded.

2. Related work

There are many guidelines about various forensic tools and techniques regarding Mobile Forensics (Nemetz et al., 2018; Easttom, 2021a, 2021b; Gogolin, 2021; Reiber, 2019; Tamma et al., 2020). There is also previous research on SQLite structure and techniques on how to retrieve deleted records and data carving from such databases (Pawlaszczyk and Hummert, 2021) (Punja and Whiffin, 2021). Pawlaszczyk D. and Hummert C. (2021) have develop an open-source forensic tool implementing those techniques, making possible to recover deleted records from a database. There is also research about the significance to conduct thorough forensic analysis of an SQLite database in order to be aware of the value of deleted records.

Anglano et al. (2017) documented Telegram application' s file and folder structure and analyzed its database. Additional forensic analysis of Telegram's database has been done through the years (Boiko Бойко, 2018) (dfirfpi, 2020).

There are also some correlations with other IMAs like WhatsApp and Viber (Sutikno et al., 2016).

3. Methodology

Experiments Setup: For the experiments two different devices were used. An LG G6 (H870) with Android 9 (security patch dated May 2019) and a Samsung A50 (A505FN/DS) with Android 11 (security patch dated July 2021). Since the access to the paths presented in Table 1 is normally prohibited, *root* access was gained on both devices. This was achieved by unlocking the bootloader and installing a custom recovery menu (Team Win Recovery Project-TWRP) and a Magisk solution (Manager, 2021). The devices' user-data directory was wiped prior to the experiments. On both devices different sim cards were activated and the Telegram application was installed using the same version of apk (through *adb -d install* command), a new user was added, and no additional applications were installed (e.g., google accounts) to mitigate the level of "noise" and unrelated data. Four different Telegram accounts exchanged the total number of the messages. This setup is not considered a forensically sound method for actual evidence's examination but was necessary to implement a white box methodology. The topic of extracting a mobile device's full file system or physical image is out the scope of this research.

Experiments Methodology: The experiments were divided into three phases as depicted in the workflow (Fig. 1). Namely, the reconnaissance phase, the main phase and the last phase. In the first phase an identification of general information about

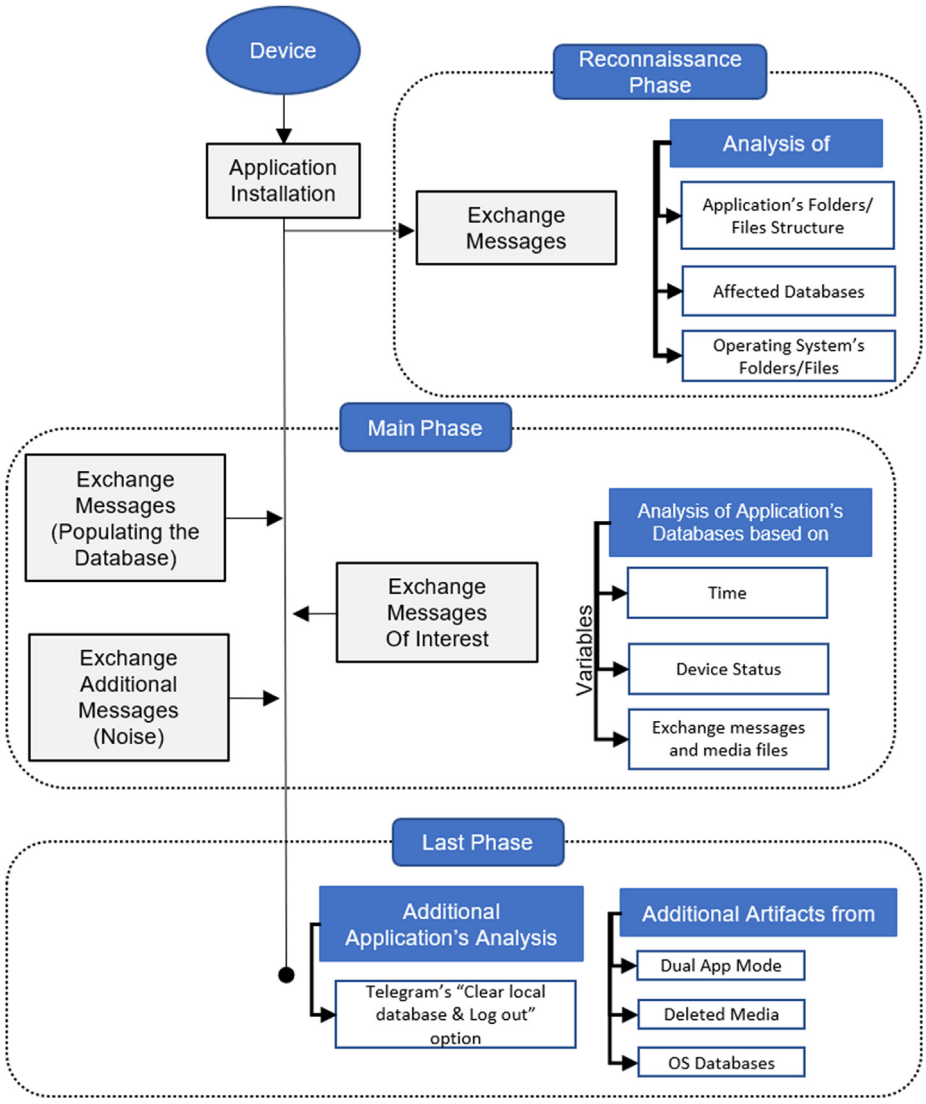


Fig. 1. Workflow of methodology.

Telegram's structure and how interacts with the Android OS, was made. In the second phase the volatility or the persistence of the deleted records was checked under three different variables/conditions: (1) the elapsed time until device's forensic acquisition, (2) the device's statuses (power on/off, airplane mode) and (3) the interaction with the messaging application itself (e.g., creating, reading, deleting messages). In the last phase, several Telegram's artifacts residing in the device examined, as well as the possibility of retrieving any information from its Servers.

Experiments Data: In order to examine as many potential occasions an examiner may face, as possible, several forensic images were acquired, in various time frames and in different statuses of the devices (Table 2). Time frames included the image acquisition of the device or the database of interest in specific time intervals. On the other hand, (1) leaving the device in normal mode (standby), (2) placing it in a specialized Faraday Box, (3) rebooting or shutting it down and (4) terminating all device communications by setting it in Airplane mode were examined as devices possible different. Total conducted experiments number is shown in Table 3.

During the experiments many media files were exchanged created both by default camera application and the Telegram's camera feature. All the media files were deleted manually and removed as well from the trash bin folder. For the extraction and analysis of the devices' and Telegram application's data a vast number of forensic tools commercial and non were utilized (Table 4) and compared. In order to monitor the alterations in any interaction with the application targeted acquisitions of specific directories or files (database) were made via pull commands.

By no means the utilized tools in this Paper are not the only ones that can be used for the same purposes.

Table 2
Reconnaissance phase variables.

Status	Time Frame	Action
Standby	As soon as possible	None
Standby	After 2/6/24 h	None
Standby	As soon as possible	Removing Sim
Airplane mode	After 2/6/24 h	None
After reboot	24 h	Removed Sim
After shutdown	48 h	None
Faraday box	After 2/6/24 h	None

Table 3
Total conducted experiments' number.

Physical Images	Acquiring Database's folder			Exchanged Messages
	Reconnaissance Phase	Main Phase	Last Phase	
12	6	180	–	2.200

Table 4
Tools utilized during the study.

Usage	Tool	License
Forensics Analysis	Access Data (Forensic ToolKit-FTK) 7.4	Commercial
	Autopsy Suite 4.19.1	Open Source
	Belkasoft X 1.10	Commercial
	Cellebrite UFED Physical Analyzer 7.50	Commercial
	Magnet Axiom 5.7, 6.0	Commercial
	MSAB (XRY and XAMN)	Commercial
	Oxygen forensics (Oxygen Detective) 14.1	Commercial
	slo-sleuth/android blob cache (https://github.com/slo-sleuth/android_blob_cache.) (media files)	Open Source
	X-Ways Forensics 20.1 (& HEX analysis)	Commercial
SQLite Analysis	aLEAPP 1.9.9	Open Source
	Andriller 3.6.1	Open Source
	Belkasoft X 1.10	Commercial
	bring2lite (Meng and Baier, 2019)	Open Source
	DB Browser for SQLite 3.12.2	Open Source
	Forensic Toolkit for SQLite	Commercial
	FQlite 1.5.5	Open Source
	KS DB Merge Tools for SQLite	Free Tool & Commercial
	Oxygen forensics (SQLite Viewer) 14.1	Commercial
	Paraben E3 Forensic Platform 3.1	Free Tool & Commercial
	SQLECmd (https://github.com/EricZimmerman/SQLECmd.)	Open Source
	SQLite-Deleted-Records-Parser (https://github.com/mdegrazia/SQLite-Deleted-Records-Parser.)	Open Source
	SQlite Expert 5.4	Free Tool & Commercial
	SQLite Forensic Explorer 2.0.0	Free Tool & Commercial
	Teleparser (https://github.com/RealityNet/teleparser.)	Open Source
	undark 0.6 (https://github.com/alitrack/undark.)	Open Source
Image Acquisition	ADB Commands (CLI) & SDK Platform Tools 31.03	Open Source
	Belkasoft X 1.10	Commercial
	Cellebrite UFED 4 PC 7.48	Commercial
	Magnet Acquire 2.45	Free Tool
	MSAB (XRY and XAMN)	Commercial
	Oxygen forensics (Oxygen Extractor) 14.1	Commercial

4. Telegram database's analysis and findings

Comparing the two examined mobile devices there were no differences between them, in the folder structure or the files of the application. In any case the initial size of **cache4.db** was 1.336 kb and its corresponding WAL file was 4.475 kb. The database was composed by fifty-two (52) tables. Forensically, the four most interesting tables for the examined scenario were the *dialogs*, *enc_chats*, *messages,* and *users* whose most valuable fields are presented in Table 5. The meaning of the fields is a result of this research combined with the one of Anglano et al. (2017). Moreover, tables *user_contacts_v7* and *user_phones_v7* contained information about the users participating in chat threads.

By executing queries or parsing with forensic tools (Table 6) combining the fields shown in Table 5 several conclusions were drawn from the analysis of the data inside **cache4.db**. An example of query results is presented in Fig. 2.

Some initial conclusions were:

- The same message resides in the WAL file several times in different states.
- In a secret chat the participants name could not be directly decoded, and a special process had to be done (Fig. 6).

Table 5
Total conducted experiments' number.

Field	Meaning
	Table: dialogs
did	Dialog id of the secret or normal chat
date	10-digit timestamp of a chat-thread's last message in Unix
unread count	Information about the unread messages of the dialog
last_mid inbox_max outbox_max	In conjunction could clarify the size of the dialog
	Table: users (All users, including owner)
uid	User's unique id number 10-digit form
name	User's name
data	User's connected number in binary form (BLOB)
	Table: enc_chats
uid	Secret Chat's User unique id number
user	User's unique id number (connected with uid of table users)
name	User's name (connected with name of table users)
	Table: messages
uid	Participant's unique id
read_state	Indicates whether a message has been read
send_state	Clarified whether an outgoing message has been sent
date	Timestamp of the message in Unix Seconds
body	Body of text messages, along with several other information, in a binary serialized form (BLOB)
out	Indicates a message's direction
TTL	Autodeleting messages timer (in seconds) set by the user

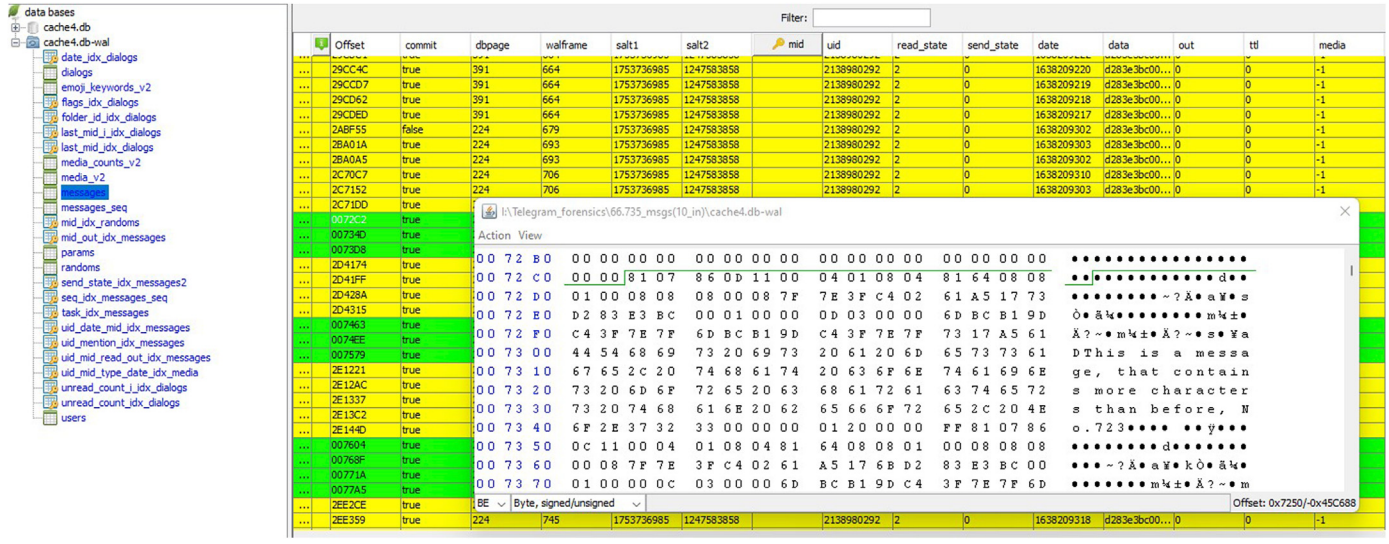
Table 6
SQLite queries.

DB Browser for SQLite

```
SELECT
users.name AS 'Name',
messages.UID,
messages.MID AS 'Message ID',
users.data AS 'Info',
datetime( messages.date, 'unixepoch', 'localtime' ) AS 'Date',
CASE messages.SEND_STATE
    WHEN '1' THEN 'The message has not reached Server'
    ELSE 'The message was delivered to the Server'
END AS 'Message Local Status',
CASE messages.out
    WHEN '0' THEN 'Incoming'
    ELSE 'Outgoing'
END AS 'Message Direction',
CASE messages.read_state
    WHEN '2' THEN 'Non Received'
    WHEN '3' THEN 'Received'
END AS 'Message status',
messages.data AS 'Body',
messages.TTL AS 'Timer',
messages.sfSource AS 'Source'
FROM messages
LEFT JOIN users ON users.uid = messages.uid
ORDER BY messages.date ASC
```

	Name	UID	Message ID	Info	Date	Message Local Status	Message Direction	Message status	Body	Timer	Source
168	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:05	The message has not reached Server	Outgoing	Non Received	BLOB	30	WAL-96-commit
169	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:06	The message was delivered to the Server	Outgoing	Non Received	BLOB	30	WAL-111-commit
170	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:06	The message was delivered to the Server	Outgoing	Received	BLOB	30	WAL-115-commit

FQLite Carving Tool

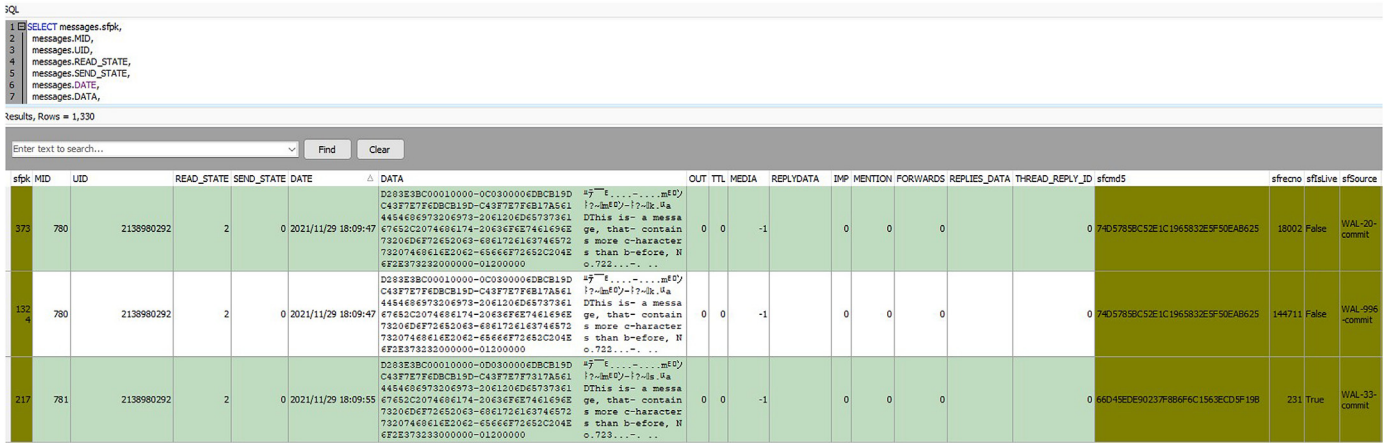


(*continued on next page*)

Table 6 (*continued*)

DB Browser for SQLite

Forensic Browser for SQLite



- The autodelete feature did not affect the ability to recover this kind of messages, since it did not have any other additional properties except the data in column *TTL*.
- The body of the messages is in a binary form (BLOB) and could not be easily interpreted.

4.1. Telegram Data Structure (TDS) term

As noted, the content of some fields is in binary form (BLOB). Anglano et al. (2017) named this type of data as "TDS" (Telegram Data Structure). In the same research (2017) referred that the serialized data are structured in the Telegram Language as composite types (include integers and strings) and defined by a 4-byte integer number in little Indian. Each 4-bytes integer comprise one object type. In April 2020 through the examination of 42.000 lines of Telegram code, 1.340 objects were found (dfirfpi, 2020). Telegram Team adds, removes or redefines these objects by adding or removing application's features. In 2018 Boiko M. explains some of these objects, even though many of them had been redefined until this research is conducted. Analysis of some TDSs and objects found in the examined application's version are presented below throughout examples. In the first example there is an exchange of two messages (one incoming and one outgoing) of a normal chat thread (records 162 and 163) (Fig. 3). The same is presented in Fig. 4 for a secret chat thread (records 225 and 226). The observation and the analysis of these two figures and the included continuously changing 4-byte integers (objects), explains the difficulty of decoding the database. Some of the most important objects (i.e., users, timestamps, type of chat thread) are labelled with different colors and explained inside each figure. In Fig. 5 the decoding of TDS objects of an outgoing media file (photo) in a Secret chat message is depicted.

Table 7
The state of **cache4.db** and its WAL file in different conditions.

Database's State	Database's Size (in kb)	freelist count	Messages in db	Messages in WAL
Initial	1.336	0	0	0
Normal	1.568	22	198	123
Complete Deletion	1.568	39	225	33
Commitment of 20 incoming messages	1.568	39	5	165

	Name	UID	Message ID	Info	Date	Message Local Status	Message Direction	Message status	Body	Timer	Source
168	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:05	The message has not reached Server	Outgoing	Non Received	BLOB	30	WAL-96-commit
169	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:06	The message was delivered to the Server	Outgoing	Non Received	BLOB	30	WAL-111-commit
170	NULL	-7424347666923913216	-210225	NULL	2021-11-29 22:44:06	The message was delivered to the Server	Outgoing	Received	BLOB	30	WAL-115-commit

Fig. 2. SQLite query result on secret chat.

	Name	Info	Date	Message Status	Message Direction	Message status	Body
162	bad actor 2;;;	BLOB	2021-11-27 23:52:33	NULL	Incoming	Received	BLOB
163	bad actor 2;;;	BLOB	2021-11-27 23:53:31	NULL	Outgoing	Received	BLOB

Records including normal chat messages.

Table: users

	uid	name	status	data
	Filter	Filter	Filter	Filter
1	1903238211	zugni feyda;;;	1638046912	BLOB
2	1923643811	bad actor 2;;;	-100	BLOB
3	2135101684	μιχάλης;;;	-100	BLOB
4	2138980292	edgar allan poe;;;	-100	BLOB

The "users" table of the "cache4.db". The record 1 (device's user) and record 2 are the participants.

```
0000 d2 83 e3 bc 00 01 00 00 c5 00 00 00 6d bc b1 9d  ............m...
0010 a3 79 a8 72 6d bc b1 9d a3 79 a8 72 a1 a8 a2 61  .y.rm....y.r...a
0020 16 54 68 69 73 20 69 73 20 6d 65 73 73 61 67 65  .This is message
0030 20 4e 6f 2e 31 36 30 00 01 20 00 00              No.160.. ..
```

Objects in record 162 (Normal Incoming Message)

BCE383D2	Integer flagging this TDS as a normal message between two users
72A879A3	The hex value of the user's id sending the message in the conversation (1923643811)
72A879A3	The hex value of the user's id participating in the conversation (1923643811)
61A2A8A1	The timestamp of the message (1638049953) in Unix Seconds

```
0000 d2 83 e3 bc 02 03 00 00 c6 00 00 00 6d bc b1 9d  ............m...
0010 43 1c 71 71 6d bc b1 9d a3 79 a8 72 db a8 a2 61  C.qqm....y.r...a
0020 16 54 68 69 73 20 69 73 20 6d 65 73 73 61 67 65  .This is message
0030 20 4e 6f 2e 31 36 31 00 20 63 ed 3d 01 20 00 00   No.161. c.=. ..
0040
```

Objects in record 163 (Normal Outgoing Message)

BCE383D2	Integer flagging this TDS as a normal message between two users
71711C43	The hex value of the user's id sending the message in the conversation (1903238211)
72A879A3	The hex value of the user's id participating in the conversation (1923643811)
61A2A8DB	The timestamp of the message (1638050011) in Unix Seconds

Fig. 3. TDSs in Normal Chat thread.

4.2. Volatility of cache4.db

The experiments confirmed that both the **cache4.db** and its corresponding WAL file were susceptible to certain device status changes but did not seem to alter the data inside the *messages* table. Those actions included: (i) leaving the device powered on and with network, (ii) after the finishing boot sequence of the device and the application auto starts when network was available and (iii) at opening of the app after closing it.

Under the following circumstances both files were not modified: (i) leaving the device powered on in airplane mode, (ii) rebooting/turning off the device while it was in airplane mode, (iii) at closing the app and removing it from the task manager of the android device, (iv) removing the SIM card (which was the only thing associated with account), while the device is live and after it was deactivated and then powered on without the SIM and (v) removing notifications of incoming messages.

The database's *messages* table was modified when: (i) a message was created, no matter whether it was successfully sent or not, (ii) a message was received, (iii) a message was opened. This was regardless of whether it was an outgoing message which was read by the receiver or an incoming which was read by the user and (iv) a message or a thread was deleted.

There was a difference in the gravity of the acts. An outgoing message occupied more records and space in the WAL file than reading an incoming message. The element of luck is also involved. A crucial factor is the timing of the in-question messages deletion. Whether it occurs right after a WAL commit to the database, or before it. This is a parameter purely random and cannot be controlled in realistic situations. In Fig. 7 is presented a list of the different experiments which were executed (*Test No*), the prior condition before deleting the in-question messages (*Parameters prior to deletion*), the different actions in which they were still recoverable and are green colored (*Incoming/Outgoing/Opened/Deleted Messages*) and the action which finally led to the removal of the chat thread from the main database or the WAL file and is red colored (*Parameters prior to deletion*).

As mentioned **cache4.db** did not have the Auto Vacuum feature enabled and thus it creates freelist pages in which potentially deleted messages could be retrieved. However, even if the Secure Delete feature is also disabled, there was some kind of failsafe for wiping the deleted records. This was tested multiple times in both devices by deleting hundreds of records, leading to freelist pages creation, and then attempting to recover them. In every case, immediately after the deletion, most of the messages were recoverable, but after committing some new ones (incoming or

225	NULL	NULL	2021-11-29 20:24:26	NULL	Outgoing	Received	BLOB
226	NULL	NULL	2021-11-29 20:24:39	NULL	Incoming	Received	BLOB

Records including secret chat messages.

Table: users

	uid	name	status	data
	Filter	Filter	Filter	Filter
1	1903238211	zugni feyda;;;	1638046912	BLOB
2	1923643811	bad actor 2;;;	-100	BLOB
3	2135101684	μιχάλης;;;	-100	BLOB
4	2138980292	edgar allan poe;;;	-100	BLOB

The "users" table of the "cache4.db". The record 1 (device's user) and record 3 are the participants.

```
0000 fa 55 55 55 03 03 00 00 fd ca fc ff 00 00 00 00  .UUU............
0010 43 1c 71 71 6d bc b1 9d f4 10 43 7f da 1a a5 61  C.qqm.....C....a
0020 1d 54 68 69 73 20 77 69 6c 6c 20 63 6f 73 74 20  .This will cost
0030 79 6f 75 20 31 30 30 30 20 65 75 72 6f 73 00 00  you 1000 euros..
0040 20 63 ed 3d 15 c4 b5 1c 00 00 00 00 00 00 00 00   c.=............
0050
```

Objects in record 225 (Secret Outgoing Message)

555555FA	Integer flagging this TDS as a secret message between two users
71711C43	The hex value of the user's id sending the message in the conversation (1903238211)
7F4310F4	The hex value of the user's id receiving the message in the conversation (2135101684)
61A51ADA	The timestamp of the message (1638210266) in Unix Seconds

```
0000 fa 55 55 55 01 03 00 00 fc ca fc ff 00 00 00 00  .UUU............
0010 f4 10 43 7f 6d bc b1 9d 43 1c 71 71 e7 1a a5 61  ..C.m...C.qq...a
0020 19 57 65 20 61 67 72 65 65 64 20 6f 6e 20 66 69  .We agreed on fi
0030 76 65 20 68 75 6e 64 72 65 64 00 00 20 63 ed 3d  ve hundred.. c.=
0040 15 c4 b5 1c 00 00 00 00 00 00 00 00              ............
```

Objects in record 226 (Secret Incoming Message)

555555FA	Integer flagging this TDS as a secret message between two users
7F4310F4	The hex value of the user's id sending the message in the conversation (2135101684)
71711C43	The hex value of the user's id receiving the message in the conversation (1903238211)
61A51AE7	The timestamp of the message (1638210279) in Unix Seconds

Fig. 4. TDSs in Secret Chat thread.

outgoing) all the messages residing in the main database were erased (Table 7). This was verified by searching specific messages as a string, in the database and WAL file, with a hex editor.

4.3. Wiped databases

In Telegram there were two occasions where no records of the database were recoverable. (i) When the user logged out of the device, triggering a complete database's wipe and (ii) by clearing the local database from the settings menu. In any case the difference between the timestamp of **cache4.db** and the Android' OS usage history application, might be a strong indicator that one of the above occasions have occurred.

4.4. SQLite database analysis summary

There are possibilities to retrieve deleted records (messages) from:

- the main database file (cache4.db) if there are no new commitments after the records' deletion
- the WAL file, which is the main database's supplementary file, under different and volatile conditions (examples are shown in Fig. 7).

5. Telegram's media-pictures files attachments

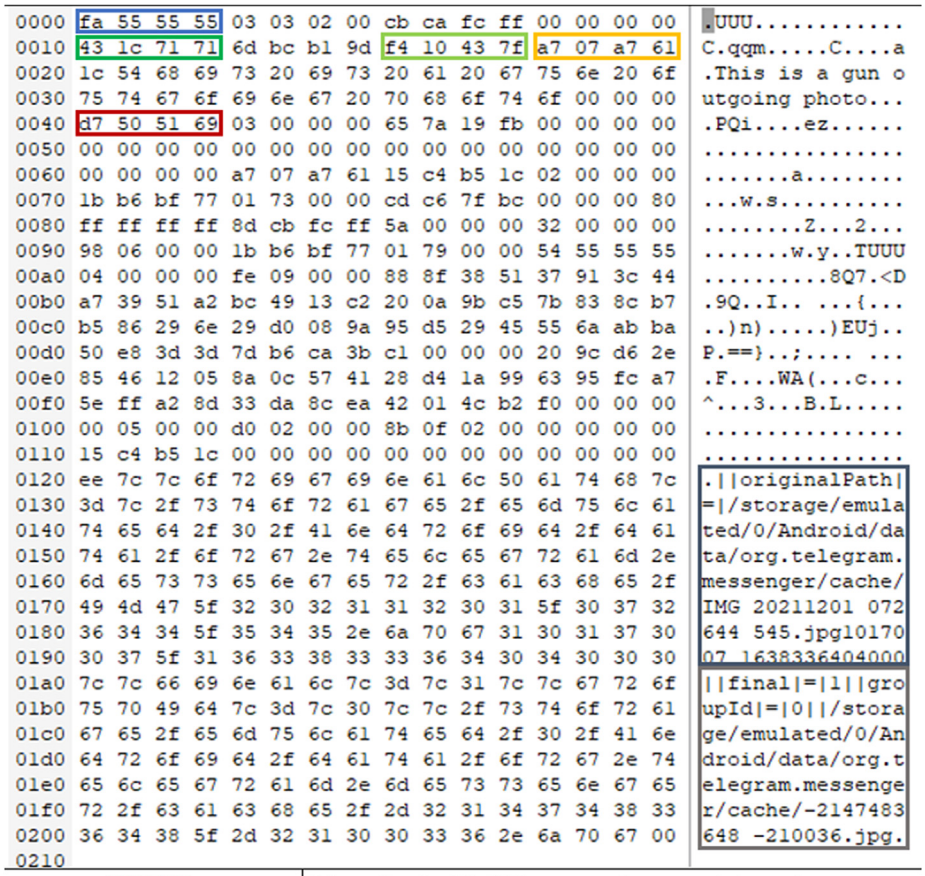
Telegram administrated the shared media files, in a unique way based on the type of the chat thread (normal/secret).

5.1. Normal chat media transmission

After a media file (picture) deletion some artifacts which have remained on the android operating system were utilized to retrieve its information. To be more accurate this was done by comparing those artifacts with the records located in **cache4.db**. Those artifacts' paths were for Samsung device: *data/com.sec.android.gallery3d/cache*, and for the LG device: (i) *media/0/DCIM/.thumbnails* and (ii) *media/0/Android/data/com.android.gallery3d/cache/imgcache.0*. For the latter one the conversion (https://github.com/slo-sleuth/android_blob_cache.) of the blob file *imagecache.0* was necessary (Fig. 9).

5.2. Secret chat media transmission

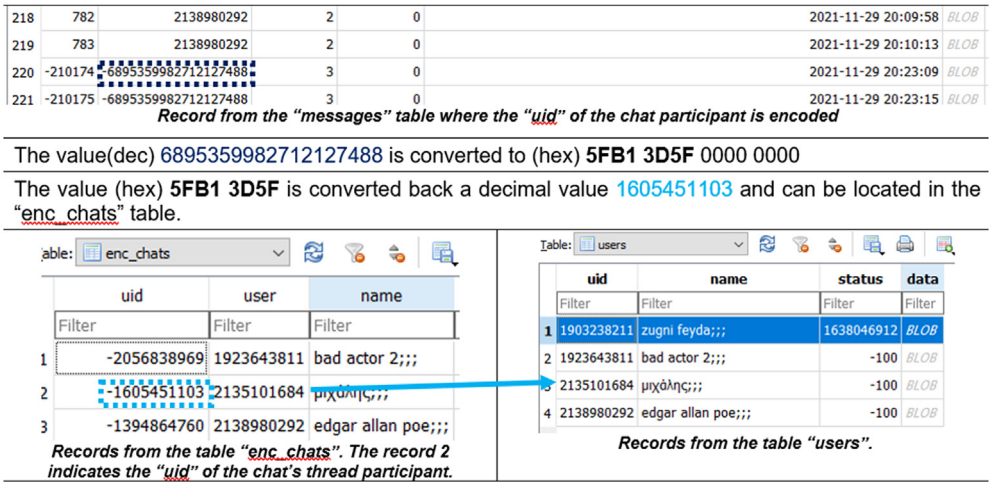
In a secret chat when the media were captured with in-app camera feature never resided outside the cache folder of the application. After deleting the secret chat messages, it was impossible to retrieve/determine the content of the files



555555FA	Integer flagging this TDS as a secret message between two users
71711C43	The hex value of the user's id sending the message in the conversation (1903238211)
7F4310F4	The hex value of the user's id receiving the message in the conversation (2135101684)
61A707A7	The timestamp of the message (1638336423) in Unix Seconds
695150D7	End of the media message caption

The original path of the media file and the temporary path in the cache folder of the application are readable in ASCII characters.

Fig. 5. Outgoing media file TDS in Secret Chat thread.



218	782	2138980292	2	0	2021-11-29 20:09:58	BLOB
219	783	2138980292	2	0	2021-11-29 20:10:13	BLOB
220	-210174	-6895359982712127488	3	0	2021-11-29 20:23:09	BLOB
221	-210175	-6895359982712127488	3	0	2021-11-29 20:23:15	BLOB

Record from the "messages" table where the "uid" of the chat participant is encoded

The value(dec) 6895359982712127488 is converted to (hex) **5FB1 3D5F** 0000 0000

The value (hex) **5FB1 3D5F** is converted back a decimal value 1605451103 and can be located in the "enc_chats" table.

Table: enc_chats

	uid	user	name
	Filter	Filter	Filter
1	-2056838969	1923643811	bad actor 2;;;
2	-1605451103	2135101684	μιχάλης;;;
3	-1394864760	2138980292	edgar allan poe;;;

Records from the table "enc_chats". The record 2 indicates the "uid" of the chat's thread participant.

Table: users

	uid	name	status	data
	Filter	Filter	Filter	Filter
1	1903238211	zugni feyda;;;	1638046912	BLOB
2	1923643811	bad actor 2;;;	-100	BLOB
3	2135101684	μιχάλης;;;	-100	BLOB
4	2138980292	edgar allan poe;;;	-100	BLOB

Records from the table "users".

Fig. 6. How to decode participant' s *uid* from a Telegram's Secret Chat.

Test No	Parameters prior to deletion	DELETION OF THE IN-QUESTION CHAT THREAD	Incoming Messages	Outgoing Messages	Messages that were opened	Messages that were deleted	Action that removed the messages in-question
1	None		15	5	15	None	10 outgoing messages
2	All messages of the database were deleted		50	None	20	None	the rest 30 messages were opened
3	60 messages included on the thread alongside with in-question ones were deleted		50	None	None	80 (From other chat thread)	50 messages were opened
4	None		20		60	None	10 incoming messages
5	WAL commit was forced[1]		20	15	None	None	40 messages were opened
6	All messages of the database were deleted, and WAL commit was forced		70	None	None	None	70 read
7	None		50	None	None	None	70 read

Fig. 7. Progress of experiments' results on **cache4.db** and its WAL file.
[1.] The database was monitored in real time by exchanging single messages, until a WAL commit occurred.

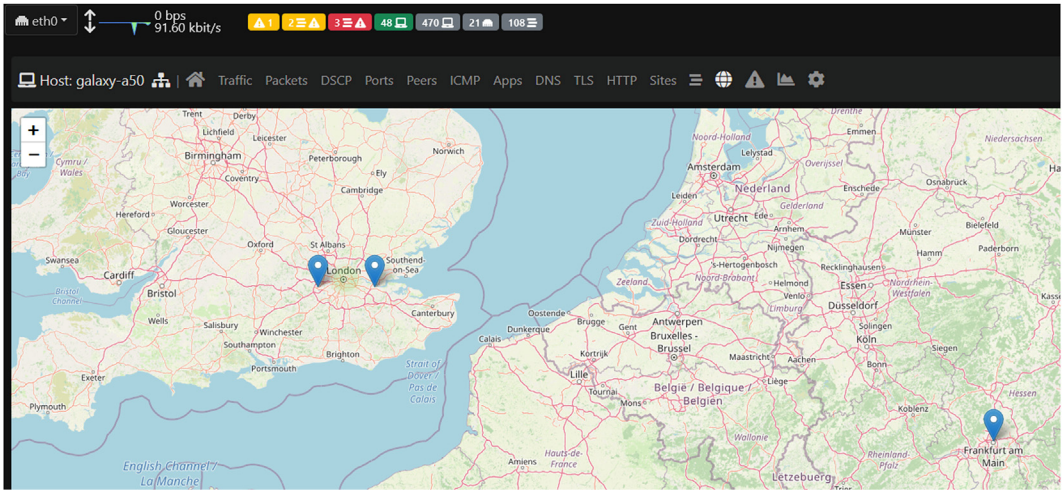


Fig. 8. Telegram application's cloud servers where the devices connected (tool used: ntopng with geoip plugin, device examined: Samsung A50).

(attachments) which were sent. During the experiments, it was observed that the corresponding thumbnail files had always a minus one in its file name values than the ones in the database. For instance, the thumbnail of the transmitted media named *-2147483648_-2100**2**.jpg*, but in the database record its name was *-2147483648_-2100**3**.jpg* (Fig. 10). This was a recurring pattern, but it was not decoded.

6. Other findings

6.1. Dual Application Mode

In the experimental course the Dual app mode was also examined. It was branded with different names depending on the device i.e., Dual apps for LG and Dual Messenger for Samsung. When this feature was enabled a second user/account was created in the filesystem with individual folders and files and separate read/execute/write permissions. Thus, for each user distinct Telegram directories and files were stored.

6.2. Application's cloud servers

Telegram is registered as an American LLC and UK LLP, with headquarters in Dubai. The messages originating from Greece were relayed/routed from servers in UK (Fig. 8). As a 2021 FBI training document indicates (Cimpanu, 2021) there is «no messages and contact information provided for law enforcement to pursue a court order. As per Telegram's privacy statement, for confirmed terrorist investigations, Telegram may disclose IP and phone number to relevant authorities».

6.3. Software decoding/parsing capabilities

With the volume of TDS being added, removed and modified, forensic tools and software are unable to keep up with their decoding. Some commercial ones had different levels of success, while others could not decode Telegram's data at all. Some simple examples are that Oxygen Forensics 14.1 could properly decode the undeleted secret media messages but not the normal ones (Fig. 11). On the contrary in UFED PA 7.50 the exact opposite occurred (Fig. 12). One other aspect is the versioning of the software and tools. For instance, AXIOM 5.7 was unable to decode either secret or normal media messages (Fig. 13), while the AXIOM 6.0 could decode only the normal ones (Fig. 14). This is also the case when it comes to analyzing only the SQLite database of the application.

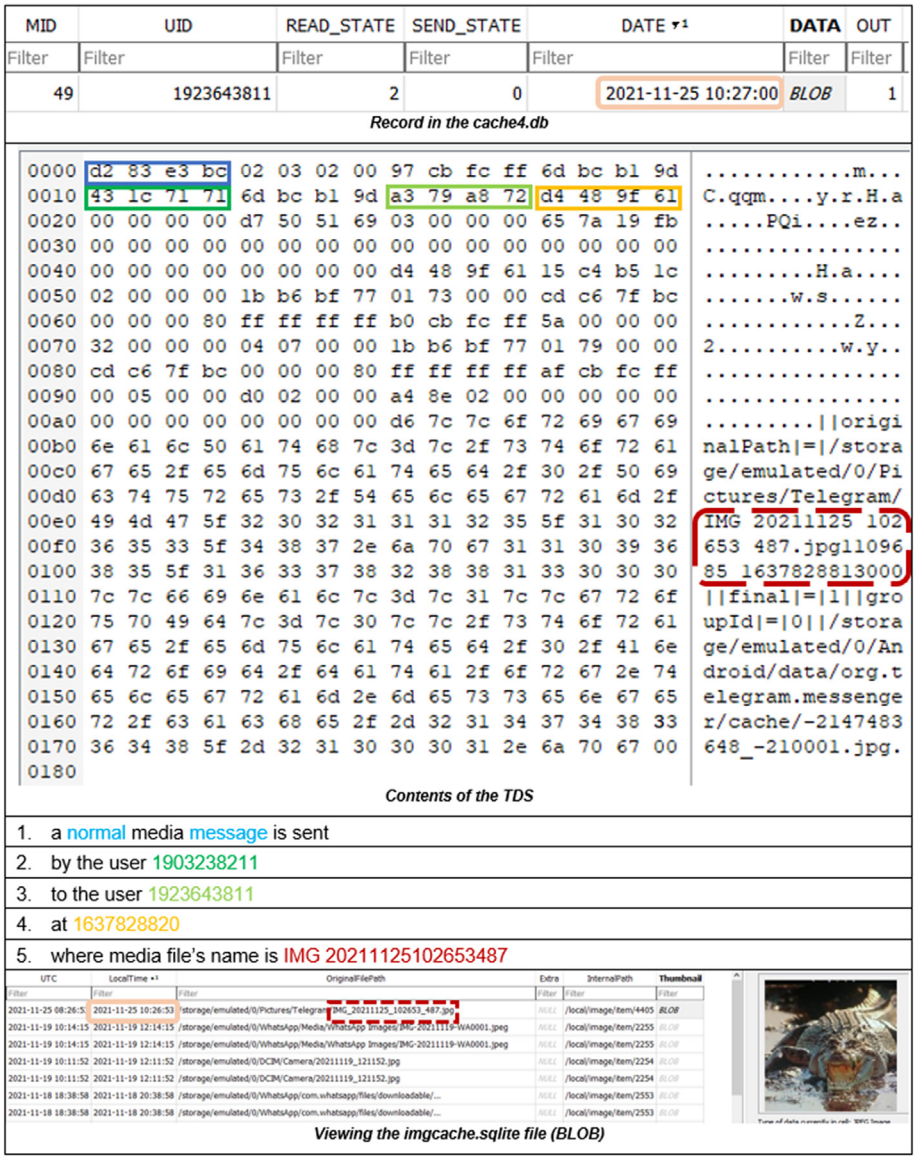


Fig. 9. Correlation of a sent image file's record in **cache4.db** with *imgcache.sqlite* file (tool used: DB Browser for SQLite, Examined Device: LG G6).

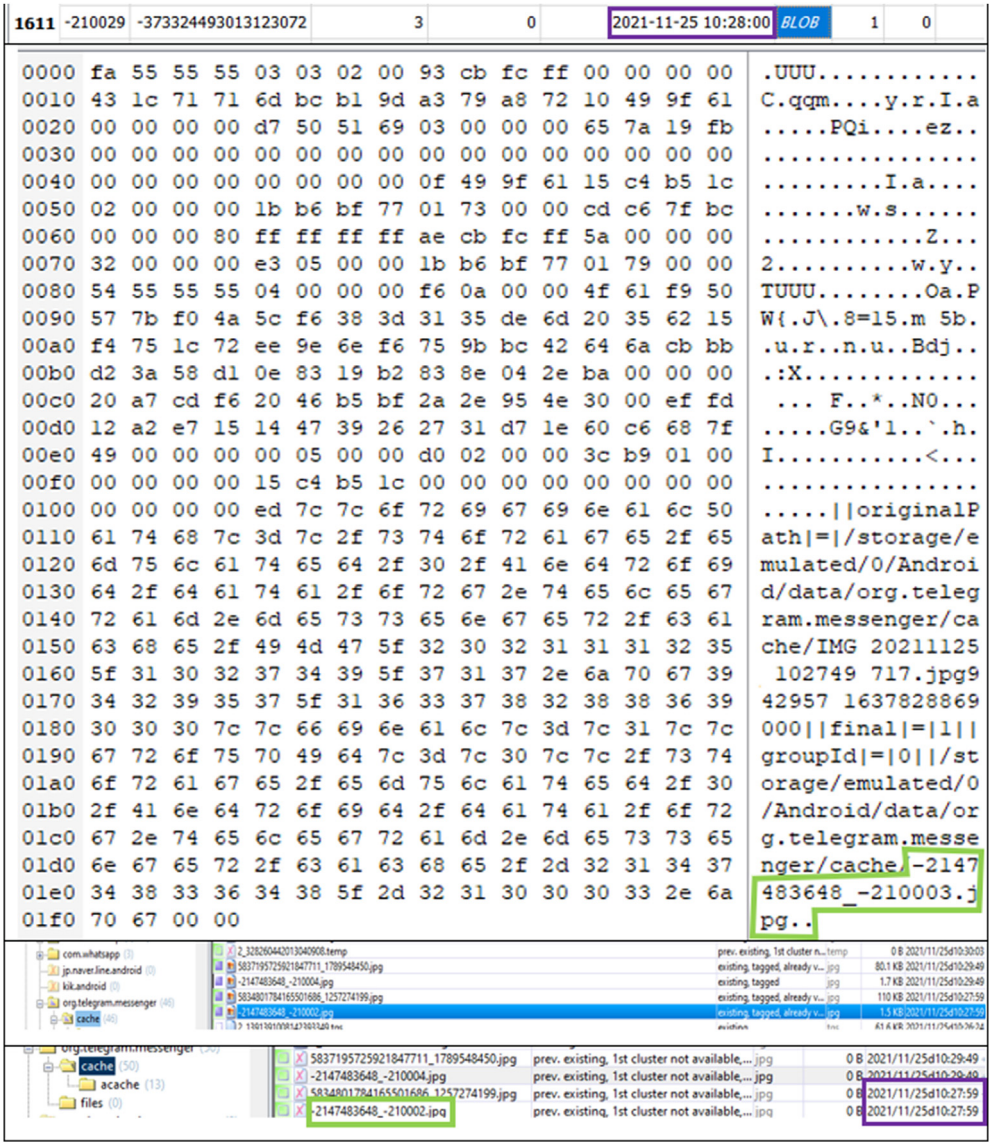


Fig. 10. Detection of a sent image file in **cache4.db** from a Secret chat thread.

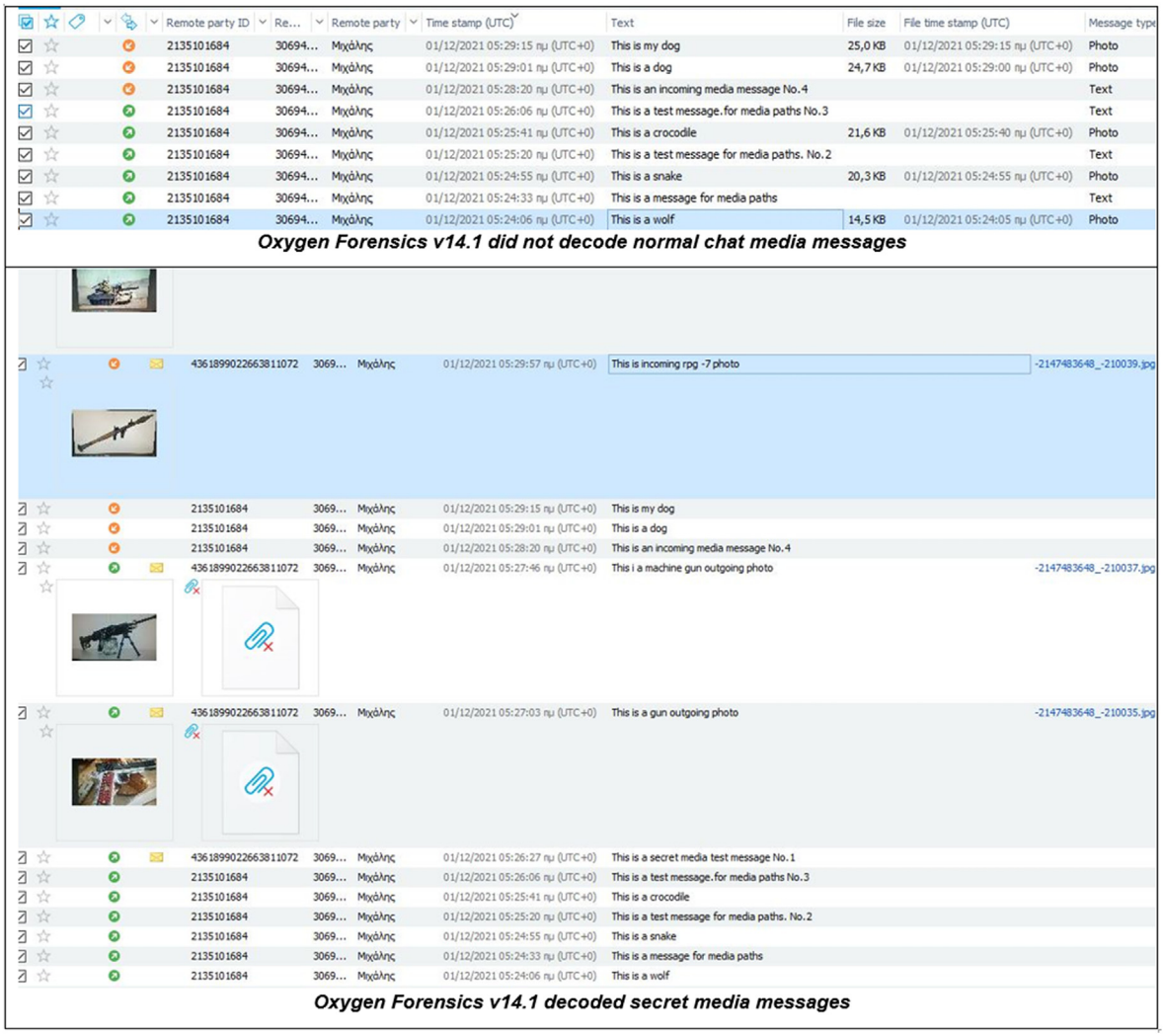


Fig. 11. Oxygen Forensics 14.1 parsing/decoding normal and secret chats.

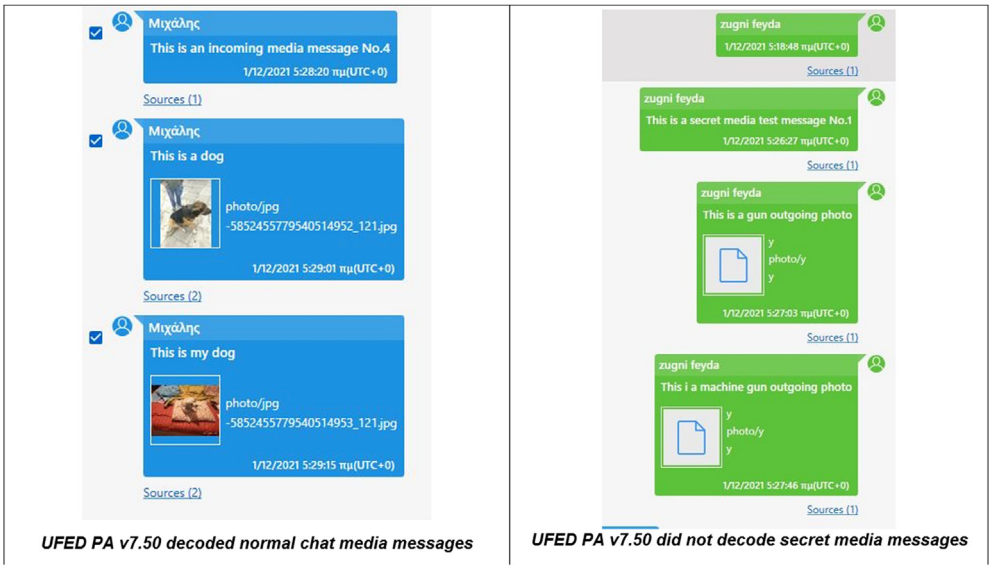


Fig. 12. UFED PA 7.50 parsing/decoding normal and secret chats.

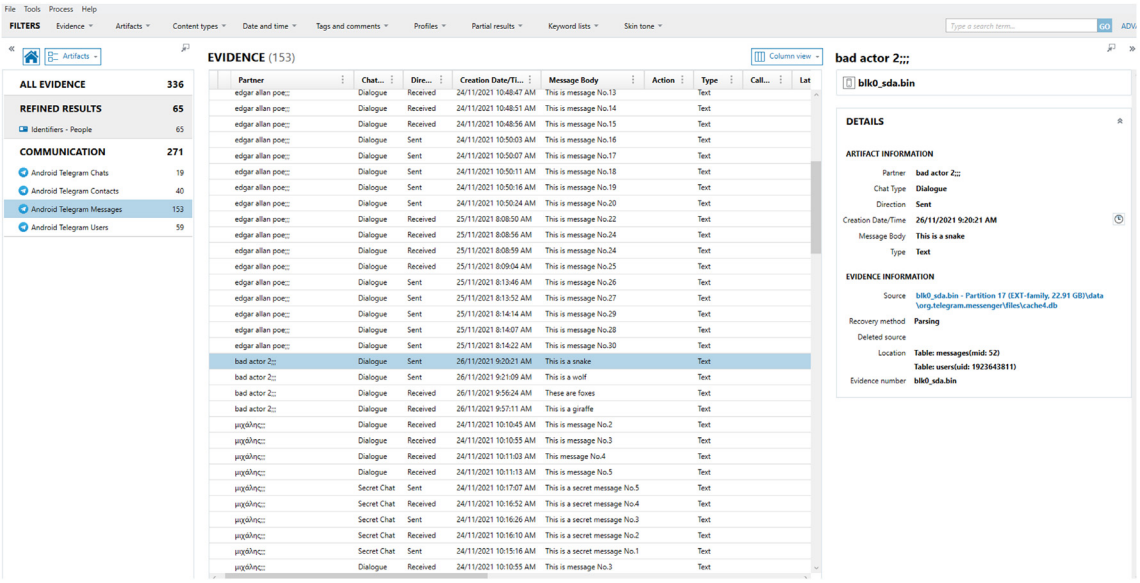


Fig. 13. AXIOM 5.7 parsing/decoding media files (pictures) normal and secret chats.

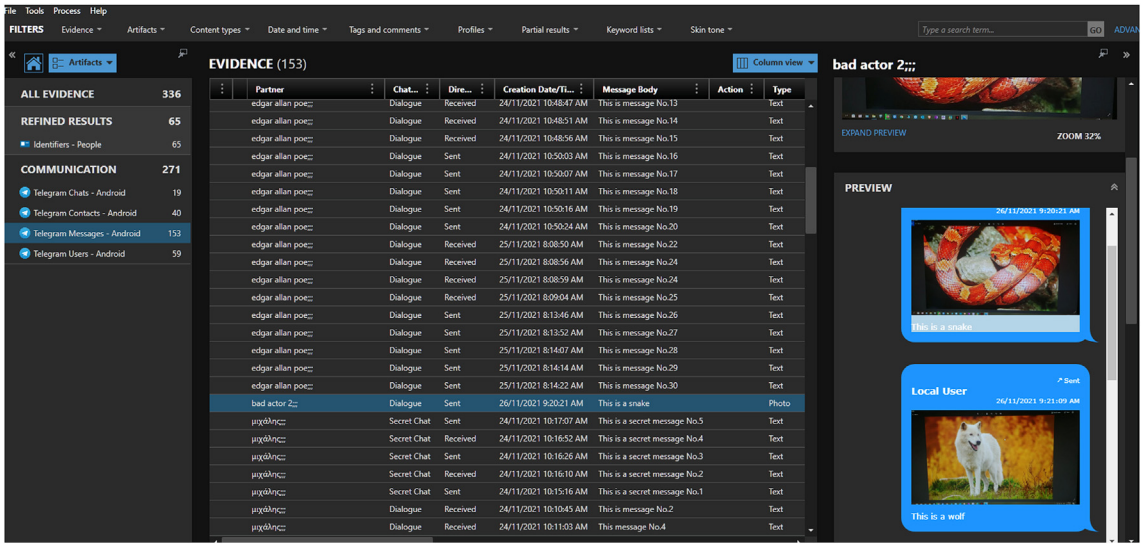


Fig. 14. AXIOM 6.0 parsing/decoding media files (pictures) normal and secret chats.

7. Conclusions

Telegram's structure is complex and very challenging for forensics investigation. Its database's characteristics (i.e., WAL journaling mode without Auto Vacuum and Secure Delete features), makes it feasible to retrieve a large number of deleted records from the WAL as long as the examiner has the least interaction with the application. It was impossible to recover deleted records from database's freelist pages. As for deleted media files (pictures) the recovery of relevant artifacts depended solely on the chat thread type (normal/secret). Due to lack of information from application's cloud Servers, the only way to retrieve deleted messages and media files is by examining the device where the application is installed. Forensic tools and software were challenged as far as decoding several artifact types. Their results had to be validated and cross-checked. That could be done by using several tools and their latest versions.

As future work, more android applications should be examined to find out if there are more volatile data on their databases and how react in different occasions.

References

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 published on: 20/09/2017.

Boiko Бойко, M., 2018. TELEGRAM ELECTRONIC CORRESPONDENCE RESEARCH on ANDROID DEVICES («ДОСЛІДЖЕННЯ ЕЛЕКТРОННОЄ ПЕРЕПИСКИ TELEGRAM НА ПРИСТРОЯХ ANDROID»), Kiev, 2018, CACHE4.DB FILE of TELEGRAM for ANDROID (PART 1), Digital Forensics (dflab.blogspot), published on: 03/01/2019, retrieved from URL: https://dflab.blogspot.com/2019/01/cache4db-file-of-telegram-for-android.html. last accessed on 05/12/2021.

Cimpanu, C., 2021. FBI document shows what data can be obtained from encrypted messaging apps. created on: 30/11/2021, retrieved from URL: https://therecord.media/fbi-document-shows-what-data-can-be-obtained-from-encrypted-messaging-apps. last accessed 09/12/2021.

Dean, B., 2021. How Many People Use Telegram in 2021? 55 Telegram Stats,

BACKLINKO published on: 14/10/2021, retrieved from URL: https://backlinko.com/telegram-users. last accessed on 05/12/2021.
dfirfpi, 2020. teleparser, ZENA FORENSICS something about digital forensics and something not. published on: 06/04/2020, retrieved from URL. https://blog.digital-forensics.it/2020/04/teleparser.html. (Accessed 5 December 2021).
Easttom, C., 2021a. An In-Depth Guide to Mobile Device Forensics. CRC Press, Boca Raton, Florida. https://doi.org/10.1201/9781003118718 published on, 22/10/2021.
Easttom, C., 2021b. Digital Forensics, Investigation, and Response Fourth Edition. Information Systems Security & Assurance Series, Jones & Bartlett Learning, Burlington, Massachusetts, USA, ISBN 9781284226065 published on 24/08/2021.
Gogolin, G., 2021. Digital Forensics Explained, second ed. CRC Press, Boca Raton, Florida, ISBN 9780367503437. published on: 12/04/2021.
https://github.com/alitrack/undark.
https://github.com/EricZimmerman/SQLECmd.
https://github.com/mdegrazia/SQLite-Deleted-Records-Parser.
https://github.com/RealityNet/teleparser.
https://github.com/slo-sleuth/android_blob_cache.
Manager, Magisk, 2021. Download Magisk Manager Latest Version 23.0 for Android 2021 retrieved from URL: https://magiskmanager.com. (Accessed 9 November 2021).
Meng, C., Baier, H., 2019. bring2lite: a structural concept and tool for forensic data analysis and recovery of deleted SQLite records. Digit. Invest. 29, 31–41. https://doi.org/10.1016/j.diin.2019.04.017 published on: 14/07/2019.
Nemetz, S., Schmitt, S., Freiling, F., 2018. A standardized corpus for SQLite database forensics. Digit. Invest. 24, 121–130. https://doi.org/10.1016/j.diin.2018.01.015 published on: 21/03/2018.
Pawlaszczyk, D., Hummert, C., 2021. Making the invisible visible –techniques for recovering deleted SQLite data records. Int. J. Cyber Forensics. Adv. Threat Investig. Creative Commons License. 1 (1–3), 27–41. https://doi.org/10.46386/ijcfati.v1i1-3.17 last accessed 09/11/2021.
Punja, S., Whiffin, I., 2021. Missing SQLite Records Analysis, DFIR Review. https://dfir.pubpub.org/pub/33vkc2ul. (Accessed 30 December 2021). created on: 29/07/2021, Retrieved from URL.
Reiber, L., 2019. Mobile Forensic Investigations, A Guide to Evidence Collection, Analysis, and Presentation, second ed. McGraw-Hill Education, ISBN 978-1-26-013510-7. eBook published 2019.
Sutikno, T., Handayani, L., Stiawan, D., Riyadi, M.A., Subroto, I., 2016. WhatsApp, viber and telegram: which is the best for instant messaging? Int. J. Electr. Comput. Eng. 6 (3), 909–914. https://doi.org/10.11591/ijece.v6i3.10271 published on: 26/05/2016.
Tamma, R., Skulkin, O., Mahalik, H., Bommisetty, S., 2020. Practical Mobile Forensics Fourth Edition, Forensically Investigate and Analyze iOS, Android, and Windows 10 Devices. Packt Publishing, Birmingham, UK, ISBN 978-1-83864-752-0 published on: 04/2020.
Telegram, 2021. End-to-End encryption, secret chats. retrieved from URL: https://core.telegram.org/api/end-to-end. last accessed on 05/12/2021.