Forensic Science International: Digital Investigation 40 (2022) 301347

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

Forensic analysis of instant messengers: Decrypt Signal, Wickr, and Threema

Jihun Son [a], Yeong Woong Kim [a], Dong Bin Oh [b], Kyounggon Kim [c, *]

[a] International Cybercrime Research Center, Korean National Police University, 100-50 Hwangsan-gil, Shinchang-myeon, Asan-si, 31539, Chungcheongnam-do, South Korea
[b] School of Cyber Security, Korea University, 13, Jongam-ro, Seongbuk-gu, 02841, Seoul, South Korea
[c] Center of Excellence in Cybercrime and Digital Forensics, Naif Arab University for Security Science, Khurais Rd, Ar Rimayah, Riyadh, 14812, Saudi Arabia

ARTICLE INFO

Article history:
Received 30 August 2021
Received in revised form
10 January 2022
Accepted 10 January 2022
Available online 31 January 2022

Keywords:
Instant Messenger
Signal
Wickr
Threema
Forensics

ABSTRACT

As organized criminals use instant messengers, it becomes increasingly important to obtain digital evidence from instant messengers. Recently, instant messengers apply end-to-end encryption, so all digital evidence can only be obtained from your mobile device. However, some instant messengers encrypt and store database and multimedia files, making forensic analysis of mobile devices difficult. In this paper, we present a methodology for analyzing the decryption algorithm of the messenger, and apply this methodology to Signal, Wickr, and Threema. We extracted data from both unrooted and rooted devices and performed static and dynamic analysis. As a result, we succeeded in decrypting all the encrypted database, multimedia, log, and preferences files of three messengers. We describe the decryption algorithms and disclose all decryption scripts.

1. Introduction

Starting with AOL Instant Messenger (AIM) in 1997, instant messenger has become one of the most popular applications in the world. WhatsApp, the most representative instant messenger, has approximately 2 billion active users, while Facebook Messenger and WeChat have 1.3 billion and 1.2 billion active users, respectively (Statista, 2021). Organized criminals also use instant messengers, mainly to exchange information about drugs, weapons, and child pornography. The FBI operated fake encrypted messengers to trick organized criminals, arresting more than 800 suspects and confiscating drugs and weapons (BBC, 2021). Therefore, from a digital forensic point of view, messenger analysis is becoming increasingly important.

Recently, instant messengers have introduced end-to-end encryption technology to combat privacy breaches such as mass surveillance activities by intelligence agencies (Rösler et al., 2018). If end-to-end encryption is applied to the messenger, the conversation history is not stored on the server, so it becomes important to obtain evidence from the local device. However, some instant messengers (e.g., Signal) apply encryption to databases and multimedia files, making it impossible to decrypt without knowing the decryption algorithm. Another security feature of these messengers, messenger lock feature, also makes it difficult to collect evidence from devices. All these security features of instant messenger stand as a barrier to forensic analysts.

In this paper, we analyzed the decryption algorithms of Signal, Wickr and Threema messengers on Android devices. All of these messengers support end-to-end encryption by default. They encrypt both database and multimedia files. They also have the messenger lock feature, which allows users to lock the messenger with a password. Our goal is to decrypt all encrypted files of Signal, Wickr, and Threema. If possible, we tried to decrypt without the user's password, even if the messenger lock feature was enabled.

The contributions of this paper are as follows.

- We presented a methodology for analyzing the decryption algorithm of instant messenger. We extracted data from both unrooted and rooted devices and performed static and dynamic analysis on messenger applications.
- We decrypted all encrypted files of Signal, Wickr, and Threema. Compared to previous studies, our study found a new

* Corresponding author.
E-mail address: kkim@nauss.edu.sa (K. Kim).

https://doi.org/10.1016/j.fsidi.2022.301347
2666-2817/

decryption algorithm, expanded the range of decryptable files, and corrected outdated parameters.
- We described the decryption algorithms in detail and have released all decryption scripts through *GitHub*.[1]

The remainder of this paper is structured as follows. Section 2 introduces existing research. Section 3 presented a forensic analysis methodology for instant messenger. Section 4 applies the analysis methodology to messengers and details the algorithms and parameters that we found. Section 5 provides evaluations and limitations of the results. Finally, Section 6 provides conclusions of the study.

2. Related work

Many studies conducted forensic analysis on instant messenger. Husain and Sridhar (2009) analyzed AIM, Yahoo! Messenger, and Google Talk messenger in the iOS devices and found *Conversation log*, *Buddy list*, and *Plaint text password* in some messengers. Mahajan et al. (2013) investigated artifacts according to user activities such as login and chat reception in WhatsApp and Viber messenger on Android devices. Satrya et al. (2016) provided analysis guidance such as logcat, packet capture, and database artifact analysis for the private chat feature of Telegram, Line, and Kakaotalk messenger on Android devices. Chang and Chang (2019) set different configuration environments for Line Messenger in Windows 10 and analyzed artifacts, but could not decrypt the encrypted database. These studies analyzed messenger artifacts, but unencrypted artifacts were analyzed. Even when encrypted files existed, these studies did not try to analyze the decryption algorithm.

Many studies have been conducted to decrypt encrypted data stored by instant messengers. Cortjens et al. (2011) found that the WhatsApp database file is always encrypted using the same key in the Android WhatsApp application. Anglano et al. (2016) founded an algorithm to decrypt the database using the passphrase set by the user and found a method to retrieve the password from volatile memory in the ChatSecure application on Android devices. Wu et al. (2017) described how to decrypt WeChat's encrypted database. Choi et al. (2019) found that encrypted databases of KaKaoTalk and NateOn in Windows environment can be decrypted without a user password, and that the database key of QQ messenger is stored in an external server. Kim et al. (2020) revealed the decryption algorithm of the database of telegram X and BBM-Enterprise messenger in Windows 10, MacOS mojave, Android, and iOS.

There are several studies on Signal and Wickr, which are the subject of this paper. Azhar et al. (2020) analyzed the Signal messenger in the rooted Android phone using Oxygen Forensic for Android, but they did not obtain any data on conversations and account information. Kaczyński (2019) analyzed database protection mechanisms of Signal, but it is insufficient to be used for forensics; The author has not even found the database decryption algorithm. Barton and Azhar (2016), and Azhar and Barton (2017) discovered the encrypted database in the Wickr messenger and found that internal files with the '*.wic*' extension were used for encryption but could not success to decrypt the database file. Kim et al. (2021) discovered Wickr's database and multimedia file decryption algorithm on Android and iOS devices. However, we found that some parameters became out of date as the Wickr version was upgraded. In addition, The method presented in that paper could not decrypt files without the user's password.

[1] https://github.com/hunjison/Messenger-Forensics.

Meanwhile, there have been attempts to automate the analysis of database decryption. Zhang et al. (2020) has automated the analysis of the database encryption method by applying the probability model to Amandroid, which is Android application analysis framework for analyzing data flows. However, this study does not appear to be effective for analyzing our messengers. In the case of multimedia file and log file encryption, unlike the case of database encryption, the function used for encryption and the path of the encrypted file are not predictable. In order to use the suggested search algorithm, manual analysis is required to obtain the necessary information. In addition, it is impossible to analyze Wickr because Wickr use its own native library for all encryption. Finally, considering that the source code was not disclosed in this study, and all source code of Signal and Threema are disclosed, we decided that manual analysis would be more efficient.

3. Analysis methodology

We performed forensic analysis on Signal, Wickr, and Threema messengers on the Android devices. The criteria for selecting messengers are as follows. All of these messengers default to end-to-end encryption and encrypt not only databases but also multimedia files such as images, videos, and documents. They also support the messenger lock feature that requires password or fingerprint authentication when opening the app. Our goal is to decrypt all encrypted files of Signal, Wickr, and Threema. If possible, we tried to decrypt without the user's password, even when the user enabled the messenger lock feature. This section introduces our method for systematic analysis.

3.1. Data acquisition

Files created by messengers are saved in Internal storage and External storage of the Android device. The path of Internal storage is *'/data/data/<PACKAGE>'*, *<PACKAGE>* means the package name of the messenger app. The path of External storage is *'/storage/emulated/0/'*, and the path may change depending on the device. We will call the path of Internal storage 'INTERNAL' and the path of External Storage 'EXTERNAL'.

We tried to acquire data from both rooted and unrooted devices. We were able to acquire INTERNAL and EXTERNAL data through Android Debug Bridge (ADB) on the rooted device. On the unrooted device, however, we could only get EXTERNAL data via ADB. We tried several methods to acquire INTERNAL data from unrooted devices. We tried to back up the device using ADB Backup, Smart Switch, and Google Drive mobile Backup, but it failed because the backup files did not include our messenger's INTERNAL data (Some INTERNAL data from other apps were included). We also tried to root the device to extract data from unrooted device. Almehmadi and Batarfi (2019); Boueiz (2020) have succeeded in one-click rooting using applications such as KingoRoot and software such as Dr.Fone, and also succeeded in rooting by installing a custom ROM using a custom recovery image such as TWRP. However, on the devices used in our experiments (see the device specifications below), all these methods failed except the method using Magisk. Magisk is a suite of open source software for customizing Android, which has a feature that provides root access to user. Magisk requires a full data wipe to root Samsung devices with Android 9.0 or higher (Installation of Magisk). Data cannot be extracted because all data is wiped during the rooting process. Therefore, our attempt to extract data by rooting the device failed.

We found a way to acquire INTERNAL data through the "Messenger Backup Migration". Signal, Wickr, and Threema both have the messenger backup function, which transfers all conversations from the old device to the new device. To acquire INTERNAL

data using the backup function, we need 1 unrooted device and 1 rooted device: the unrooted device contains the data we want to analyze, and the rooted device will contain the backup data. We create a backup file in the messenger on the unrooted device and move this file to the rooted device. Then, we install the messenger application on the rooted device and restore the backup file to the application. We can obtain INTERNAL data from the rooted device to PC through ADB. Fig. 1 shows the process of restoring the INTERNAL data.

For analysis, we used two unrooted and three rooted devices, unrooted are Galaxy S10 5G with Android OS 9 and Galaxy A20e with Android OS 10, rooted are Galaxy A20e with Android OS 11 and Nox emulators with Android OS 7 and 9. We assumed that we knew the Android device's lock password (fingerprint or PIN). The instant messenger that we analyzed were Signal v5.19.4, Wickr Me v5.84.6, Threema v4.55 downloaded from the Google Play Store.

3.2. Data listing

We have collected and listed all encrypted files such as databases, multimedia files, log files, and configuration files. We tried to get as much hint as possible from the extracted data to facilitate the analysis of the next step. In all 3 messengers, database files were located in the INTERNAL/*databases* folder. Multimedia files were located in the INTERNAL or EXTERNAL, but it was not difficult to find them because of the large file size. Binary data that appeared to be used for decryption existed as XML files in the INTERNAL/*shared_prefs* folder or as binary files in the INTERNAL/*files* folder. We opened and checked the contents of the collected files and excluded empty or unrelated files from the list. We tried to list the original file names that we found, and if there were multiple files with similar format, only one was listed with asterisk (*). In addition, we observed file changes according to user actions (e.g., sending or receiving messages, reading messages) or user settings (e.g., messenger lock).

3.3. Decryption process analysis

We used static analysis and dynamic analysis to find the decryption flow of the messenger. For static analysis, Android Studio was used when the messenger was open source, and JEB and IDA were used when the messenger was not open source. Based on the information obtained in the previous step, we could set the starting point of static analysis. For dynamic analysis, Frida was used. Dynamic analysis has been very helpful in reducing the analysis time because we can check the parameters and return values of functions directly. We used Android Studio v4.2.2, Frida v15.0.6, IDA Pro v7.2, and JEB Decompiler 3.0.0 for analysis.

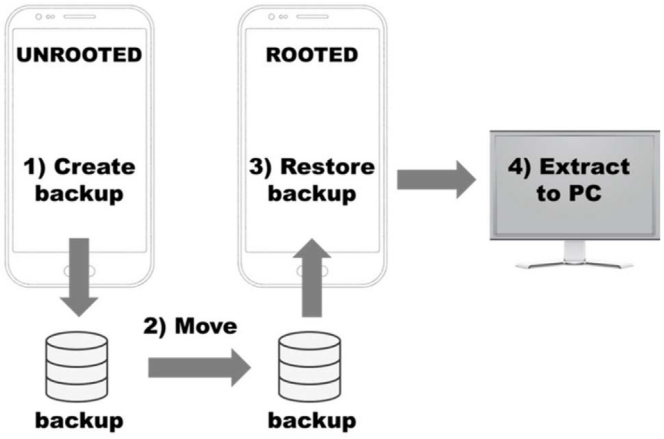

Fig. 1. Messenger Backup Migration from unrooted device.

3.4. Verification

We wrote a decryption script based on the analysis results. We verified the script by opening the database with the derived key and decrypting the multimedia, log, and preferences files. SQLCipher, a database encryption module used by all three messengers, requires *PRAGMA* values for decryption. We verified the *PRAGMA* values using DB Browser for SQLite v3.12.2. We also checked whether the decrypted multimedia files were exactly the same as the original files. In the case of log and preference files, We checked whether the files were successfully decrypted and listed the information that we can extract from these files.

In the next chapter, we conduct a case study on three messengers using the analysis methodology presented here.

4. Messenger analysis

4.1. Case 1: Signal

Signal is a free, open source messenger developed by the *Signal Technology Foundation*. The Electronic Frontier Foundation (EFF) includes Signal in its Surveillance Self-Defense Guide. Twitter CEO Jack Dorsey in 2020 and SpaceX and Tesla CEO Elon Musk in 2021, respectively, mentioned Signal on Twitter, resulting in a surge in downloads. Signal has more than 50 million downloads on the Google Play Store. Signal supports Group chats, as well as group voice and video calls, and uses the Signal protocol to perform end-to-end encryption for all communications (Signal homepage). Signal uses SQLCipher to encrypt SQLite database, and stores file attachments and media as encrypted blobs within the application sandbox (Signal-Blog, 2020). Signal supports the messenger lock feature, which allows users to enter the password, pin, or fingerprint used by the Android device when opening the messenger.

4.1.1. Data listing

The package name of Signal is *'org.thoughtcrime.securesms'*. We listed files that we found from the messenger's internal and external storage in the Table 1. There were four SQLCipher-encrypted database files, and *'.db-shm'* and *'.db-wal'* file also existed for each database. Multimedia files were encrypted and stored in the *'app_parts'* folder. Cache files were also encrypted and stored in the *'cache/image_manager_disk_cache'* folder. Multimedia files are created when we send or receive them, while cache files are created when they are displayed on the screen. *'journal'* file contains logs of cache files. Log files were encrypted, divided into several files. In some cases, log files existed as a database, *'signal-logs.db'*. We found several binary data that seemed to be used for decryption inside *'files'* folder and *'shared_prefs'* folder. There was a small change in the files depending on whether the messenger lock feature was used, but it was not related to the decryption.

4.1.2. Decryption process analysis

We describe Signal's step-by-step decryption process in Table 2.

4.1.2.1. Database decryption (Step1). Signal uses the Android Keystore to protect the encryption key. Android Keystore supports *Extraction prevention*, which ensures keys come out of secure hardware and prevents keys from entering the application process. So we cannot extract the keys directly from the Android Keystore. Sabt and Traoré (2016) studied the threat model of attacking the Android Keystore through a malicious application. Κασαγί αννης

Table 1
Data listing of signal.

File	Path	File name
databases	INTERNAL/databases/	signal-jobmanager.db signal-key-value.db signal-megaphone.db signal-logs.db (do not always exist) signal.db
files	INTERNAL/files/	*PersistedInstallation.W0RFRk ... lmMjY2.json
multimedia	INTERNAL/app_parts/	*part2285170834062364792.mms
preferences	INTERNAL/shared_prefs/	SecureSMS-Preferences.xml org.thoughtcrime.securesms_preferences.xml
cache	INTERNAL/cache/image_manager_disk_cache/	*13ff1b ... f3cf51.0 journal
	INTERNAL/cache/log/	*log-1626760079310
backup	EXTERNAL/Backups	*signal-2021-07-27-18-55-04.backup

Table 2
Signal's decryption process.

Step	Function	Algorithm	Parameter
1	Key generation (database, multimedia, log)	AES256-GCM-DECRYPT(c, k, i)	c: ciphertext ‖ authentication tag(both from XML file) k: key from Android Keystore i: iv from XML file
2	Key generation for multimedia decryption	HMAC-SHA256(k, d, l)	k: key from Step1 (*modernKey*) d: data_random from database l: length (fixed to 32)
3	Multimedia decryption	AES256-CTR-DECRYPT(c, k, n)	c: ciphertext from .mms file k: key from Step2 n: nonce (fixed to 0)
4	Log decryption	AES256-CBC-DECRYPT(c, k, i)	c: ciphertext from log file k: key from Step1 (*LogKey*) i: iv from log file

(2018) studied how to import key pairs used in other apps through a rogue app. With reference to these two studies, we developed an app that can steal the key of Signal's Android Keystore. Because of *Extraction prevention*, we designed the app to do Step1 decryption of Table 2 with that key as we cannot extract the key directly from Android Keystore.

The encrypted database name of the Signal is *'signal.db'*. To generate the database decryption key, we need to obtain a *'pref_database_encrypted_secret'* value from *'org.thoughtcrime.securesms_preferences.xml'* file. This value is a JSON string with the *'data'*, *'iv'* as a key, and a base64-encoded data as a value. The *'data'* value contains the ciphertext and the authentication tag of AES256-GCM, and the *'iv'* value contains the initial vector (IV). The key for database decryption is obtained from Android Keystore as mentioned above. We can obtain the database decryption key by performing AES256-GCM decryption. This decryption process is the same for multimedia decryption and log decryption. We developed an app to extract the key from the Android Keystore and perform decryption as shown in Fig. 2.

When the Android API level is lower than 23, we can get the database key from *'pref_database_unencrypted_secret'*. Since this value is not encrypted, it can be used directly as a database key.

4.1.2.2. Multimedia decryption (Step2, Step3). To get the multimedia decryption key, we need a *modernKey* and a *data_random* value. We can get the base64 encoded *modernKey* by performing Step1 decryption on the *'pref_attachment_encrypted_secret'* value as shown in Fig. 3. We can get the *data_random* value from inside the decrypted database. Using *modernKey* as a key and *data_random* value as data, performing the HMAC-SHA256 operation, we can calculate the multimedia decryption key. Using this key, we can obtain the original multimedia file by decrypting the *'.mms'* file with AES256-CTR. The initial vector(IV) is fixed to 0.

4.1.2.3. Log decryption (Step4). We can get the log decryption key by performing Step1 decryption on *'pref_log_encrypted_secret'*. The encrypted log file has a structure in which iv, length, and ciphertext are repeated as shown in Fig. 4. We can obtain the original log files by repeatedly performing AES256-CBC decryption on each encrypted log file. When the log file exists as a database, we can use the database key obtained in Step1.

4.1.3. Verification

Signal's database key is a 64-length hex string converted from a 32-byte byte array. Through static code analysis, we could find the *PRAGMA* value of SQLCipher.

- *PRAGMA cipher_default_page_size = 4096*
- *PRAGMA cipher_default_kdf_iter = 1*
- *PRAGMA cipher_hmac_algorithm = HMAC_SHA1*
- *PRAGMA cipher_kdf_algorithm = PBKDF2_HMAC_SHA1*

Using these *PRAGMA* values, we could decrypt the database as shown in Fig. 5. In the *'sms'* table, we could found text data sent and received by the user. In the *'part'* table, we could find information related multimedia files.

In the *'part'* table of the decrypted database, one row contains data for one multimedia file. We can get the file path from the *'_data'* field, the original file name from the *'file_name'* field, and the *data_random* from the *'data_random'* field. As a result of decrypting the multimedia files, we found that the decrypted files are identical to the original files except for the image files. Image files appear to be compressed during transmission. Fig. 6 shows the result of

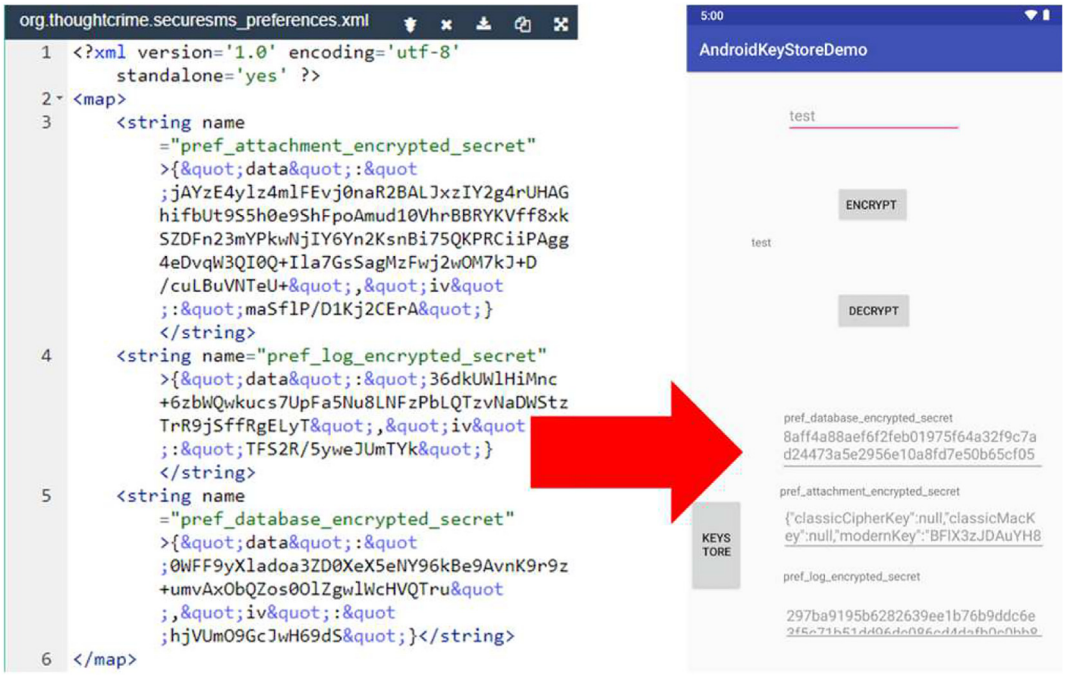

Fig. 2. Extract Android Keystore by developing an application.

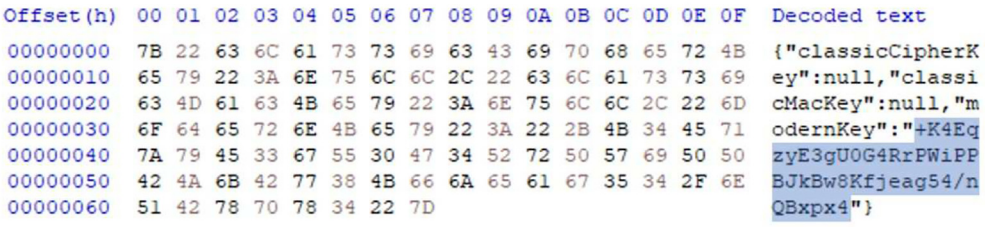

Fig. 3. How to obtain *modernKey*

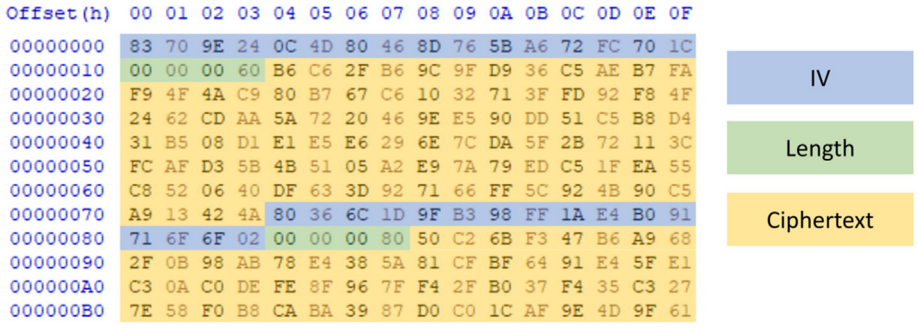

Fig. 4. A Structure of Signal's encrypted log files.

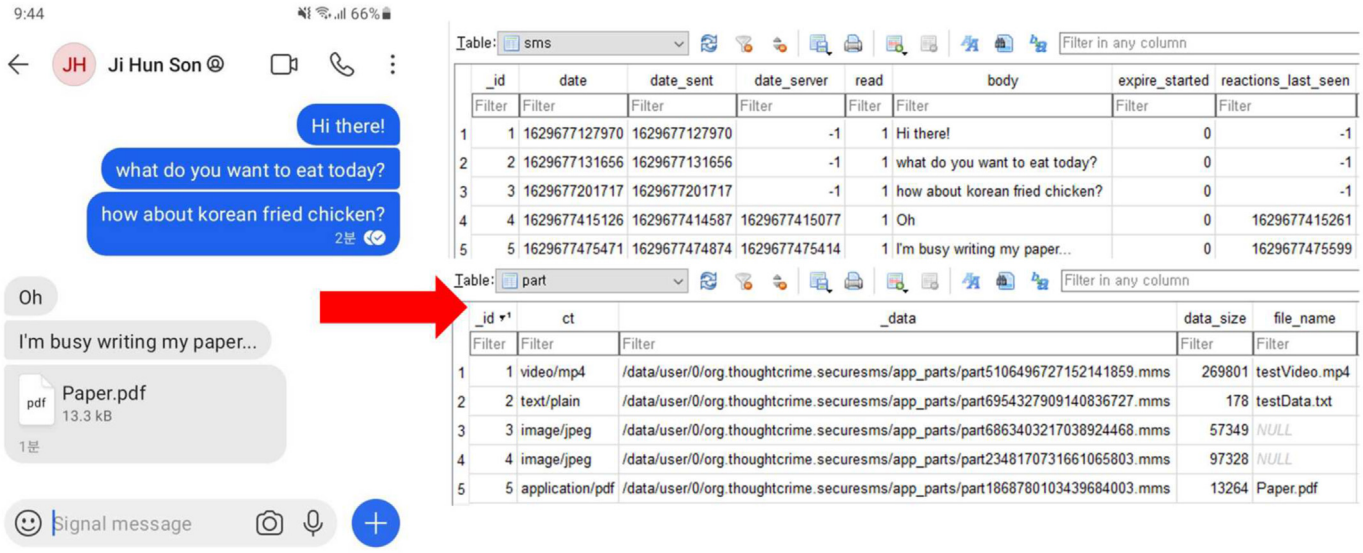

Fig. 5. Signal's database decryption.

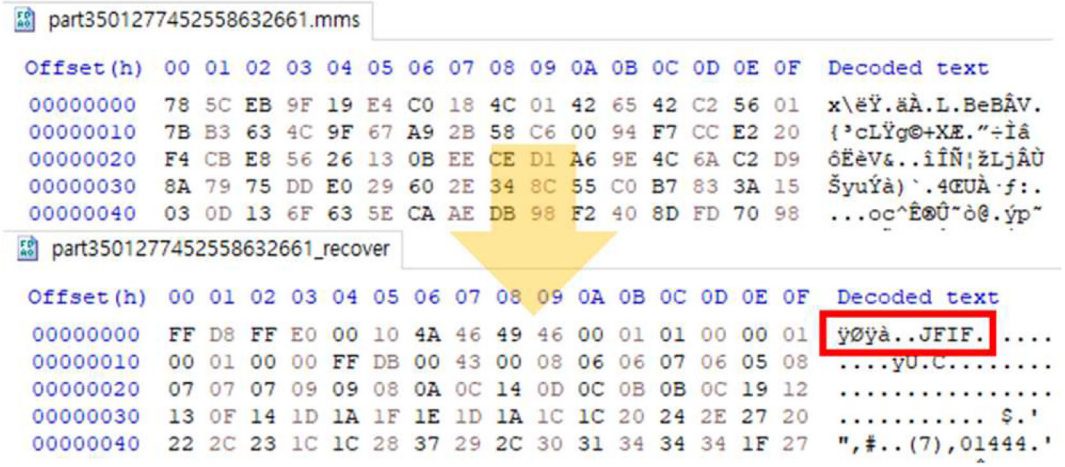

Fig. 6. Signal's multimedia file decryption.

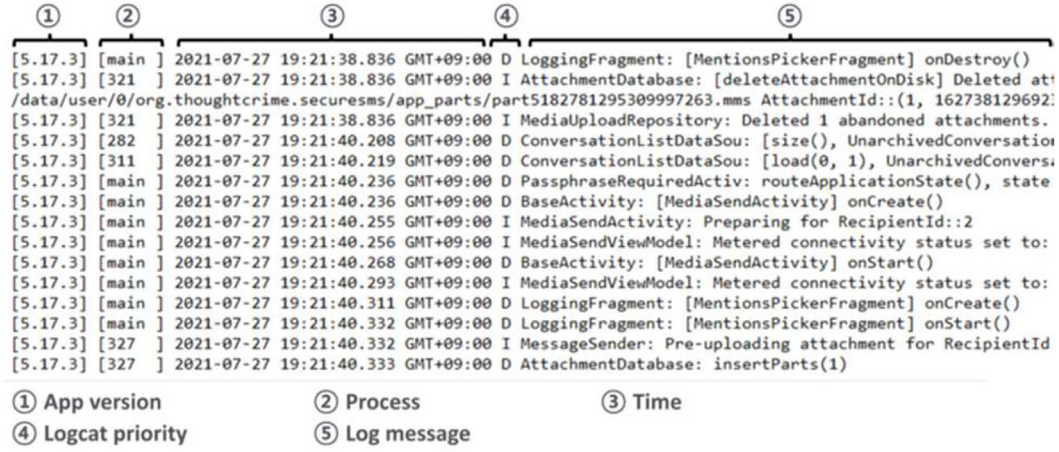

Fig. 7. A structure of Signal's decrypted log file.

successful decryption of multimedia files.

We also succeeded in decrypting log files. Signal's log files contain information on the user's time and behavior of using Signal, which is important for tracking the user's behavior. Fig. 7 shows the structure of the decrypted log files.

4.2. Case 2: Wickr

Wickr is a free instant messenger developed by *Wickr Inc*. Wickr met all security standards set forth by the National Security Agency in 2020. Recently, *Wickr Inc* was acquired by Amazon in June 2021. Although Wickr has many releases, Wickr Me, Wickr Pro, Wickr RAM, and Wickr Enterprise, Kim et al. (2021) have shown that there was no difference in the type and structure of data stored on local devices. In this paper, we conduct our analysis based on Wickr Me, a free messenger that people use the most. Wickr Me has been downloaded more than 5 million times on the Google Play Store. Wickr supports end-to-end encryption for all text chat, file transfer, and audio/video communications, and provides *Expiration* and *Burn-on-read* functions that automatically delete messages. Wickr supports messenger lock, which allows users to log in using their password when opening the app.

4.2.1. Data listing

The package name of Wickr is *'com.mywickr.wickr2'*, and the files that could be found from the messenger's internal and external storage were listed in Table 3. The name of the encrypted database file was *'wickr_db'*, and *'.db-shm'* and *'.db-wal'* files existed. All files in the INTERNAL/*files*/folder were encrypted. We assumed that *'.wic'* files were data used for decryption and that *'.prefs'* files were encrypted preference files. Multimedia files were also encrypted and had names in universally unique identifier (UUID) format. Multimedia files were created when we read or sent messages. Log files were saved in plain text, and their name could be created in various languages depending on the system language of the Android device. Some files existed in the cache folder and the preference folder, but they were not related to encryption or contained empty values. When the user enables the messenger lock feature, the data of the *'prefs_com.mywickr.wickr2_preferences'* and *'prefs_prefs.wic'* files are changed.

Table 3
Data listing of wickr.

File	Path	File name
databases	INTERNAL/databases/	wickr_db
files	INTERNAL/files/	kck.wic
		kcd.wic
		sk.wic
		ds.wic
		prefs_prefs.wic
		prefs_registration.prefs
		prefs_migrationPrefs.prefs
		prefs_proxyConfig.prefs
		prefs_sso.prefs
		prefs_com.mywickr.wickr2_preferences
multimedia	INTERNAL/files/enc/	*6f3533a2-5314-4006-883d-ede3507f23af
log	INTERNAL/files/logs/	*Jul_28_2021.log

4.2.2. Decryption process analysis

We describe Wickr's step-by-step decryption process in Table 4.

4.2.2.1. Database decryption (Step1, Step2, and Step3). Wickr has two login methods: *i)* password login and *ii)* automatic login. When the user sets the messenger lock feature, password login is activated. Kim et al. (2021) revealed the decryption algorithm for password login, which requires the password as a parameter. The password is set by the user in Wickr messenger, so it is difficult to obtain it without the user's cooperation. We found the decryption algorithm for automatic login. In the algorithm, the user password is not required, so we can extract keys from only the files inside the device. Moreover, even if the user changes the login method, automatic login to password login, the decryption algorithm that we found still works. This is because Wickr does not delete files used for automatic login, and because both login methods generate the same database encryption key. Automatic login is the default setting in Wickr.

Wickr stores the database key with double encryption. The key for the first decryption is *devinfo*, which is a value created from *android_id* in the Android system. The *android_id* value can be obtained by searching for the package name in the *'/data/system/users/0/settings_ssaid.xml'* file. The process to convert *android_id* to *devinfo* is as follows, and can be seen in Table 5. The *android_id* value is changed to UUID format after MD5 hashing. In the generated UUID data, offset 14 is changed to 3, and offset 19 is changed to 8, 9, a, or b according to the remainder of the dividing the value by 4. Finally, by hashing the generated UUID to SHA256, we can get the *devinfo* value. We found this process by hooking the function and experimenting with about 10000 cases, and there were no exceptions.

The first decryption to get Wickr's database key is AES256-GCM decryption with *devinfo* as key and *kcd* (data of *kcd.wic*) as data. *kcd* includes ciphertext, authentication tag, and iv, and the structure is as shown in Fig. 8. The second decryption is also AES256-GCM decryption using *kck* (data of *kck.wic*) as the key and the value derived by performing the first decryption (Step2 in Table 4) as

Table 4
Wickr's decryption process.

Step	Function	Algorithm	Parameter
1	Devinfo generation	SHA256(UUID(MD5(a))) *described in Table 5.	a: android_id
2	1st database key generation	AES256-GCM-DECRYPT(c,k,i)	c: ciphertext ‖ authentication tag(both from kcd.wic) k: devinfo from Table 5. i: iv from kcd.wic
3	2nd database key generation	AES256-GCM-DECRYPT(c,k,i)	c: ciphertext ‖ authentication tag(both from step1) k: kck from kck.wic i: iv from step1
4	Multimedia Decryption	AES256-GCM-DECRYPT(c,k,i)	c: ciphertext ‖ authentication tag(both from encrypted multimedia file) k: key from messagePayload data in database i: iv from encrypted multimedia file
5	Preferences decryption	AES256-GCM-DECRYPT(c,k,i)	c: ciphertext ‖ authentication tag(both from each pref) k: devinfo from Table 5. i: iv from each pref

Table 5
The process of calculating *devinfo*

Description	Example
Extract android_id	android-4d26601eb9bedfef
Do MD5 hash	b2792f62f9c288e9385de263989b1be8
Convert to UUID format	b2792f62-f9c2-88e9-385d-e263989b1be8
Substitute	b2792f62-f9c2-38e9-b85d-e263989b1be8
Do SHA256 hash	f645a3 ... 80f6a2

data. Since this value has the same structure as *kcd*, we can decrypt it in the same way. After performing these two decryptions, we can obtain the database key.

4.2.2.2. Multimedia decryption (Step4). The decryption algorithm for Wickr's multimedia files has already been revealed in the previous study (Kim et al., 2021). When we tested the revealed decryption algorithm, we found that it still works well.

4.2.2.3. Preferences decryption (Step5). Preferences file means 6 files with *'pref'* in the name among the files listed in Table 3. The decryption key is *devinfo* calculated in Table 5. We can get the data from each *prefs* file, and the structure of the data is the same as in Fig. 8. We can obtain the original file by performing AES256-GCM decryption on 6 preferences files.

4.2.3. Verification

The SQLCipher *PRAGMA* values of Wickr are the same as the default *PRAGMA* values of SQLCipher 4, and those values are as follows.

- *PRAGMA cipher_default_page_size = 4096*
- *PRAGMA cipher_default_kdf_iter = 256000*
- *PRAGMA cipher_hmac_algorithm = HMAC_SHA512*
- *PRAGMA cipher_kdf_algorithm = PBKDF2_HMAC_SHA512*

According to the results of the previous study (Kim et al., 2021), these values were the same as the default *PRAGMA* values of SQLCipher 3, but we confirmed that they were outdated as the app version was upgraded. Wickr's database decryption key is a 66-character hex string converted from a 33-byte byte array which is resulting from Step2 of Table 4. Using the database key and *PRAGMA* value, we can decrypt the database as shown in Fig. 9.

In the *'Wickr_Message'* table of the decrypted database, we found that each row contains information about each multimedia file. We can extract the original file name from the *'cachedText'* field, and the encrypted file name, the decryption key from the *'messagePayload'* field. We succeeded in decrypting the multimedia files as shown in Fig. 10. We have verified that all types of decrypted multimedia files are identical to the original files. We also succeeded in decrypting the preferences file. Table 6 lists information that we can obtain from preferences files.

4.3. Case 3: Threema

Threema is a paid open source instant messenger developed by *Threema GmbH*. Threema took second place among 18 messenger apps evaluated by German consumer group Stiftung Warentest in 2015 (Stiftung Warentest). Threema has its own messaging protocol like Signal and WhatsApp, and active research is being conducted on its security (Rösler et al., 2018; Rösler, 2018; Scheitle et al., 2016). Threema is mainly used in Switzerland and currently has more than 10 million users (Jungfer, 2021). Threema has more than 1 million downloads based on the Google Play Store. Threema encrypts all communications end-to-end, including group messages, photos,videos, files, and voice calls. Threema includes its own app-specific encryption based on AES-256 to protect stored messages and media. Threema supports the messenger lock feature, which is a login method using the password when opening the app.

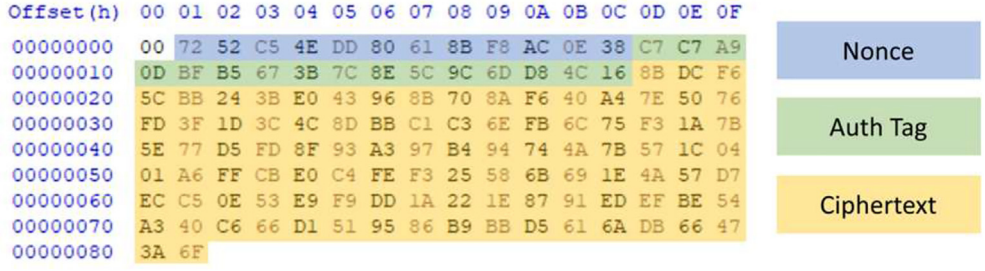

Fig. 8. The structure of kcd.wic

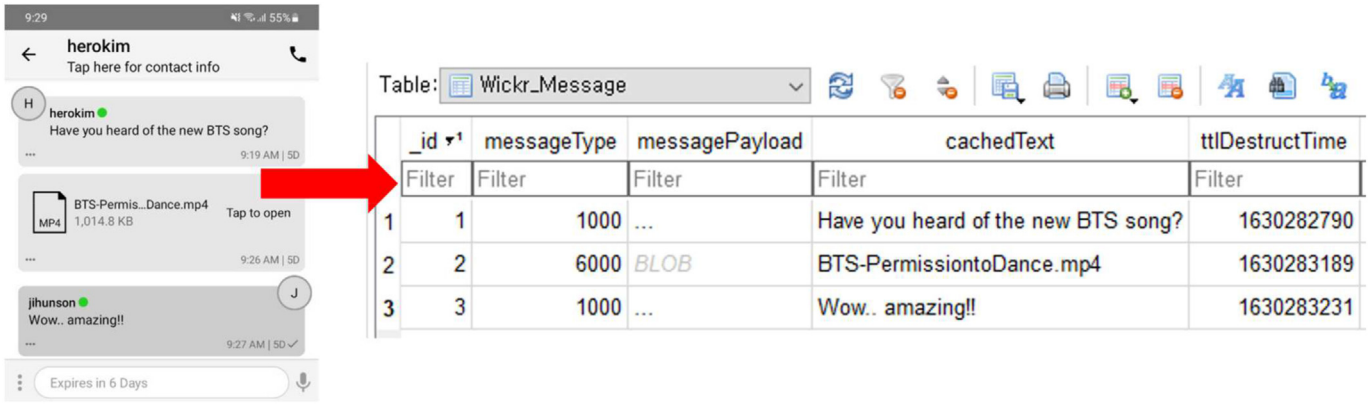

Fig. 9. Wickr's database decryption.

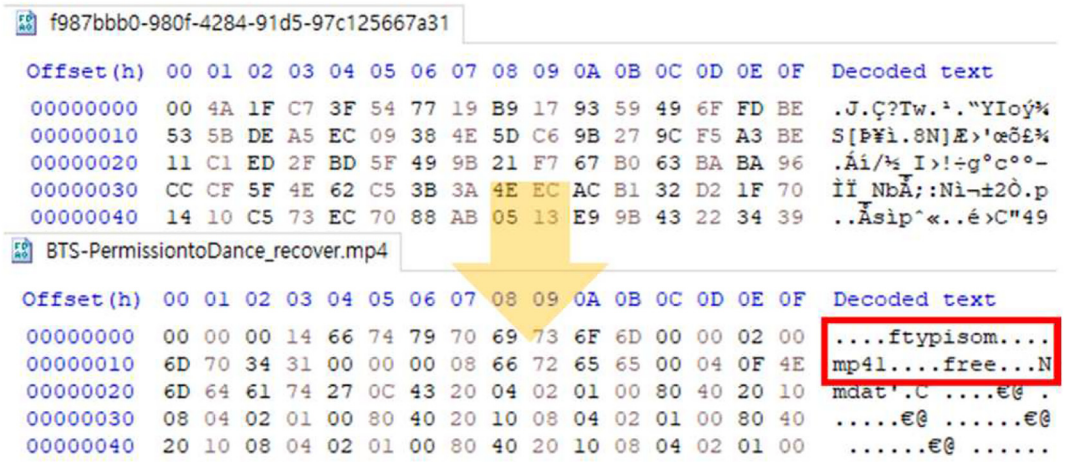

Fig. 10. Wickr's multimedia file decryption.

Table 6
Wickr's preference files.

File Name	Contents
prefs_proxyConfig.prefs	Proxy configuration and URL of remote server
prefs_prefs.wic	Name of the user account, count of force suspension, or login failure
prefs_com.mywickr.wickr2_preferences	Most settings for using the app (e.g. screenshots, previews)
prefs_registration.prefs	empty
prefs_sso.prefs	empty
prefs_migrationPrefs.prefs	empty

4.3.1. Data listing

The package name of Threema is *'ch.threema.app'*, and the files that we could find from the messenger's internal and external storage were listed in Table 7. There were two database files, and the name of the encrypted database file was *'threema4.db'*. *'.db-shm'* and *'.db-wal'* file did not exist. Another database file, *'threema-nonce-blob4.db'*, was not encrypted, and there were several nonce values inside. The files in the INTERNAL/*files*/ folder were all binary data except for *'device_id'*. We assumed that these data would be used in the decryption process. Multimedia files were encrypted and stored as hidden files in the EXTERNAL/*Android/data*/

Table 7
Data listing of threema.

File	Path	File name
databases	INTERNAL/databases/	threema4.db threema-nonce-blob4.db
files	INTERNAL/files/	device_id key.dat msgqueue.ser .crs-pref_message_drafts .crs-pref_push_token .crs-pref_threema_safe_masterkey .crs-private_key
multimedia	EXTERNAL/Android/data/ch.threema.app/files/data	*.0caccd ... 349b6c *.0caccd ... 349b6c_T
preferences	INTERNAL/shared_prefs/	ch.threema.app_preferences.xml
cache	INTERNAL/cache/image_manager_disk_cache/	*778ee7 ... b98e25.0 journal
backup	EXTERNAL/Threema/Backups/	*threema-backup_59NMJV ... 282863_1.zip

Table 8
Threema's decryption process.

Step	Function	Algorithm	Parameter
1	Passphrase key generation	PBKDF2-HMAC-SHA1(p, s, i, d)	p: passphrase s: salt from key.dat i: iteration(fixed to 10,000) d: dkLen(fixed to 256)
2	Database key generation	(k XOR o) XOR p	k: protected key from key.dat o: obfuscation key(fixed) p: Passphrase key from Step1
3	Multimedia decryption	AES256-CBC(c, k, i)	c: ciphertext from encrypted multimedia file k: key from step2 i: iv from encrypted multimedia file

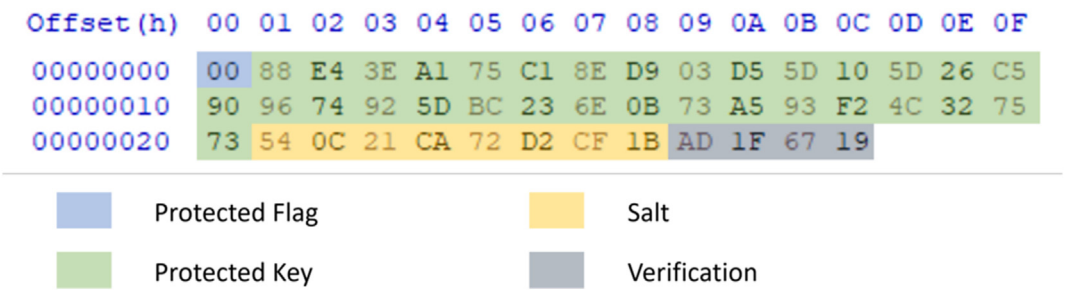

Fig. 11. The structure of key.dat.

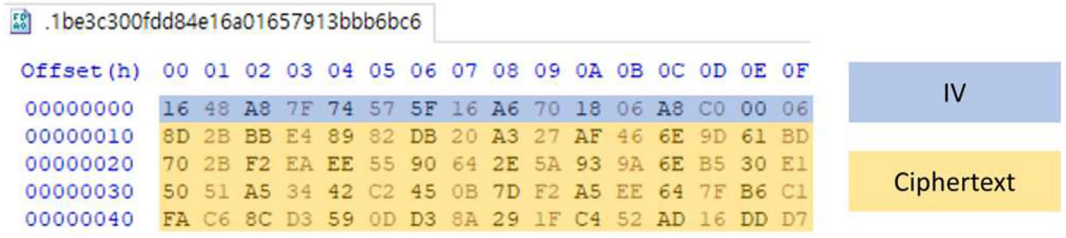

Fig. 12. A structure of Threema's encrypted multimedia file.

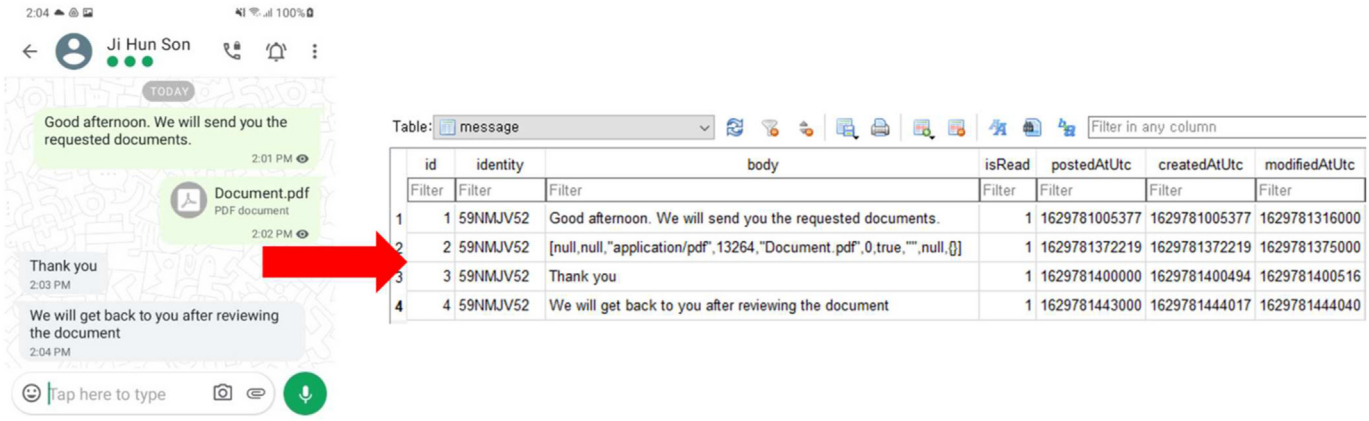

Fig. 13. Threema's database decryption.

ch.threema.app/files/data/, and we assumed that multimedia files with *'_T'* in the name were thumbnails. Cache data existed in the *'cache/image_manager_disk_cache'* folder as in Signal messenger. A backup file is created when a user runs a backup in the app. We found that the data of the *'key.dat'* and *'msgqueue.ser'* file are changed when using the messenger lock feature.

4.3.2. Decryption process analysis

We describe Threema's step-by-step decryption process in Table 8.

4.3.2.1. Database decryption (Step1, Step2). The Threema database key can be obtained from the file *'key.dat'*, whose structure is shown in Fig. 11. The *Protected Flag* changes depending on whether the user enable messenger lock feature. If the database key is unprotected (value is 0), the result of XORing the *Protected Key* with the obfuscation key becomes the database key. The obfuscation key is a fixed value, which is hardcoded in the app. If the database key is protected (value is not 0), the result of an additional xor operation with *passphraseKey* is the database key. The algorithm for calculating *passphraseKey* is PBKDF2-HMAC-SHA1. The Arguments of PBKDF2-HMAC-SHA1 are the user's passphrase and the salt from *'key.dat'*, and some fixed values. Threema's database cannot be decrypted without the user passphrase because the *'key.dat'* file is changed depending on the messenger lock feature.

4.3.2.2. Multimedia decryption (Step3). Threema's multimedia decryption key is the same as the database decryption key. In the encrypted multimedia file, 16 bytes from offset 0 are used as iv, and the rest of the data is used as ciphertext. The structure of the encrypted multimedia file can be seen in Fig. 12. By performing AES256-CBC decryption on these values, we can decrypt the multimedia file. In the case of thumbnail files, the decryption algorithm and parameters are the same as original files.

4.3.3. Verification

The Threema database decryption key is a 67-character string with *x""* added to the 32-byte byte array from Step 2 of the Table 8. For example, when the value of Step2 is *|x00|x01 . . . |x31*, the database decryption key is *x"0001 . . . 31"*. The *PRAGMA* value of

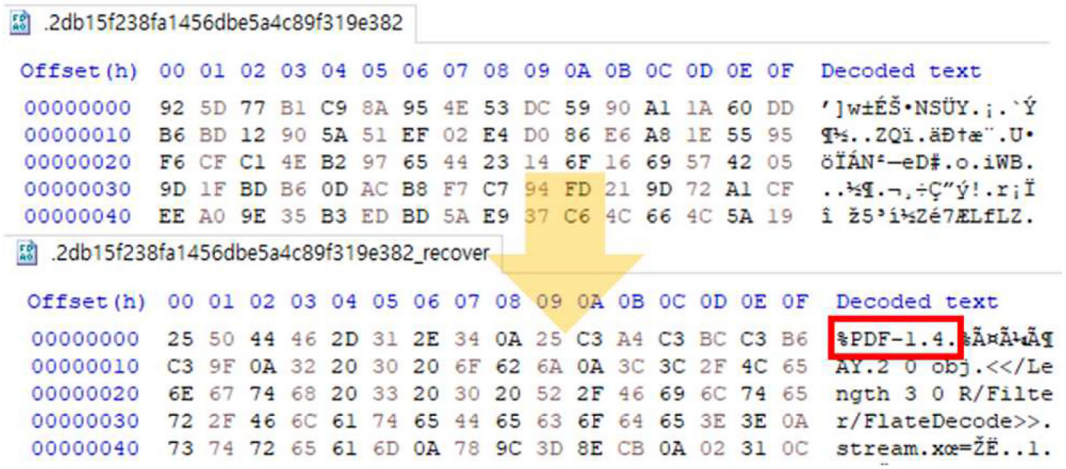

Fig. 14. Threema's multimedia file decryption.

SQLCipher is as follows, and we were able to successfully decrypt the database as shown in Fig. 13.

- *PRAGMA cipher_default_page_size = 4096*
- *PRAGMA cipher_default_kdf_iter = 1*
- *PRAGMA cipher_hmac_algorithm = HMAC_SHA512*
- *PRAGMA cipher_kdf_algorithm = PBKDF2_HMAC_SHA512*

In the *'message'* table of the decrypted database, we found that each row contains information about each multimedia file. We can extract the original file name and mime type from the field *'body'*, and the encrypted file name from the field *'uid'*. Thr result of decrypting multimedia file is Fig. 14. We found that the names of the image and video files were changed to the format *'yyyyMMdd-HHmmssSSS'* (year, month, day, hour, minute, second, and millisecond). Image files are different from the original files because they are compressed when sending messages. However, when the user sent the image *'as a file'*, it was the same as the original. The rest of the multimedia files were almost same as the original, because of the padding added to the end of the file.

4.4. Summarize

Table 9 summarizes the analysis results of this paper. In this table, algorithms and parameters are simply described, thus it is easy to compare the decryption methods between messengers.

5. Discussion

Although Signal, Wickr, and Threema have over 50 million, 5 million, and 1 million downloads respectively on Google Play Store, forensic research on their messengers is insufficient. We extracted data from both unrooted and rooted devices and performed static and dynamic analysis for finding the decryption algorithms of

Table 9
Summarize.

	Database key generation algorithm	Database decryption algorithm	Multimedia decryption algorithm	File decryption algorithm	Data required for decryption
Signal	AES256-GCM-DECRYPT(p, k) p: "pref_database_encrypted_secret" k: android keystore	SQLCipher page_size: 4096 kdf_iter: 1 hmac: HMAC_SHA1 kdf: PBKDF2_HMAC_SHA1	A = AES256-GCM-DECRYPT(p, k) key = HMAC-SHA256(A, d, l) AES256-CTR-DECRYPT(m, key, n) p: "pref_attachment_encrypted_secret" k: android keystore d: data_random l: length(fixed to 32) m: multimedia file n: nonce(fixed to 0)	key = AES256-GCM-DECRYPT(p, k) AES256-CBC-DECRYPT(l, key) p: "pref_log_encrypted_secret" k: android keystore l: log file	Android keystore
Wickr	A = AES256-GCM-DECRYPT (kcd, d) AES256-GCM-DECRYPT(A, kck) kcd: kcd.wic d: devinfo kck: kck.wic	SQLCipher page_size: 4096 kdf_iter: 256000 hmac: HMAC_SHA512 kdf: PBKDF2_HMAC_SHA512	AES256-GCM-DECRYPT(m, k) m: multimedia file k: key from messagePayload	AES256-GCM-DECRYPT(p, k) p: pref file d: devinfo	Android_id
Threema	A = PBKDF2-HMAC-SHA1(p, s, i, d) (k XOR o) XOR A p: password s: salt from key.dat i: iteration(fixed to 10000) d: dkLen(fixed to 256) k: protected key from key.dat o: obfuscation key(fixed)	SQLCipher page_size: 4096 kdf_iter: 1 hmac: HMAC_SHA512 kdf: PBKDF2_HMAC_SHA512	AES256-CBC(d, m) d: database decryption key m: multimedia file	No file	User password

Signal, Wickr, and Threema. As a result, we could obtain the decrypted database, multimedia, log, and preferences files from the extracted data. Forensic investigators can use these results to analyze Signal, Wickr, and Threema. They will be able to obtain evidence from the decrypted conversation history and photos, and will be able to obtain clues to infer the user's behavior from the decrypted log files.

Our paper has advanced results compared to previous studies. In the case of Signal, we developed an application to extract keys from Android Keystore. We succeeded in decrypting database, multimedia, and log files. In the case of Wickr, we found a decryption method without a user's password, found outdated parameters, and expanded the decryption range to the preference file. In the case of Threema, we succeeded in decrypting database, multimedia files. To the best of our knowledge, there are no other studies on Threema's decryption algorithm.

Our paper also has limitations. If we cannot pass the messenger lock, we cannot create a messenger backup. Since a password is required to pass the messenger lock, we cannot obtain data from an unrooted device without the password. In this case, forensic investigators should consider other advanced data acquisition methods (Feng et al., 2018, 2019). In Wickr, messenger backup has not been tested as it requires the Wickr Pro network administrator to enable single sign-on (SSO). In Threema, we could get the decrypted database and multimedia files by unzipping the messenger backup file. However, the results of the analysis are still meaningful because the messenger backup function can disappear at any time.

6. Conclusions

In this paper, we present a methodology to analyze the messenger's decryption algorithm. This methodology covers the entire process of analyzing messenger applications, from data extraction to decryption. We extracted data from the unrooted device through Messenger Backup Migration. We listed the extracted data to obtain hints for analysis. We found a decryption algorithm through static and dynamic analysis and wrote a decryption script for verification. As a result, we could decrypt all encrypted database, multimedia, log, and preferences files of Signal, Wickr, and Threema. Our results broaden the range of files that we can decrypt and discover a new decryption algorithm. The decryption scripts we have disclosed will be useful when forensic investigators analyze messengers. In the future work, we plan to investigate the messenger artifacts that change according to the user's actions and examine the traces left by the disappearing messages.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Almehmadi, T., Batarfi, O., 2019. Impact of android phone rooting on user data integrity in mobile forensics. In: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). IEEE, pp. 1–6.

Android Keystore. Android keystore system android developers. https://developer.android.com/training/articles/keystore?hl=ko. (Accessed 20 August 2021).

Anglano, C., Canonico, M., Guazzone, M., 2016. Forensic analysis of the chatsecure instant messaging application on android smartphones. Digit. Invest. 19, 44–59.

Azhar, M.H.B., Barton, T.E.A., 2017. Forensic Analysis of Secure Ephemeral Messaging Applications on Android Platforms, in: Inter- National Conference on Global Security, Safety, and Sustainability. Springer, pp. 27–41.

Azhar, H., Cox, R., Chamberlain, A., 2020. Forensic investigations of popular ephemeral messaging applications on android and ios platforms. Int. J. Adv. Secur. 13, 41–53.

Barton, T., Azhar, M., 2016. Forensic Analysis of the Recovery of Wickr's Ephemeral Data on Android Platforms, in: the First International Confer- Ence on Cyber-Technologies and Cyber-Systems. IARIA, pp. 35–40.

BBC, 2021. Anom: hundreds arrested in massive global crime sting using messaging app - bbc news. https://www.bbc.com/news/world-57394831. (Accessed 30 August 2021).

Boueiz, M.R., 2020. Importance of rooting in an android data acquisition. In: 2020 8th International Symposium on Digital Forensics and Security (ISDFS). IEEE, pp. 1–4.

Chang, M.S., Chang, C.Y., 2019. line messenger forensics on Windows 10. J. Comput. 30, 114–125.

Choi, J., Yu, J., Hyun, S., Kim, H., 2019. Digital forensic analysis of encrypted database files in instant messaging applications on windows operating systems: case study with kakaotalk, nateon and qq messenger. Digit. Invest. 28, S50–S59.

Cortjens, D., Spruyt, A., Wieringa, W., 2011. WhatsApp database encryp- tion project report. Technical Report. Tech. Rep. 2011. https://www.os3.nl/media/2011-2012.

Electronic Frontier Foundation(EFF). Surveillance self-defense. https://ssd.eff.org/. (Accessed 19 August 2021).

Feng, P., Li, Q., Zhang, P., Chen, Z., 2018. Logical acquisition method based on data migration for android mobile devices. Digit. Invest. 26, 55–62.

Feng, P., Li, Q., Zhang, P., Chen, Z., 2019. Private data acquisition method based on system-level data migration and volatile memory forensics for android applications. IEEE Access 7, 16695–16703.

Google Play Store, a. Signal private messenger - apps on google play. https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms&hl=en_US&gl=US. (Accessed 19 August 2021).

Google Play Store, b. Threema. secure and private messenger - apps on google play. https://play.google.com/store/apps/details?id=ch.threema.app&hl=en_US&gl=US. (Accessed 19 August 2021).

Google Play Store, c. Wickr me – private messenger - apps on google play. https://play.google.com/store/apps/details?id=com.mywickr.wickr2&hl=en_US&gl=US. (Accessed 19 August 2021).

Husain, M.I., Sridhar, R., 2009. Iforensics: Forensic Analysis of Instant Messaging on Smart Phones, in: International Conference on Digital Forensics and Cyber Crime. Springer, pp. 9–18.

Jungfer, M., 2021. Number of threema users climbed to over 10 million - digitec. https://www.digitec.ch/en/page/number-of-threema-users-climbed-to-over-10-million-20061. (Accessed 19 August 2021).

Kaczyński, K., 2019. Security analysis of signal android database protec- tion mechanisms. Int. J. Inf. Technol. Secur. 4, 63–70.

Kim, G., Park, M., Lee, S., Park, Y., Lee, I., Kim, J., 2020. A study on the decryption methods of telegram x and bbm-enterprise databases in mobile and pc. Forensic Science International: Digit. Invest. 35, 300998.

Kim, G., Kim, S., Park, M., Park, Y., Lee, I., Kim, J., 2021. Forensic analysis of instant messaging apps: decrypting wickr and private text messaging data. Forensic Sci. Int.: Digit. Invest. 37, 301138.

Κασαγί αννης, G., 2018. Security Evaluation of Android Keystore. Mas- ter's thesis. Πανεπιστ ημιο Πειραί ως.

Mahajan, A., Dahiya, M., Sanghvi, H., 2013. Forensic Analysis of Instant Messenger Applications on Android Devices arXiv preprint arXiv:1304.4915.

Rösler, P., 2018. On the End-To-End Security of Group Chats in In-Stant Messaging Protocols.

Rösler, P., Mainka, C., Schwenk, J., 2018. More is less: on the end-to-end security of group chats in signal, whatsapp, and threema. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 415–429.

Sabt, M., Traoré, J., 2016. Breaking into the Keystore: A Practical Forgery Attack against Android Keystore, in: European Symposium on Research in Computer Security. Springer, pp. 531–548.

Satrya, G.B., Daely, P.T., Shin, S.Y., 2016. Android forensics analysis: pri- vate chat on social messenger. In: 2016 Eighth International Conference on Ubiquitous and Future Networks (Icufn). IEEE, pp. 430–435.

Scheitle, Q., Wachs, M., Zirngibl, J., Carle, G., 2016. Analyzing locality of mobile messaging traffic using the matador framework. In: International Conference on Passive and Active Network Measurement. Springer, pp. 190–202.

Signal-Blog, 2020. Storage management for signal android. https://signal.org/blog/storage-management-for-android/. (Accessed 19 August 2021).

Statista, 2021. Most popular social networks world- wide as of july 2021, ranked by number of active users. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. (Accessed 19 August 2021).

a Wu, J.. Installation of magisk. https://topjohnwu.github.io/Magisk/install.html#samsung-system-as-root. (Accessed 28 December 2021).

b Wu, J.. Magisk. https://github.com/topjohnwu/Magisk. (Accessed 28 December 2021).

Wu, S., Zhang, Y., Wang, X., Xiong, X., Du, L., 2017. Forensic analysis of wechat on android smartphones. Digit. Invest. 21, 3–10.

Zhang, Y., Li, B., Sun, Y., 2020. Android encryption database forensic analysis based on static analysis. In: Proceedings of the 4th International Conference on Computer Science and Application Engineering, pp. 1–9.