# Android Forensics Analysis: Private Chat on Social Messenger

**Conference Paper** · July 2016

DOI: 10.1109/ICUFN.2016.7537064

CITATIONS
18

READS
2,451

**3 authors:**

Gandeva Bayu Satrya
Telkom University
**25** PUBLICATIONS **97** CITATIONS
SEE PROFILE

Philip Tobianto Daely
Kumoh National Institute of Technology
**19** PUBLICATIONS **82** CITATIONS
SEE PROFILE

Soo Young Shin
Kumoh National Institute of Technology
**310** PUBLICATIONS **1,402** CITATIONS
SEE PROFILE

# Android Forensics Analysis: Private Chat on Social Messenger

G. B. Satrya, P. T. Daely, and S. Y. Shin
IT Convergence Engineering
Kumoh National Institute of Technology
{gandevabs, 20156130, wdragon}@kumoh.ac.kr

***Abstract*—In this paper, we discuss remnant data from private chat, secret chat, and hidden chat in social messenger applications for Android. We explain all the artifacts which are produced by social messengers. We also provide interpretations of generated messages as well as how they relate to one another. Based on the investigation results of Android forensics and analysis in this paper, an analyst or investigator will be able to read, reconstruct, and present the chronology of the messages which have been created by the user. By using two smartphones with different brands and different Android OS versions as experimental objects, we conducted a digital investigation in a forensically sound manner.**

***Index Terms*—Social messenger, remnant data, private chat, normal chat, Android forensics.**

## I. Introduction

In the 2G and 3G era, Short Message Service (SMS) was a service broadly used by mobile phone users. However, it is very different in the 4G era, where SMS is used less and social media as well as social messenger applications keep emerging. Social media applications and social messenger applications are online media which can be used free of charge by users to share and exchange information in the form of text, picture, audio, or video through the Internet. Various well-known social messenger applications usually come with interesting features such as normal chat, private chat, message notification, location sharing, contact sharing, file sharing, and status updating systems.

Based on statistical information in January 2016 [1], the total number of registered active users of social networks was 1.55 billion people. The increased number of active users and the development of internet technology can give the opportunity for some individuals, groups of people or organizations to commit cybercrime [2][3][4]. Blackmailing, identity theft, transaction fraud, and other types of criminal acts using computers or smartphones are all considered as cybercrimes.

Many kinds of social messenger applications are now available on Google Play Store and can be downloaded for free. Therefore, the fast growing market of social messenger applications raises concerns about privacy and security. In this paper, we discuss digital forensics on Android smartphones for social messenger applications according to [5]. It is hoped that this paper can help the forensics analyst or investigator in cybercrime cases which use social messenger applications.

The paper can be summarized as follows:

1) We discuss the process of acquisition, analysis, and also interpretation of private chat's metadata which are obtained from Telegram, Line, and KakaoTalk;
2) We show how the artifacts are related to one another from the results of data acquisition to data analysis of normal chat and private chat;
3) We present a guide for forensics analysts and investigators to conduct a cybercrime investigation on social messenger applications.

The rest of the paper is organized as follows. In section II we review existing work. In section III we describe the methodology which we use in analysis. Then, in section IV, we explain in detail the process of digital forensics from identification to presentation. Finally, in section V, we provide the conclusion of this paper.

## II. Related Works

Husein et al. 2009 [6] have studied and reported the forensics analysis of three different instant messengers (IMs): AIM, Yahoo! Messenger, and Google Talk (both client based and web based versions). Their work shows various useful artifacts related to IMs can be recovered from the iPhone, including username, password, buddy list, last log-in time, and conversation timestamp as well as conversation details. Forensics investigation of IM on smartphones such as the iPhone poses a new challenge for the investigators as well as researchers due to the uniqueness of file systems, lack of standard methods, and the limited number of tools.

Mahajan et al. 2013 [7] have conducted forensics data analysis on two widely used social messenger applications on Android smartphones: WhatsApp and Viber. The tests and analysis were performed with the aim of determining what kind of data and information can be found on the device's internal memory which are related to social messenger applications, e.g. chat log and history, sent and received image or video files, etc.

Anglano [8] in his paper shows how to reconstruct the list of contacts and the chronology of the exchanged messages by users. The correlation of the contents of the chat database with the information in the log files allows the investigator to determine which messages have been deleted, when these messages have been exchanged, and which users exchanged them. But this paper does not explain any hash function type in the acquisition process.

In this paper, we present details about digital forensics investigation on three social messenger application, which are Telegram, Line and KakaoTalk. The focus of the investigation was Telegram's "Secret Chat", Line's "Hidden Chat", and KakaoTalk's "Secret Chat". Two smartphones were used; Xiaomi Redmi Note 1W and Samsung Galaxy Note II. Conducting the investigation in a forensically sound manner, we executed data acquisition and data analysis. The difference between this paper and previous papers is that we performed both offline and online (live logging) forensics investigations on normal chat and private chat activities. To validate the acquired data as digital evidence in court, we implemented cryptographic hash function SHA-256 (SHA-2 family with digest length of 256 bits) as the hash value.

## III. Investigation Methodology

### A. Identification

Identification is the procedure of identifying which objects can be used as digital evidence, where those objects are stored, and what the impact of user activity will be [9]. Our role here is as a forensics analyst or investigator. In the investigation, we used two smartphones as experimental objects- Xiaomi Redmi Note 3G with Android OS v4.4.2 and Samsung Galaxy Note II SHV-E250K with Android OS v4.3. The social messenger applications that we used as study cases were Telegram v3.4.2, Line v5.10.0, and KakaoTalk v5.3.3 for Android OS.

### B. Preservation

Preservation is the most important procedure in digital forensics investigations [9]. By upholding the values of the Convention on Cybercrime [10][11], it is very important to record every event and activity that happens or is performed upon the digital evidence. In this matter, it is vital for a forensics analyst or investigator to have a certification or experience that can be accounted for in court. It is also important to conduct the investigation in a forensically sound manner, with minimal alteration to the evidence, and if there is an alteration, it must be justified in the court.

The smartphones in our experiment were in a rooted condition so that we could proceed further to data acquisition. In this data acquisition process, the tools that we used for investigation were Android Debug Bridge (ADB) v1.0.32, SQLite Browser v3.8.0, Hex Editor Neo v6.20.02.5651, Busybox v4.1, Root Browser v2.2.3, Online Nandroid Backup v4.4.5, Shark for Root v1.0.2, dex2jar v2.0, sha256sum (Linux Ubuntu 14.04) and Notepad++ v6.8.8. We used SHA-2 as the hash value for all the evidence to ensure their admissibility in court.

### C. Analysis

Analysis is a procedure in digital forensics to extract, process, and interpret digital evidence [9]. The digital evidence from the process of preservation cannot be read or used directly for presentation. With the help of the mentioned tools, we proceeded to analysis. The method we used was to compare the evidence before and after user events or activities happened. If the hash value of the evidence changed after the event or activity happened, then we would continue with a deeper analysis. Besides offline forensics investigation, we also did online forensics investigation by recording live log data, just in case the artifacts could not be acquired from the smartphone.

### D. Presentation

Presentation is the final objective of digital forensics investigations where we prove a cybercrime in court [9]. In this paper, we have presented all the activities of digital forensics investigation that were performed. This paper is expected to be a guidance for researchers, forensics analysts, investigators and other cyberlaw practitioners.

## IV. Private Conversation Digital Forensics

### A. Research Preparation

The scope of this investigation was to compare and analyze normal chat and private chat artifacts in Android devices which related to the usage of Telegram, Line and KakaoTalk application. Our experimental scenario involved two users in a one to one communication. To find out the difference between normal chat and private chat, we performed digital forensics investigations on normal chat and private chat for the three social messenger applications. The first user with a Samsung Galaxy Note II, started normal chat with the second user who was using a Xiaomi Redmi Note. We then conducted the procedures in Section `Investigation Methodology` above. After that, we conducted the same procedures for private chat.

To find the chat's artifacts, we used offline forensics investigation methods (with the smartphones backup files as the digital evidence) and online forensics investigation (with the live log record of the smartphones during the chat activities as the digital evidence). In the analysis procedure, we compared the acquired data before and after the chat activity happened. At the end of the investigation, we presented the evidence in the form of a set of files including each file's name, directory, and content.

### B. Private Chat Investigation

*1) Post-mortem Telegram:* Telegram had one database and some supporting files that were stored in directory `\data\data\org.telegram.messenger\files\*.*`. The acquired database artifacts related to normal chat that were obtained from the Telegram directory are shown in Figure 1. We can see that it is readable and without encryption. The acquired database artifacts related to secret chat are shown in Figure 2. Based on these artifacts of normal chat and secret chat, the content of a chat message is readable by using SQLite Browser. If we open each database of normal chat and secret chat using Hex Editor Neo, we will be able to see the difference between normal chat and secret chat. In normal chat, there is a hex pattern of "`XX 1e 55 56 XX`", followed by the chat text message. Meanwhile in secret chat, there is a hex pattern of "`XX 13 bf 56 XX`", followed by the chat text message.

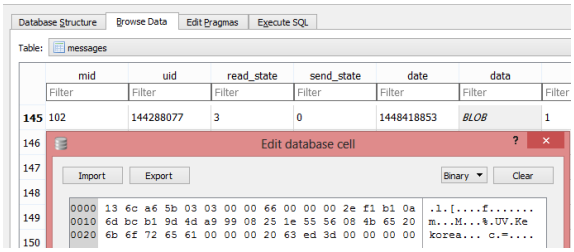



Fig. 1: Telegram: normal chat

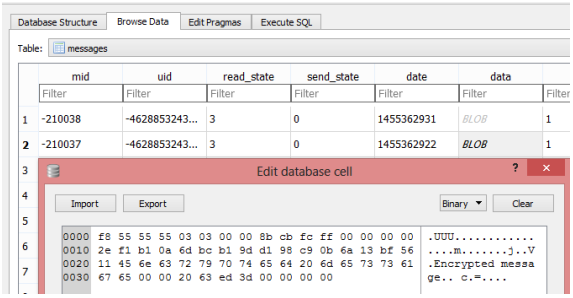



Fig. 2: Telegram: secret chat

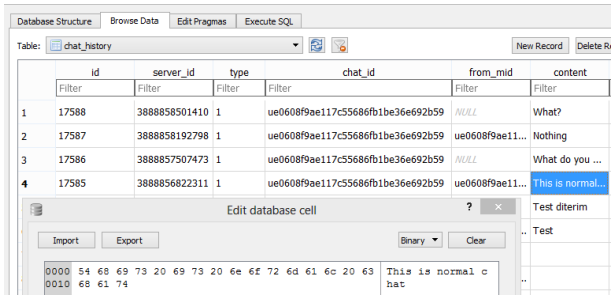



Fig. 3: Line: normal chat

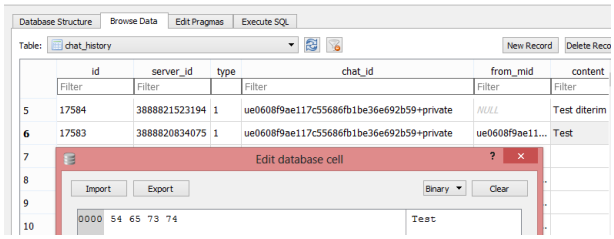



Fig. 4: Line: hidden chat

The recorded log of normal chat and private chat is shown in Table I. It shows only the timestamp of normal chat activity, while the content of chat is unclear. But, we can see the content of chat by opening the database file. The log of normal chat is shown in rows 1 and 2. Meanwhile in private chat, we can see the encryption key exchange log in the third row of Table I. We also provide the Telegram captured packet (Appendix A). From the stream of the sniffed packet, there is no difference between normal chat and secret chat. The interaction between client's smartphone (IP address: `10.8.0.1`) and Telegram server (IP address: `91.108.56.145`) is encrypted with Secure Sockets Layer (SSL).

*2) Post-mortem Line:* The investigation procedure for Line is similar as with Telegram. There are 18 database files for Line application, which are stored in directory `\data\data\jp.naver.line.android\databases\*.*`. Figure 3 shows the table `chat_history` inside database `naver_line`. From Figure 3 we can read directly the content of chat between two users, as long as the chat history has not been deleted. There is no encryption at all for the artifacts of Line's normal chat. It is a bit different with Line's hidden chat in Figure 4. Although it is also not encrypted, we need to acquire it before a certain time, because it has a self-destruct mechanism with a timer that ranges from two seconds to one week after the message is sent. If we proceed further to the analysis on table `chat_history`, there is a column `parameter` where we can see the remaining time left of the private chat message. From the timer information, we can see when the message was received by the user's smartphone. We can also look for artifacts related to destructed messages in `naver_line_private_chat` on table `destruction_message`.

For online forensics investigation of Line, we recorded the log of both smartphones during chatting. In Table II, we can see the artifacts that can be used as digital evidence for normal chat and hidden chat. From this log record, we found no significant difference between normal chat and hidden chat. This statement is backed up with the result of packet capturing of the user's smartphone during Line chatting (Appendix B). The interaction between Line user (IP address: `10.8.0.1`) with Line server (IP address: `125.209.222.202`) is encrypted with SSL.

*3) Post-mortem KakaoTalk:* The procedure of investigation for KakaoTalk was also the same as with Telegram and Line. At first, we performed an offline forensics investigation. If we open directory `\data\data\com.kakao.talk\databases\*.*`, we will find three database files, which are `KakaoTalk.db`, `KakaoTalk2.db`, and `password.db`. We focused on `KakaoTalk.db` because all the related artifacts to normal chat and secret chat on KakaoTalk is in there. There are two tables; `chat_logs` and `chat_rooms` that are shown in Figure 5 and 6. Table `chat_logs` contains encrypted details of all chat activity and Table `chat_rooms` contains the last message of chat activity.

The interesting part in KakaoTalk is shown in Figure 5. In the second row of table `chat_logs`, in column `message` it shows `"iRmTBWjfrMqWh2mPjUY4G0gUNVXhjtaOBiASHmx9QUM="`. From a quick glance, based on its characteristics, we first assumed that these ASCII characters were data that were encoded in base64 encoding schemes. But when we tried to decode it back to the original data, we were not able to get anything useful connected to KakaoTalk chatting, because another special encryption by KakaoTalk messenger is applied to it.

TABLE I: Live Telegram chat log

| Activity | Logcat Timestamp |
|---|---|
| Normal Chat (1) Text | 02-06 19:34:12.246 6089 6089 I dalvikvm: Could not find method org.telegram.ui.Cells.ChatBaseCell.onProvideStructure, referenced from method org.telegram.ui.Cells.ChatMessageCell.onProvideStructure |
| Normal Chat (2) Text | 02-06 19:34:12.311 6089 6089 I dalvikvm: Could not find method android.app.Activity.checkSelfPermission, referenced from method org.telegram.ui.Components.ChatActivityEnterView$8.onTouch |
| Secret Chat (1) Text | 02-06 22:45:10.337 3129 5624 I GCM : GCM message org.telegram.messenger 0:1455371110283840%8eeb79b0f9fd7ecd |
| Secret Chat (2) Text | 02-06 22:45:10.387 2505 3065 D ActivityManager: startService callerProcessName:org.telegram.messenger, calleePkg-Name: org.telegram.messenger |

TABLE II: Live Line chat log

| Activity | Logcat Timestamp |
|---|---|
| Normal Chat (1) Text | 02-14 16:05:56.770 174 381 I BufferQueue: [jp.naver.line.android/jp.naver.line.android.activity. chathis-tory.ChatHistoryActivity](this:0xb7d7a428,id:106,api:1,p:3741,c:174) [queue] fps:3.97, dur:1008.39, max:502.72, min:9.51 |
| Normal Chat (2) Text | 02-14 16:05:56.778 174 174 I BufferQueue: [jp.naver.line.android/jp.naver.line.android.activity. chathis-tory.ChatHistoryActivity] (this:0xb7d7a428,id:106,api:1,p:3741,c:174) [release] fps:3.96, dur:1009.81, max:502.56, min:16.19 |
| Secret Chat (1) Text | 02-14 15:50:08.559 174 472 I BufferQueue: [jp.naver.line.android/jp.naver.line.android.activity.chathistory. ChatHistory-Activity](this:0xb7d7a428,id:95,api:1,p:3741,c:174) [queue] fps:2.98, dur:1007.49, max:503.62, min:89.28 |
| Secret Chat (2) Text | 02-14 15:50:08.566 174 174 I BufferQueue: [jp.naver.line.android/jp.naver.line.android.activity.chathistory. ChatHistory-Activity](this:0xb7d7a428,id:95,api:1,p:3741,c:174) [release] fps:2.98, dur:1007.20, max:503.77, min:97.08 |
| Life time chat | 02-14 15:50:08.635 3741 3741 D ListView: mSelectorRect.setEmpty in layoutChildren this=jp.naver.line.android.activity.chathistory.list.ChatHistoryListView42a0bdb8 V.ED.VC. .....ID 0,0-720,514 #7f0f033e app:id/chatlog |

The other way for forensics analysts or investigators to investigate is shown in table `chat_rooms`. We are still able to retrieve some artifacts related to chatting from this table, but it only stores the last text message of the chat. As we have shown in Figure 6 in column `type`, the table `chat_room` also stores some artifacts related to secret chat. However, it only shows the last text message of secret chat and not the previous messages.

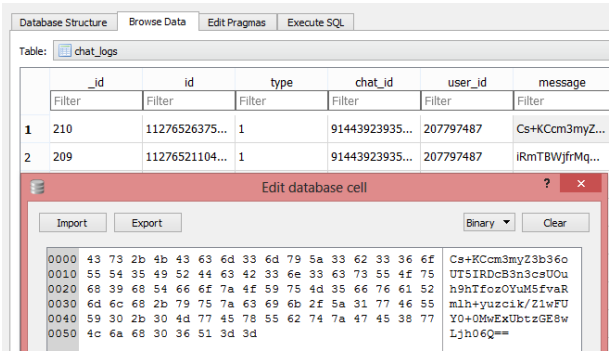



Fig. 5: KakaoTalk database: table *chat_logs*

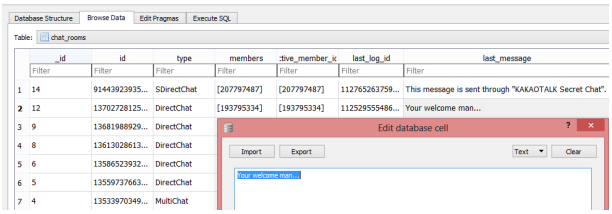



Fig. 6: KakaoTalk database: table *chat_rooms*

The online forensics investigation of KakaoTalk was conducted by recording the smartphone's log using Logcat and capturing the packet using Wireshark during the chatting activity. The artifacts in Table III are digital evidence for normal chat and secret chat. From the live log that was recorded using Logcat, we found no significant difference between normal chat and hidden chat. We can see that in the log, it shows "`package-private`" in the thread information part. A screenshot of the captured packet during the chatting activity is also provided (Appendix C). From the packet capturing results, we can see the packets during chatting and also the process of key exchange. All of them are secured with TLSv1 protocol. The packets were going in and out between the user's smartphone (IP address: `10.8.0.1`) and KakaoTalk server (IP address: `110.76.141.57`).

### C. Discussion

In Table IV we present guidance that could be useful for forensics analysts or investigators during investigation of normal chat and private chat. We have explained how to interpret artifacts related to normal chat and private chat of three social messenger applications in `Post-mortem Telegram`, `Post-mortem Line`, and `Post-mortem KakaoTalk`. In Table V we present the summary of our investigation. All of these artifacts can be used as admissible evidence in court. This investigation is conducted under the assumption that the smartphone from a crime scene has not been switched off and is unlocked. Finally, this paper is made with no intention to defame any developer or company or country. This paper aims to help improve digital forensics investigations in solving cybercrime cases, specifically cases related to Android devices.

TABLE III: Live KakaoTalk chat log

| Activity | Logcat Timestamp |
|---|---|
| starting chat | 02-14 18:40:01.466 2504 3072 D ActivityManager: startService callerProcessName:com.kakao.talk, calleePkgName: com.kakao.talk |
| Normal Chat (1) Text | 02-14 18:50:07.701 5993 5993 W dalvikvm: method Lcom/kakao/talk/widget/IcsLinearLayout;.drawDividersHorizontal incorrectly overrides package-private method with same name in Landroid/widget/LinearLayout; |
| Normal Chat (2) Text | 02-14 19:41:59.936 5932 5932 W dalvikvm: method Lcom/kakao/talk/widget/DividedLinearLayout;.drawDividersHorizontal incorrectly overrides package-private method with same name in Landroid/widget/LinearLayout; |
| Secret Chat (1) Text | 02-14 19:41:59.936 5932 5932 W dalvikvm: method Lcom/kakao/talk/widget/DividedLinearLayout;.drawDividersHorizontal incorrectly overrides package-private method with same name in Landroid/widget/LinearLayout; |
| Secret Chat (2) Text | 02-14 19:42:00.646 5932 5932 W dalvikvm: method Lcom/kakao/talk/widget/IcsLinearLayout;.drawDividersHorizontal incorrectly overrides package-private method with same name in Landroid/widget/LinearLayout; |

TABLE IV: Directories of artifacts on social messenger

| Messenger | Activity | Location to Retrieve |
|---|---|---|
| Telegram | Normal Chat | `\data\data\org.telegram.messenger\files\cache4.db` |
| | Secret Chat | `\data\data\org.telegram.messenger\files\cache4.db` |
| Line | Normal Chat | `\data\data\jp.naver.line.android\databases\naver_line` |
| | Hidden Chat | `\data\data\jp.naver.line.android\databases\naver_line` |
| KakaoTalk | Normal Chat | `\data\data\com.kakao.talk\databases\KakaoTalk.db` |
| | Secret Chat | `\data\data\com.kakao.talk\databases\KakaoTalk.db` |

TABLE V: Summary of social messenger chat forensics

| Messenger | Activity | Online Forensics (Live Log & Captured Packet) | Offline Forensics (Databases Artifacts) |
|---|---|---|---|
| Telegram | Normal Chat | Encrypted with SSL | No Encryption and easy to find |
| | Secret Chat | Encrypted with SSL | No Encryption and easy to find |
| Line | Normal Chat | Encrypted with SSL | Some tables are encrypted but easy to find |
| | Hidden Chat | Encrypted with SSL | Some tables are encrypted but easy to find (artifacts acquired before self-destruct activated) |
| KakaoTalk | Normal Chat | Encrypted with TLSv1 | Encrypted, except the last message |
| | Secret Chat | Encrypted with TLSv1 | Encrypted, except the last message |

## V. Conclusion

Based on the six investigation scenarios above, we have successfully presented digital forensics investigations of normal chat and private chat in Telegram, Line and KakaoTalk social messenger applications on Android devices. We show the important correlation between the chat, database files, log and captured packet that can be used as digital evidence. We also provide guidance for digital forensics analysts and investigators that can be used while dealing with cybercrime cases related to these three social messenger applications. In future work, we will investigate deleted chat recovery, decryption of encrypted chat, and memory forensics in smartphones.

## Acknowledgment

This work was supported by the Brain Korea 21 Plus Project in 2016.

## References

[1] Internet World Stats. Leading social networks worldwide as of january 2016, ranked by number of active users (in millions). http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Accessed: 2016-02-01.

[2] Noora Al Mutawa, Ibrahim Baggili, and Andrew Marrington. Forensic analysis of social networking applications on mobile devices. *Digital Investigation*, 9:S24–S33, 2012.

[3] G.B. Satrya and S.Y. Shin. Optimizing rule on open source firewall using content and pcre combination. *Journal of Advances in Computer Networks*, 3(3):308–314, 2015.

[4] Gandeva B. Satrya, Niken D.W. Cahyani, and Ritchie F. Andreta. The detection of 8 type malware botnet using hybrid malware analysis in executable file windows operating systems. In *Proceedings of the 17th International Conference on Electronic Commerce 2015*, ICEC '15, pages 5:1–5:4, New York, NY, USA, 2015. ACM.

[5] Andrew Hoog. *Android forensics: investigation, analysis and mobile security for Google Android*. Elsevier, 2011.

[6] Mohammad Iftekhar Husain and Ramalingam Sridhar. iforensics: forensic analysis of instant messaging on smartphones. In *Digital forensics and cyber crime*, pages 9–18. Springer, 2009.

[7] Aditya Mahajan, MS Dahiya, and HP Sanghvi. Forensic analysis of instant messenger applications on android devices. *arXiv preprint arXiv:1304.4915*, 2013.

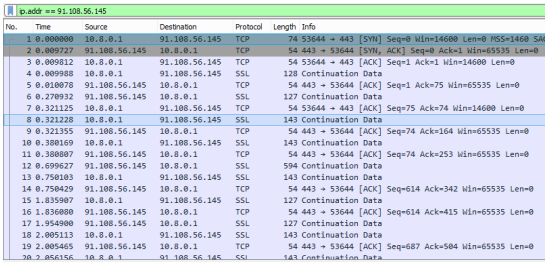
[8] Cosimo Anglano. Forensic analysis of whatsapp messenger on android smartphones. *Digital Investigation*, 11(3):201 – 213, 2014. Special Issue: Embedded Forensics.

[9] Rodney McKemmish. *What is forensic computing?* Australian Institute of Criminology Canberra, 1999.

[10] Council of Europe. Convention on cybercrime. *European Treaty Series*, 23, November 2001.

[11] National Law Information Center. Hyeongbeob [criminal act]. http://www.law.go.kr/lsEfInfoP.do?lsiSeq=178999#. Accessed: 2016-01-01.

## APPENDIX A
## CAPTURED PACKET OF TELEGRAM MESSENGER

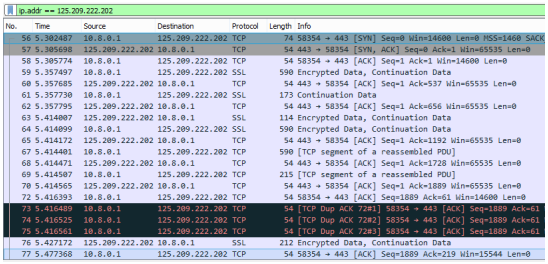


ip.addr == 91.108.56.145

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 10.8.0.1 | 91.108.56.145 | TCP | 74 | 53644 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA |
| 2 | 0.009727 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 |
| 3 | 0.009812 | 10.8.0.1 | 91.108.56.145 | TCP | 54 | 53644 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0 |
| 4 | 0.009988 | 10.8.0.1 | 91.108.56.145 | SSL | 128 | Continuation Data |
| 5 | 0.010078 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=1 Ack=75 Win=65535 Len=0 |
| 6 | 0.270932 | 91.108.56.145 | 10.8.0.1 | SSL | 127 | Continuation Data |
| 7 | 0.321125 | 10.8.0.1 | 91.108.56.145 | TCP | 54 | 53644 → 443 [ACK] Seq=75 Ack=74 Win=14600 Len=0 |
| 8 | 0.321228 | 10.8.0.1 | 91.108.56.145 | SSL | 143 | Continuation Data |
| 9 | 0.321355 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=74 Ack=164 Win=65535 Len=0 |
| 10 | 0.380169 | 10.8.0.1 | 91.108.56.145 | SSL | 143 | Continuation Data |
| 11 | 0.380807 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=74 Ack=253 Win=65535 Len=0 |
| 12 | 0.699627 | 91.108.56.145 | 10.8.0.1 | SSL | 594 | Continuation Data |
| 13 | 0.750103 | 10.8.0.1 | 91.108.56.145 | SSL | 143 | Continuation Data |
| 14 | 0.750429 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=614 Ack=342 Win=65535 Len=0 |
| 15 | 1.835907 | 10.8.0.1 | 91.108.56.145 | SSL | 127 | Continuation Data |
| 16 | 1.836080 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=614 Ack=415 Win=65535 Len=0 |
| 17 | 1.954900 | 91.108.56.145 | 10.8.0.1 | SSL | 127 | Continuation Data |
| 18 | 2.005113 | 10.8.0.1 | 91.108.56.145 | SSL | 143 | Continuation Data |
| 19 | 2.005465 | 91.108.56.145 | 10.8.0.1 | TCP | 54 | 443 → 53644 [ACK] Seq=687 Ack=504 Win=65535 Len=0 |
| 20 | 2.056156 | 10.8.0.1 | 91.108.56.145 | SSL | 143 | Continuation Data |

Fig. 7: Telegram Captured Packet

## APPENDIX B
## CAPTURED PACKET OF LINE MESSENGER

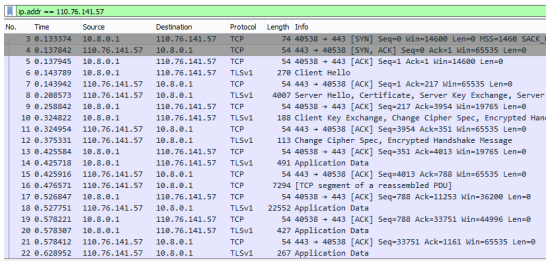


ip.addr == 125.209.222.202

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 56 | 5.302487 | 10.8.0.1 | 125.209.222.202 | TCP | 74 | 58354 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK |
| 57 | 5.305698 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 |
| 58 | 5.305774 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | 58354 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0 |
| 59 | 5.357497 | 10.8.0.1 | 125.209.222.202 | SSL | 590 | Encrypted Data, Continuation Data |
| 60 | 5.357685 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [ACK] Seq=1 Ack=537 Win=65535 Len=0 |
| 61 | 5.357730 | 10.8.0.1 | 125.209.222.202 | SSL | 173 | Continuation Data |
| 62 | 5.357795 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [ACK] Seq=1 Ack=656 Win=65535 Len=0 |
| 63 | 5.414007 | 125.209.222.202 | 10.8.0.1 | SSL | 114 | Encrypted Data, Continuation Data |
| 64 | 5.414099 | 10.8.0.1 | 125.209.222.202 | SSL | 590 | Encrypted Data, Continuation Data |
| 65 | 5.414172 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [ACK] Seq=1 Ack=1192 Win=65535 Len=0 |
| 67 | 5.414401 | 10.8.0.1 | 125.209.222.202 | TCP | 590 | [TCP segment of a reassembled PDU] |
| 68 | 5.414471 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [ACK] Seq=1 Ack=1728 Win=65535 Len=0 |
| 69 | 5.414587 | 10.8.0.1 | 125.209.222.202 | TCP | 215 | [TCP segment of a reassembled PDU] |
| 70 | 5.414565 | 125.209.222.202 | 10.8.0.1 | TCP | 54 | 443 → 58354 [ACK] Seq=1 Ack=1889 Win=65535 Len=0 |
| 72 | 5.416393 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | 58354 → 443 [ACK] Seq=1889 Ack=61 Win=14600 Len=0 |
| 73 | 5.416489 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | [TCP Dup ACK 72#1] 58354 → 443 [ACK] Seq=1889 Ack=61 |
| 74 | 5.416525 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | [TCP Dup ACK 72#2] 58354 → 443 [ACK] Seq=1889 Ack=61 |
| 75 | 5.416561 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | [TCP Dup ACK 72#3] 58354 → 443 [ACK] Seq=1889 Ack=61 |
| 76 | 5.427172 | 125.209.222.202 | 10.8.0.1 | SSL | 212 | Encrypted Data, Continuation Data |
| 77 | 5.477368 | 10.8.0.1 | 125.209.222.202 | TCP | 54 | 58354 → 443 [ACK] Seq=1889 Ack=219 Win=15544 Len=0 |

Fig. 8: Line Captured Packet

## APPENDIX C
## CAPTURED PACKET OF KAKAOTALK MESSENGER



ip.addr == 110.76.141.57

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 3 | 0.133374 | 10.8.0.1 | 110.76.141.57 | TCP | 74 | 40538 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_ |
| 4 | 0.137842 | 110.76.141.57 | 10.8.0.1 | TCP | 54 | 443 → 40538 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 |
| 5 | 0.137945 | 10.8.0.1 | 110.76.141.57 | TCP | 54 | 40538 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0 |
| 6 | 0.143789 | 10.8.0.1 | 110.76.141.57 | TLSv1 | 270 | Client Hello |
| 7 | 0.143942 | 110.76.141.57 | 10.8.0.1 | TCP | 54 | 443 → 40538 [ACK] Seq=1 Ack=217 Win=65535 Len=0 |
| 8 | 0.208573 | 110.76.141.57 | 10.8.0.1 | TLSv1 | 4007 | Server Hello, Certificate, Server Key Exchange, Server |
| 9 | 0.258842 | 10.8.0.1 | 110.76.141.57 | TCP | 54 | 40538 → 443 [ACK] Seq=217 Ack=3954 Win=19765 Len=0 |
| 10 | 0.324822 | 10.8.0.1 | 110.76.141.57 | TLSv1 | 188 | Client Key Exchange, Change Cipher Spec, Encrypted Han |
| 11 | 0.324954 | 110.76.141.57 | 10.8.0.1 | TCP | 54 | 443 → 40538 [ACK] Seq=3954 Ack=351 Win=65535 Len=0 |
| 12 | 0.375331 | 110.76.141.57 | 10.8.0.1 | TLSv1 | 113 | Change Cipher Spec, Encrypted Handshake Message |
| 13 | 0.425584 | 10.8.0.1 | 110.76.141.57 | TCP | 54 | 40538 → 443 [ACK] Seq=351 Ack=4013 Win=19765 Len=0 |
| 14 | 0.425718 | 10.8.0.1 | 110.76.141.57 | TLSv1 | 491 | Application Data |
| 15 | 0.425916 | 110.76.141.57 | 10.8.0.1 | TCP | 54 | 443 → 40538 [ACK] Seq=4013 Ack=788 Win=65535 Len=0 |
| 16 | 0.476571 | 110.76.141.57 | 10.8.0.1 | TCP | 7294 | [TCP segment of a reassembled PDU] |
| 17 | 0.526847 | 10.8.0.1 | 110.76.141.57 | TCP | 54 | 40538 → 443 [ACK] Seq=788 Ack=11253 Win=36200 Len=0 |
| 18 | 0.527751 | 110.76.141.57 | 10.8.0.1 | TLSv1 | 22552 | Application Data |
| 19 | 0.578221 | 10.8.0.1 | 110.76.141.57 | TCP | 54 | 40538 → 443 [ACK] Seq=788 Ack=33751 Win=44996 Len=0 |
| 20 | 0.578307 | 10.8.0.1 | 110.76.141.57 | TLSv1 | 427 | Application Data |
| 21 | 0.578412 | 110.76.141.57 | 10.8.0.1 | TCP | 54 | 443 → 40538 [ACK] Seq=33751 Ack=1161 Win=65535 Len=0 |
| 22 | 0.628952 | 110.76.141.57 | 10.8.0.1 | TLSv1 | 267 | Application Data |

Fig. 9: KakaoTalk Captured Packet