Digital Investigation 28 (2019) S50–S59

Contents lists available at ScienceDirect

Digital Investigation

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

Digital forensic analysis of encrypted database files in instant messaging applications on Windows operating systems: Case study with KakaoTalk, NateOn and QQ messenger

Jusop Choi [a], Jaegwan Yu [b], Sangwon Hyun [c], Hyoungshick Kim [a, *]

[a] Sungkyunkwan University, Republic of Korea
[b] LIGNEX1 Co., Ltd, Republic of Korea
[c] Chosun University, Republic of Korea

ARTICLE INFO

Article history:

Keywords:
Instant messaging applications
Forensic analysis
Database encryption
Key recovery

ABSTRACT

Instant messaging applications store users' personal data (e.g., user profile, chat messages, photos and video clips). Because those data typically include privacy sensitive information, most instant messaging applications are trying to protect the stored data in an encrypted form so that the authorized messaging application itself can only access the data. In this paper, we analyzed the locations and file formats of personal data files in three instant messaging applications (KakaoTalk, NateOn, and QQ) which are the most popularly used in China and South Korea. We particularly examined the encryption and decryption procedures for internal databases in those messaging applications through reverse-engineering. Our analysis results demonstrate how the database files of those instant messaging applications are stored and encrypted. Moreover, in the cases of KakaoTalk and NateOn applications, we found that their encrypted database files can successfully be recovered without requiring user password. We also found that QQ messenger stores the encryption key for the database files into an external server. This implementation may raise another privacy concern because users' personal data can be freely accessed by the service provider without user consent.

Introduction

Instant Messaging (IM) applications such as AIM, MSN Messenger, Google Talk, and WhatsApp help people to communicate with each other. IM applications have become increasingly popular because they have several advantages over other communication methods (e.g., email and phone calls). IM applications provide reliable and timely message delivery between users. Moreover, users are capable of sending messages to other users even in offline, archiving previous messages, and checking whether a message recipient already read a delivered message or not.

However, IM applications often raise serious privacy concerns because users' private messages, photos, and video clips can be exposed to unauthorized parties (Alfred and Sameer Patil, 2012). In an IM application, a user's personal data is typically stored in database files on a device (e.g., PC or smartphone) where the IM application is running. Inherently, such database files would have become one of the most attractive targets for intelligence agencies, police investigators or cybercriminals. For example, the information about friends or contacts could be misused for a variety of cybercriminal activities such as spam, phishing and rogue accounts (Kim et al., 2015). As another example, intelligence agencies or police officers can access suspects' chat database files for investigation which may violate people's privacy without their consent (Fitzpatrick, 2016). Therefore, it is crucial to protect users' personal data stored in the database files against unauthorized accesses.

To protect the database files of a user's personal data, an IM application typically encrypt the database files with a key only accessible by the IM application. Unsurprisingly, the management of this encryption key is critically important because the exposure of the encryption key provides any individual with access to the protected database files of the IM application. Our research was motivated to investigate whether the security of the encryption key in commercial IM applications would actually be acceptable to the standard criteria in the information security community.

* Corresponding author.
E-mail addresses: cjs1992@skku.edu (J. Choi), jaegwan.yu@lignex1.com (J. Yu), shyun@chosun.ac.kr (S. Hyun), hyoung@skku.edu (H. Kim).

https://doi.org/10.1016/j.diin.2019.01.011

In order to provide practical answers to this research question, we examined how the encryption keys are generated and managed in three popularly used IM applications: KakaoTalk (https://www.kakaocorp.com/?lang=en, accessed on 7 January 2019), NateOn (http://nateonweb.nate.com/, accessed on 7 January 2019) and QQ (https://im.qq.com/, accessed on 7 January 2019). KakaoTalk is the most widely used IM application in South Korea with over 49.1 million active users (Corp., 2016). NateOn is also popularly used in South Korea with over 15 million active users (Park et al., 2011). QQ is the most popularly used IM application in China with over 700 million active users (You et al., 2015). Similar to other IM applications, these messengers protect sensitive files such as the database files of users' chat history by encrypting them with a key that is not exposed to anyone except the IM applications themselves.

To evaluate the security level of the database protection mechanisms, we carefully examined the procedure of encrypting the database files including the key generation in KakaoTalk, NateOn, and QQ messengers, respectively, through a forensic analysis using reverse engineering. We particularly analyzed the IM applications running on Microsoft Windows operating systems (32/64 bit, Windows 7 and 10).

Our key contributions are summarized as follows.

- We present a generic forensic analysis methodology for IM applications. Our analysis methodology provides a step-by-step procedure for analyzing database files in IM applications.
- We particularly demonstrated how database files are stored and protected for KakaoTalk, NateOn and QQ applications running on Microsoft Windows operating systems, respectively. We found that the encrypted database files for KakaoTalk and NateOn can be decrypted without the users' consent (i.e., without requiring any information from the users). We also found that the encryption keys for QQ applications are generated on the service provider instead of the client applications, which may result in serious privacy violations on the applications.
- We suggest the best security practices to protect encrypted database files on a local device against reverse engineering.

The rest of this paper is organized as follows. In Section Related work, we briefly summarize previous studies about digital forensic analysis for IM applications. In Section Analysis methodology, we present a generic methodology to analyze the encryption and decryption procedures for database files in IM applications. In Section Case study 1: KakaoTalk, Case study 2: NateOn, and Case study 3: QQ, we explain the encryption and decryption procedures for database files in KakaoTalk, NateOn, and QQ messengers in detail, respectively. In Section Ethical considerations, we present our ethical report to fix the discovered problems in IM applications. Finally, in Section Conclusion, we conclude by summarizing the main results of this work.

Related work

Forensic analysis of popular IM applications such as WhatsApp (https://web.whatsapp.com/), Viber (http://www.viber.com/) and WeChat (https://web.wechat.com/) have attracted considerable attention because users' personal information (e.g., chat messages, photos, contacts, and user profiles) are frequently stored in the database files of the IM applications.

For WhatsApp and Viber, Mahajan and Sanghvi (Aditya Mahajan and Sanghvi, 2013) used a universal forensic extraction device (UFED) physical analyzer to extract user's personal data such as chat messages, photos, contacts and profile pictures from the internal memory of Android devices. They particularly showed how to extract metadata such as file names and transmitted times of chat messages. However, the storage locations of those files were not found in an automated manner. Chat messages and phone call history data could only be obtained through a manual analysis. Thakur (2013) performed a forensic analysis of WhatsApp applications on Android to determine what types of user data can be extracted from non-volatile external storage and internal memory of the applications. Thakur particularly identified that deleted messages in WhatsApp applications can be extracted from the internal memory of an Android device. Gao and Zhang (2013) analyzed database files of WeChat applications and found the structures, file extensions and locations of those database files.

Recently, Anglano (2014) also presented a forensic analysis of the metadata left on Android devices by WhatsApp messengers, revealing the structures (e.g., tables and fields) of database files used in WhatsApp. Unlike existing studies, however, Anglano introduced a new approach to decode and interpret the metadata obtained from WhatsApp messengers on Android devices.

Encryption is a common security practice used in IM applications because diverse types of important personal data such as chat messages and user profiles are stored and managed for many different purposes. According to the study (Alfred and Sameer Patil, 2012), users were concerned about chat message logs that could be abused. However, there have been a few studies focusing on the analysis of encrypted databases. Barghouthi and Said (Al Barghuthi and Said, 2013) analyzed several IM applications (Yahoo, Google Talk, Skype, Facebook, and Gmail) to identify the encryption methods used for protecting their chat messages through the Wireshark (packet sniffer software) and forensic investigation tools. Choi et al. (2017) analyzed the backup and restore service in KakaoTalk, revealing that chat messages can be recovered from the backup files when a user chooses a weak password for the backup service.

In this paper, we present a forensic analysis of KakaoTalk, NateOn and QQ messengers to evaluate the security of their procedures for encrypting the chat database files. We were particularly interested in the encryption key generation procedure to extract the plaintext data from the encrypted database files of chat messages.

Analysis methodology

We conducted a forensic analysis of KakaoTalk, NateOn, and QQ applications running on Windows OS. All these messenger applications use encrypted chat database files to maintain users' chat history in a secure manner where the database files are associated with each chat room that the user is involved in. When the user enters a chat room in his/her application, the database files associated with the chat room are loaded and decrypted on the device memory. The goal of our forensic analysis is to obtain the plaintext data from the encrypted chat database files without any secret information from the user. This section summarizes the generic procedure we followed in that forensic analysis.

1. Find the (encrypted) chat database files in an instant messenger application. These files can be automatically identified by a process monitoring program such as Process Monitor[1] and Resource Monitor[2] on Windows series. In KakaoTalk messenger, "chatLogs_[chat room ID].edb" is used for a chat room; in

[1] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.

[2] https://blogs.technet.microsoft.com/yongrhee/2011/01/04/how-to-pull-the-information-that-resource-monitor-resmon-exe-provides/.

NateOn messenger, “msginfo.db” is used; in QQ messenger, “Msg2.0.db” is used.
2. Check whether the chat database files are encrypted. This can be done by analyzing the entropy of the data stored in the files.
3. Find out when the chat database files are encrypted and/or decrypted during the execution of the messenger application. This process can be implemented by capturing the time when the files that are opened for use in the application are loaded into the memory.
4. Identify the cryptographic algorithms used in encrypting chat database files by examining the assembly codes (e.g., the assembly codes for AES encryption shown in Fig. 1) and the memory contents (e.g., loaded S-box values) of the application.
5. Analyze the procedure of generating an encryption key and encrypting the chat database files using the key. We are particularly interested in identifying how the encryption key is generated. For this purpose, we figured out what input values are used, which hash algorithm is used, and whether a particular key generating algorithm is used in generating the encryption key. For example, we could find out the type of the hash algorithm being used, depending on the existence of the initial input values specific to the hash algorithm.

In this paper, we show the feasibility of this analysis method through three case studies in Section Case study 1: KakaoTalk, Case study 2: NateOn and Case study 3: QQ.

Case study 1: KakaoTalk

KakaoTalk is the most popular IM service in South Korea with more than 49.1 million active users (Corp., 2016), and both mobile and desktop versions are available. Similar to WhatsApp, it provides users with a variety of features. Its fundamental feature is to send multimedia data to friends. In addition, it also provides some advanced and useful services for users such as group chat, VoIP, auto-finding friends, and chat history backup. The KakaoTalk messenger was originally developed as a mobile application (similar to WhatsApp) for smartphones such as Android and iOS devices, but the PC and Mac versions were also released recently.

There are a few studies on the security of KakaoTalk services. Kim et al. (2015) found some vulnerabilities in the automatic friend registration feature in KakaoTalk services, which can be exploited to illegally gather sensitive user information such as phone numbers, names and photos. Park et al. (Park and Kim, 2015) showed that the activities of KakaoTalk users could be inferred with a high probability from captured network packets even when those packets are encrypted using SSL/TLS. These existing studies were mainly focused on user privacy issues by looking at some features (e.g., automatic friend registration) or analyzing network traffic. To the best of our knowledge, there is no previous work that has analyzed the security of encrypted chat history databases.

KakaoTalk PC version firstly appeared in June 20th, 2013, and its

```
0073B4DB  . 8B4D 10       MOV ECX,DWORD PTR SS:[EBP+0x10]
0073B4DE  . 8B71 04       MOV ESI,DWORD PTR DS:[ECX+0x4]
0073B4E1  . 8B51 08       MOV EDX,DWORD PTR DS:[ECX+0x8]
0073B4E4  . 3350 08       XOR EDX,DWORD PTR DS:[EAX+0x8]
0073B4E7  . 3370 04       XOR ESI,DWORD PTR DS:[EAX+0x4]
0073B4EA  . 57            PUSH EDI
0073B4EB  . 8B39          MOV EDI,DWORD PTR DS:[ECX]
0073B4ED  . 3338          XOR EDI,DWORD PTR DS:[EAX]
0073B4EF  . 894D E4       MOV DWORD PTR SS:[EBP-0x1C],ECX
0073B4F2  . 8B49 0C       MOV ECX,DWORD PTR DS:[ECX+0xC]
0073B4F5  . 3348 0C       XOR ECX,DWORD PTR DS:[EAX+0xC]
```

Fig. 1. Codes for AES-128 in CBC mode in the KakaoTalk messenger.

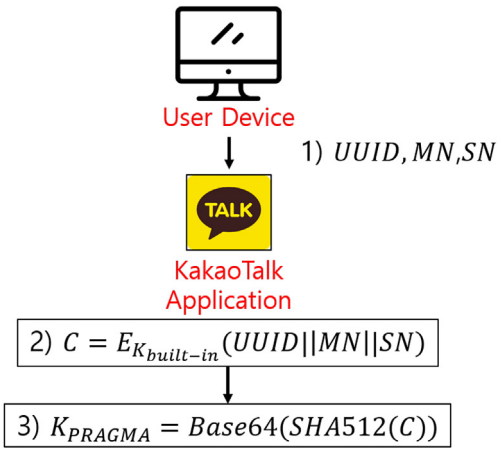


Fig. 2. PRAGMA key generation procedure before a user logins to KakaoTalk application.

chat history database has been encrypted since their appearance. We found that a sophisticated packer called Themida[3] was used to protect the KakaoTalk program from debuggers. Themida could be used to protect a program with various features such as detecting debuggers/dumpers/virtual machines/monitor blocker, obfuscating entry points, encrypting resources and so on. To bypass anti-debugging and monitor blocker features, we used Ollydbg version 1.10[4] with the Phantom[5] plug-in and the StrongOD.[6] We also used several Ollydbg plug-ins to easily identify the code used for encrypting chat databases. To search the locations of a specific value in the memory, CheatEngine[7] was used. We particularly analyzed the KakaoTalk version 2.0.8.990 running on Windows 7.

In the following, we first present the procedure for generating a key used to encrypt and decrypt the chat history database file and explain the structure of the chat history database.

Key generation procedure

The key generation procedure of KakaoTalk application is divided into before and after a user logins to KakaoTalk application. Before the user logins, Kakaotalk application initially computes a 512-bit length value called the PRAGMA key with some device-specific information. Based on the PRAGMA key, the key for encrypting and decrypting the chat history database files are generated eventually.

Fig. 2 illustrates the procedure for generating the PRAGMA key before the user logins.

1. When a user runs KakaoTalk application, the application first collects some device-specific information such as the universally unique identifier (*UUID*) of the user device and the model name (*MN*) and the serial number (*SN*) of the hard disk installed on the device.
2. KakaoTalk application is preconfigured with the built-in key $K_{built-in}$ and *IV* which is set to a null byte value. By performing AES-128 in CBC mode with $K_{built-in}$ and *IV*, the application computes the ciphertext *C* of the concatenation of *UUID*, *MN* and

[3] http://www.oreans.com/themida.php.
[4] http://www.ollydbg.de/.
[5] https://tuts4you.com/download.php?view.1276.
[6] https://tuts4you.com/download.php?view.2028.
[7] http://www.cheatengine.org/.

SN. Note that AES-128 in CBC mode is the only encryption/decryption algorithm used in KakaoTalk applications.
3. To obtain the PRAGMA key denoted as K_{PRAGMA}, KakaoTalk application computes the hash value of C using the SHA-512 hash algorithm and then encodes the hash value using the Base64 algorithm. The Base64-encoded hash value of C finally becomes K_{PRAGMA}.

After K_{PRAGMA} has been generated, KakaoTalk application asks a user to enter the user's password. With the user password denoted as PW_u and K_{PRAGMA}, the application can generate the key and IV used for encrypting/decrypting the chat history database files of the application. Fig. 3 illustrates the procedure for computing the encryption key and IV.

1. User u types his/her password to login KakaoTalk application. Let PW_u denote the entered user password.
2. KakaoTalk application submits a login request including PW_u to a KakaoTalk server. In order to protect PW_u from eavesdroppers, we assume that a secure channel between the application and the server is available, and how to set up the secure channel is out of the scope in this paper.
3. The KakaoTalk server checks whether PW_u is valid. When it is valid, the server accepts the login request and replies the KakaoTalk application with a random nonce N_u, which uniquely identifies user u.
4. KakaoTalk application constructs a string S_u by repeatedly concatenating K_{PRAGMA} and N_u until the total length of S_u becomes greater than or equal to 512 bytes; $S_u = K_{PRAGMA}||N_u \cdots K_{PRAGMA}||N_u$, where $|S_u| \geq 512$. Finally, the application truncates S_u to 512 bytes if the length of S_u is greater than 512 bytes.
5. KakaoTalk application generates the encryption key K_u of 128 bits long by calculating the MD5 hash value (Rivest, 1992) of S_u.
6. To obtain the IV required for the encryption algorithm, KakaoTalk application encodes K_u using the Base64 algorithm and then computes the hash value of the Base64-encoded K_u. The computed hash value is used as IV.

Using the key K_u and IV which have been obtained through the above procedures, the encrypted chat history database files can be decrypted using AES-128 in CBC mode. Table 1 summarizes the functions and their parameters used for key generation and chat database file encryption in the KakaoTalk application.

Database analysis

Similar to other messenger applications, KakaoTalk application supports various types of messages including text, emoticons, video clips, photos, files, etc. These messages are stored in several database files according to their types. The database files are stored in a pre-configured fixed location[8] on a file system, and there are no restrictions on accessing the files so that anyone can easily access them. KakaoTalk application uses SQLite format 3 for its database files. We carefully examined the structures of the database files used for the chat history.

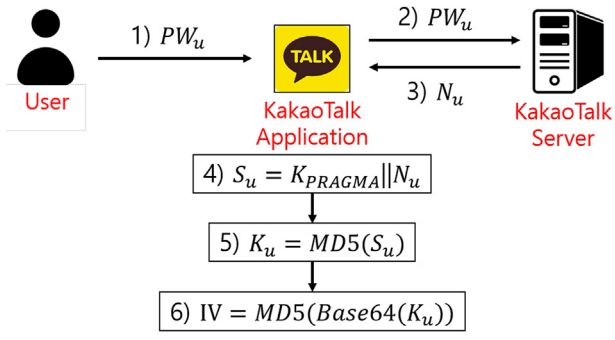


Fig. 3. Encryption key generation procedure after a user logins.

KakaoTalk applications use the following five types of database files to keep the chat history:

- chatAttachmentInfo contains the following information of a file attachment: the sender, the file size, the transmission time, the URL to download the file, the expiration time of the URL link, the token value for controlling access to the file, etc.
- chatListInfo consists of five tables and contains the following information of a chat room: the count of new messages, the number of members in this chat room and their KakaoTalk IDs, the notification messages, etc.
- chatLogs is created for each chat room and stores all the messages that have been exchanged among the members in the chat room.
- chatPhotoInfo_v2 contains the following information of each photo: the sender's KakaoTalk ID, the transmission time, the token value, the checksum of the photo file, the image type (e.g., JPEG, GIF, etc.) and so on. Note that this is the only database which is not encrypted.
- openLinkListInfo contains information of open chat rooms that the current user is involved in. An open chat room in KakaoTalk is a public chat room which allows anyone to participate in, thus some of the members in an open chat room may not be friends of the current user. This database contains the host user ID, the chat room name, and the privilege of each open chat room.

KakaoTalk applications create and encrypt a set of databases mentioned above for each chat room. Among all the databases, the chatLogs database stores all the messages exchanged among the members in a chat room, and our goal is to recover the messages. Thus we focused on analyzing the chatLogs database. The chatLogs database is further split into three tables, i.e., chatLogsDrafts, chatLog_attachment, and chatLogs, and all the activities (including the messages) in a chat room are stored into the chatLogs table. Thus, we particularly focused on the table. Table 2 describes the meaning of each field in the chatLogs table.

Case study 2: NateOn

NateOn is one of the popular IMs in South Korea with 15 million active users (Park et al., 2011). Due to its various functions such as remote desktop control and sharing painting board, NateOn is widely used in companies for business purposes in Korea.

Related to forensic analysis of NateOn, there are some studies that focused on analyzing the authentication mechanism (Shin et al., 2007; Lee et al., 2011). At the time of the studies in 2009, NateOn stores chat history databases only in the remote server, not in a local storage, and that was the reason why the authors focused on identifying the authentication mechanism required for retrieving chat databases from the server. But now, NateOn applications store chat database in local storage, so it is possible to recover chat history from the local database. Thus we performed forensic analysis of NateOn application in order to identify the encryption mechanism and the database structure.

[8] %LOCALAPPDATA%\Kakao\KakaoTalk\(hashed value depending on user information)\chat_ data.

Table 1
Summary of functions and parameters for key generation and database encryption in KakaoTalk.

Function	Algorithm	Parameter
PRAGMA key generation	AES128 in CBC	Universal unique identifier (*UUID*) Model name (*MN*) Serial Number (*SN*) built-in key ($K_{built-in}$)
	SHA512	Output of AES encryption; 2) in Fig. 2
	Base64	Output of SHA512; 3) in Fig. 2
Key generation for DB encryption	MD5	PRAGMA key (K_{PRAGMA}) User sequence number (N_u)
IV generation for DB encryption	Base64	DB encryption key (K_u)
	MD5	Output of Base64; 6) in Fig. 3
Chat database (de)encryption	AES128 in CBC	Key K_u for DB encryption IV for DB encryption

Table 2
Structure of the chatLogs table.

Field name	Meaning
logId	sequence number of the record (set by Kakao Talk server)
authorId	the user id who sent the message
type	message type (1: normal message, 2: photo, 18: attachment)
clientMsgId	unknown
sentAt	Unix epoch time (10 digits), represents the sending time.
message	sent or received message
attachment	the file name, URL, and etc.
pc_logout	unknown, always set to '0'
prev_msg_missing	whether previous message missing (if the previous message lost, sets 1 if not, sets 0)
next_msg_missing	whether next message missing. (if the next message didn't lost, sets 0)
deleted	if the message is deleted, sets 1 if not, sets 0
prevLogId	the logId, which in first row, of previous message
feed_flags	unknown, always set to '0'

The PC version of NateOn was first released in October 2002, and its chat history databases have been encrypted since March 2006.

We could not find any packing program used to protect NateOn applications from debuggers. To search the locations of specific values in memory, we used CheatEngine. We particularly analyzed the NateOn version 5.1.22.0 running on Windows 7, and we checked that the same decryption method identified below is also applied to the recent version 6.0.13.0 released on August 2018 and running on Windows 7 and Windows 10.

Our analysis results are divided into two parts. We will first present the procedure for generating a key used to encrypt and/or decrypt the chat history database file. Then we will explain the structure of the database.

Key generation procedure

The procedures for generating an encryption key are as follows (see Fig. 4):

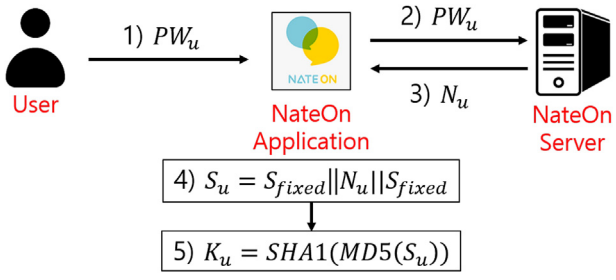


Fig. 4. Key generation procedure for encrypting chat history databases in NateOn messengers.

1. To use NateOn application, a user u types his or her password PW_u in the password field of the NateOn application.
2. NateOn application then issues the login request with PW_u to the NateOn server.
3. If the user password PW_u is valid, the NateOn server accepts the login request and replies the NateOn application with a specific number N_u, which is the unique identifier of this user u and his/her NateOn application.
4. NateOn application generates a string S_u by concatenating two S_{fixed} values and N_u received from the NateOn server as follows; $S_u = S_{fixed} :: N_u :: S_{fixed}$. For example, when S_{fixed} = "abcd" and N_u = "1000", S_u becomes "abcd::1000::abcd". The fixed string S_{fixed} is pre-configured with "SKCommunications_NateOn40" by the developer. We note that this string is common for all devices.
5. The encryption key K_u is generated by applying the MD5 (Rivest, 1992) and SHA1 (Eastlake and Jones, 2001) hash algorithms to the string S_u in order.

Along with the key generation procedure, we found that NateOn applications use the RC4 algorithm for encrypting/decrypting their chat history database files. Given any encrypted chat history database file of user u, it is possible to decrypt the database file using the key K_u obtained by following the same key generation procedure. User's number N_u can be figured out through brute-force attack or the attacker's database. User's number N_u is sequentially increased based on registration date. In addition, because the length of the user number N_u is within 10 digits, it may be possible to correctly guess the user number N_u through a brute-force attack. Moreover, we can improve the feasibility of the attack using the attacker's chat database. In NateOn, a user can add another user as a friend if the user's phone number is known to the user. In this case, the victim's number N_u can be stored into the msginfo database. Therefore, the attacker can finally obtain the victim's user number N_u. Table 3

Table 3
Summary of functions and parameters for key generation and database encryption in NateOn.

Function	Algorithm	Input Parameter
Key generation for DB encryption	MD5	User sequence number (N_u) Fixed string (S_{fixed})
	SHA1	Output of MD5; 5) in Fig. 4
Chat database (de)encryption	RC4	Key K_u for DB encryption

summarizes the functions and their parameters used for key generation and chat database file encryption in the NateOn application.

Database analysis

NateOn applications support a variety of message types such as text messages, emoticon messages, and files. These contents are stored in several database files through the chat history features. NateOn applications store database files in a pre-configured path on a file system.[9] Because this path is fixed and no restrictions are imposed on accessing the database files, anyone can easily find and access the databases. NateOn also uses SQLite format 3 for its database files. We carefully examined the structure of the database files to know what privacy-sensitive information each database stores.

NateOn applications use five types of database files as follows:

- fileBoxInfo contains the information of attached files such as the sender, the file size, the transmission time, the expiration time, the file name including the local file system path, etc.
- nbi contains the information about non-friends. NateOn allows users to send messages to even non-friends.
- msginfo contains all messages that the user has sent and received. For each chat room, NateOn applications create tables with the room ID, and the tables store the messages, the information of file attachments, and the unique identifier N_u of the sender given by the server.
- NateOn provides the feature of a secret chat room, which allows only the room members to read messages exchanged in the chat room. With this feature, even the NateOn server cannot read messages in any secret chat room. pkInfo database file contains the information of the keys associated with each secret chat room, including the server's public key and the user's public and private key pair. However, every message exchanged in a secret chat room is stored in plaintext in the msginfo database.
- roomInfo contains the information of each chat room, including the unique identifier and the name of each member in the room, the last sent message, and its sender, etc.

Unlike KakaoTalk messengers, NateOn applications create five database files in total, regardless of the number of chat rooms that a user participates in. Instead, NateOn applications maintain a separate table in the msginfo data-base for each chat room. More specifically, when the current user is involved in n chat rooms, the msginfo database contains n tables for each chat room plus one additional table named MSGDB_DEFINE which contains the version information of NateOn application being used by the user. Because the msginfo database stores all activities in every chat room, we focused on analyzing this database. Table 4 summarizes the meaning of each field in this database.

[9] %LOCALAPPDATA%\SKComunications\NATEON5\(a string in hexadecimal format)\localdb.

Case study 3: QQ

QQ messenger is the most popular IM service in China, and both mobile and desktop versions are available. According to statistics (Smith, 2017), more than 861 million active users per month use this IM service, and this number represents 59.5% of the total Internet users between 16 and 65 years of age in China. One time, the number of users who were using QQ was even over 266 million. QQ messenger basically supports Chinese, but the international version supports other languages, such as English, German, Japanese, Spanish, French, and Korean. QQ also provides a wide variety of features similar to WhatsApp and KakaoTalk, including text messaging, instant video, and voice chat, as well as online and offline file transmission, building complete and diversified cross-platform communication, online social games, music, micro-blogging, movies, among others. In particular, the chat client application of QQ supports real-time translation of over 50 languages (Wikipedia, 2017; Tencent, 2017).

In (Yu et al., 2014), Yu et al. performed a forensic analysis of the mobile version of QQ messenger (version 0x0608) to figure out the secure communication protocol in use between the mobile (client) application and the server. By analyzing wireless network packets as well as the application procedures, they successfully captured a session key in use and consequently recovered messages from encrypted packets. Different from their work, our major goal is to recover messages from encrypted chat history database files in a client-side application of QQ messenger. To achieve this goal, we analyzed a PC client application of QQ messenger and found that the key used for encrypting chat database files is generated by the QQ messenger server instead of the client application. The key is delivered to the client application through a secure channel based on Kerberos (Neuman and Ts'o, 1994). Based on our traffic analysis, we found that the key can be used to decrypt the encrypted chat database files.

In fact, this security practice has a potentially serious issue. Due to such centralized key generation and management, unlike other messengers (e.g., KakaoTalk and NateOn), QQ messenger inherently accompanies a key escrow problem that may result in privacy violations on QQ messenger; that is, the Chinese government or intelligence agency is able to access all the chat messages of every user through the QQ server. We also confirmed that the same issue exists in the mobile version of QQ messenger (Yu et al., 2014), even though we did not conduct an in-depth analysis of the mobile version in this paper.

The PC version of QQ messenger was first released in 1999, and an international version of the application was introduced for Windows in 2009 (Wikipedia, 2017). In this analysis, we particularly analyzed the international version (2.1.1369.0) of QQ application running on Windows 7. There was no packing program to protect QQ applications from debuggers. Similar to the analysis of KakaoTalk and NateOn, we used the following tools for analysis: Ollydbg version 1.10 with the Phantom plug-in and the StrongOD.

Our analysis results were divided into two steps. In the following, we will first present the procedure for generating a key that is necessary to encrypt/decrypt chat history databases, and then explain the structure of the databases.

Key generation procedure

Unlike KakaoTalk and NateOn, keys for encrypting chat history database files in QQ messengers are generated on the server side, not on the client side. After generating an encryption key for a user, the server securely delivers it to the user using Kerberos (Kohl and Neuman, 1993). Kerberos is an authentication protocol based on 'Ticket', which uniquely identifies a user and his/her access rights,

Table 4
Structure of a table of the msginfo database.

Field name	Meaning
LINE_SEQ	sequence number of the record (start the number 1 in each table)
LINE_TIME	Unix epoch time (10 digits), represents the sending time
JOINER	the user id who joins/invites this chat room
FROM_ID	the user's sequence number, name and account ID who sent the message
MESSAGE	sent or received message
MSG_FONT	the message's font size, color and style
COMMAND	a type of commands (only observed "MESG")
CONTENT_TYPE	message type (wbml: normal message, file: attached file and etc)
CONTENT	attached file's information including thumb image, downloadable URL, expired time and etc

and requires two servers, i.e., authorization server and service server. To request a service in a Kerberos-based system, a user initially sends a message to the Authorization Server (AS) to request a Ticket Granting Ticket (TGT). After verifying the user by checking the database of user credentials, the AS sends a TGT and a session key to the user. In QQ messengers, the server generates a key for encrypting chat history database files of each user, encrypt the key with the TGT of the user, and sends the user the encrypted key.

The QQ application uses TEA (Tiny Encryption Algorithm) (Wheeler and Needham, 1994) to securely store chat messages in the local storage. The TEA algorithm is a block cipher based on Feistel structure designed by Wheeler and Needham and uses the block size of 64 bits and the key length of 128 bits. QQ application uses a custom value (0x61c88647) as key schedule constant of the TEA algorithm and also uses a variant of XEX (xor-encrypt-xor) mode (Rogaway, 2004) (see Fig. 5). Table 5 summarizes the functions and their parameters used for key generation and chat database file encryption in the QQ messenger application.

Database analysis

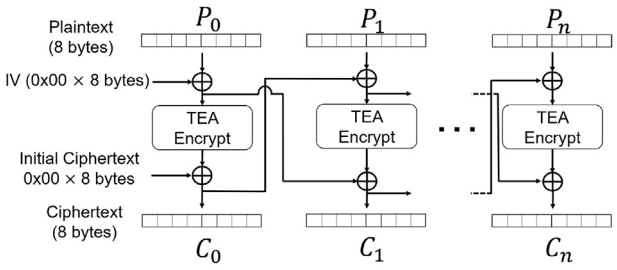
Similar to other messengers, QQ applications support a variety of message types such as text messages, emoticon message, video, image, and files. When we use default path of installation of QQ messenger application in Windows 7, those contents can be stored as several databases in default path[10]. This path can be changed by users using options, but almost user does not change, so it can be easily accessed to databases. All databases are stored in the folder appending to the QQ account which is 10 digits integer.

QQ has 8 of database files. However, we only analyzed the Msg2.0 database because we focused on the message storing method. Msg2.0 stores the encrypted chat messages. The format of the Msg2.0 database is OLE (Object Linking and Embedding) compound file which is developed by Microsoft (Microsoft Corp, 1999). OLE compound file is similar to a small file system (e.g., EXT, FAT, NTFS and etc.), and is composed of 'storage' which is similar to a directory and 'stream' which is similar to a file.

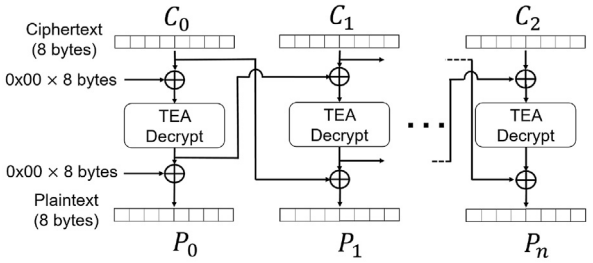
Fig. 6 shows OLE structure of the Msg2.0 database and its structures are as follows:

- buddy storage contains the chat history had been communicated. The buddy storage has several storages according to accounts of friends who have been communicated. In the buddy storage, content.dat stream stores the encrypted chat history.
- content.dat stream contains the chat history of each opposite user. To decrypt the content.dat, we have to understand the structure of content.dat stream because the stream is the unique structure. Fig. 7 shows structure of the content.dat stream. The stream has message history consist of record header, encrypted message, and information of the encrypted message. In Fig. 7, first 20 bytes are record header which has a size of the record, record index, and timestamp indicating when the message was received. Last 4 bytes indicate the size of the encrypted message stream.

[10] %USERPROFILE%\Documents\Tencent Files\(account identified number).



(a) Database encryption process of QQ.



(b) Database decryption process of QQ.

Fig. 5. Encryption and decryption process of QQ.

Table 5
Summary of functions and parameters for key generation and database encryption in QQ.

Function	Algorithm	Input Parameter
Key generation for DB encryption	Kerberos	Unknown
Chat database (de)encryption	TEA in a variant of XEX mode	Key from Kerberos Fixed initial vector

Using the decryption key which is extracted in memory, we can decrypt the encrypted message stream by following the decryption process of QQ as shown in Fig. 8. The decrypted message contains a signature for the message ("MSG"), timestamp, text feature (e.g., size, shape, bold and etc) and message.

Fig. 9 shows a message information block in QQ. A message information block contains the type and size of a message data, the data itself, and information of the variable storing the data. The

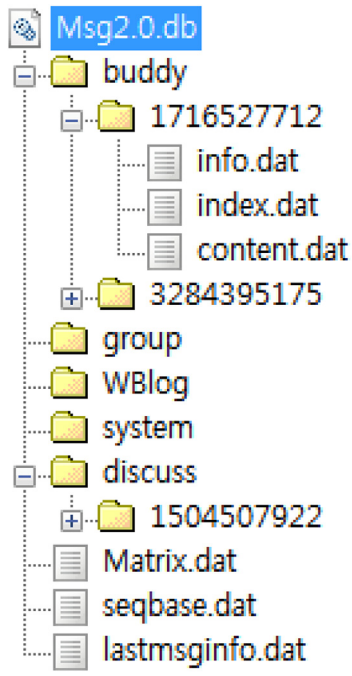


Fig. 6. OLE structure of Msg2.0 database in QQ.

structure of a message information block is shown at the bottom of Fig. 9. The first byte indicates the type of data. The next 2 bytes represents the length of the variable name of the data, and then the variable name itself follows the length field. The next 2 bytes represents the size of the data, which is followed by the data itself. The variable name and the data are encrypted using XOR. First, the XOR operation is performed for each byte of the length field of the data stream with each other. Then, an additional XOR operation is performed for each byte of the data stream. After decrypting the information contained in the message, we can see the user ID number and the nicknames of the sender and the receiver.

- discuss storage contains the group chat history. Each group chat has a 10 digit integer identifier to divide each group. Similar to buddy storage, messages of group chat history are stored in discuss storage in encrypted.

Ethical considerations

Our motivation for this work is to analyze potential risks to the encryption mechanisms for protecting chat history database files from unauthorized accesses in three popular instant messaging applications (KakaoTalk, NateOn, and QQ). We did not collect any personal data without the users' agreement during this work. We completely reported the vulnerabilities discovered in this paper to relevant organizations; Kakao corporation running the KakaoTalk service and SK Communications Co., Ltd. running the NateOn service. We also reported the same vulnerabilities anonymously to Korea Internet and Security Agency (KISA) running a bug bounty program for applications used in South Korea.

After we reported the vulnerability, KakaoTalk released the patch fixing the reported vulnerability.

Conclusion

In this paper, we presented a forensic analysis of the encryption features for users' chat history in KakaoTalk, NateOn, and QQ which

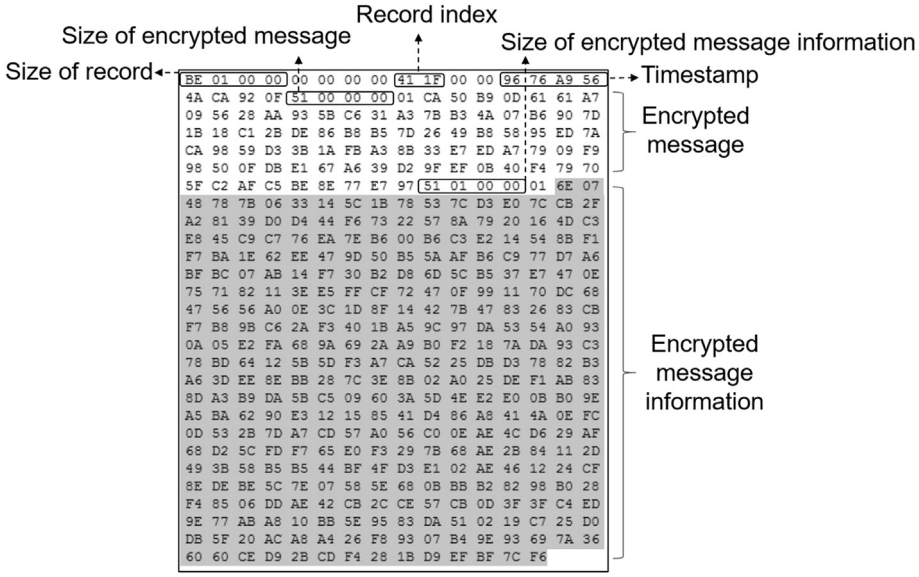


Fig. 7. Message record structure in QQ.

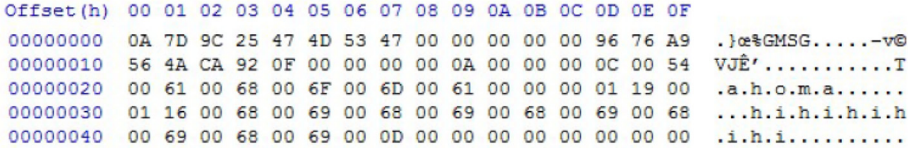


Fig. 8. Decrypted message.

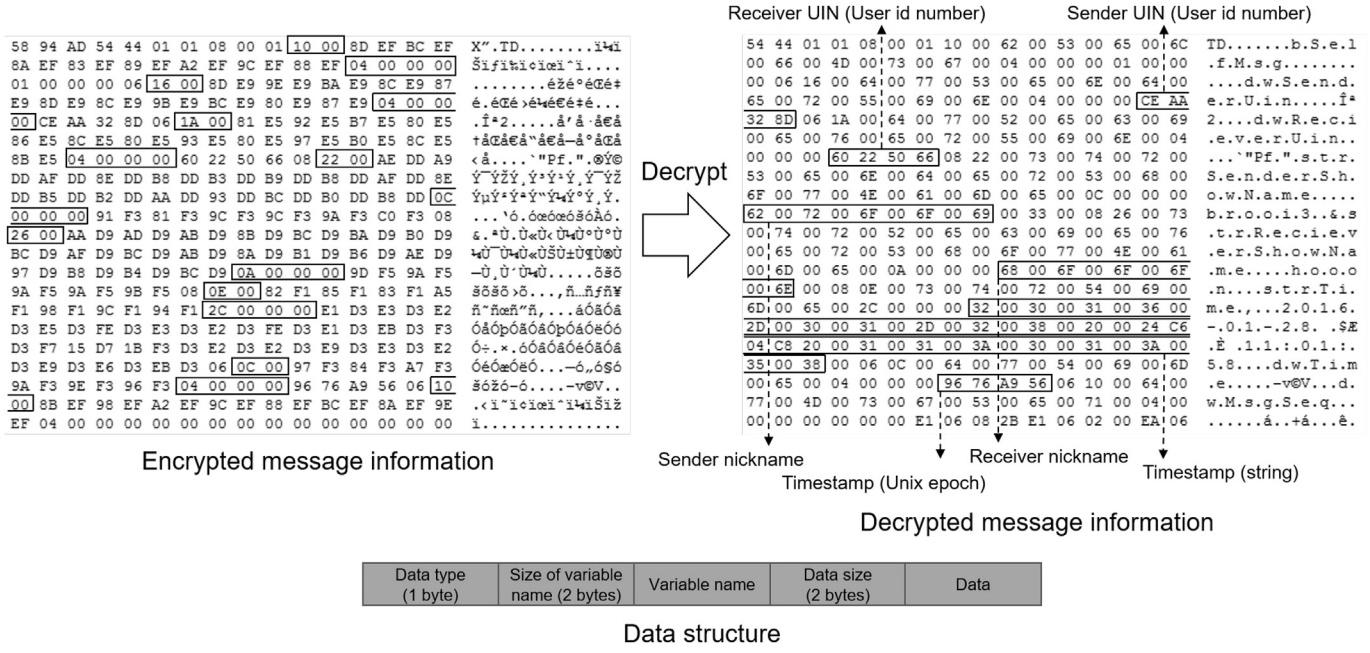


Fig. 9. Message information block in QQ.

are the most popular IMs in China and South Korea. This forensic analysis revealed how an encryption key is generated and how chat history database files are internally stored and encrypted in these IMs.

In the cases of KakaoTalk and NateOn, we found that a user's chat messages can be recovered from encrypted database files without any secret information (e.g., user password) from the user. In the case of QQ messenger, we found that an encryption key for each client application is generated by the service provider and delivered to the client application. This design of centralized key management may raise another privacy concern because the service provider can access every user's chat messages without any restrictions. Unlike QQ messenger, key generation in a KakaoTalk application contains the device-specific information (e.g., *UUID*, *MN*, and *SN*) which is not exposed to the service provider. Without knowledge of such device-specific information, the service provider cannot obtain the same key as the device owner and consequently cannot decrypt the device owner's chat messages from the encrypted database files.

Our study shows how difficult it is to securely protect the user's sensitive data files with only pure software technologies against sophisticated attackers. Although we cannot obtain the encryption key only using application analysis in QQ messenger, it has a risk that messenger server or government can obtain the user's private messages because the key is only managed by the server side. Although we currently limited our security analysis to KakaoTalk, NateOn and QQ messenger service only, we believe this type of attack can also be applicable to other IM applications.

In future work, we plan to analyze the chat history protection features of other messengers, and also develop countermeasures to protect encrypted databases against forensic analysis.

Acknowledgments

This work was supported in part by the ICT R&D program (No.2018-0-00532) and the NRF (No.2017K1A3A1A17092614, NRF-2016R1A6A3A11930593). The authors would like to thank all the anonymous reviewers for their valuable feedback.

References

Aditya Mahajan, M.S.D., Sanghvi, H.P., 2013. Forensic analysis of instant messenger applications on android devices. Int. J. Comput. Appl. 68 (8), 38–44.
Al Barghuthi, N.B., Said, H., 2013. Social networks IM forensics: encryption analysis. J. Commun. 8 (11), 708–715.
Alfred, Kobsa, Sameer Patil, B.M., 2012. Privacy in instant messaging: an impression management model. Behav. Inf. Technol. 31, 355–370.
Anglano, C., 2014. Forensic analysis of WhatsApp messenger on android smartphones. Digit. Invest. 11 (3), 201–213.
Choi, J., Park, J., Kim, H., 2017. Forensic analysis of the backup database file in KakaoTalk messenger. In: Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on. IEEE, pp. 156–161.
Corp, K., 2016. 2Q FY2016 Earnings Conference Call. http://www.kakaocorp.com/upload_resources/ir/event/ir_event_20160825012832.pdf. (Accessed 7 September 2016).
Eastlake 3rd, D., Jones, P., September 2001. US Secure Hash Algorithm 1 (SHA1). RFC 3174, Cisco Systems. https://tools.ietf.org/rfc/rfc3174.txt. (Accessed 24 January 2017).
Fitzpatrick, A., 2016. Here's How Police Get a Suspect's Facebook Information. http://mashable.com/2012/12/18/police-facebook/#ERE6rnihaPqw. (Accessed 14 September 2016).
Gao, F., Zhang, Y., 2013. Analysis of WeChat on iPhone. In: 2nd International Symposium on Computer, Communication, Control and Automation, pp. 278–281.
Kim, E., Park, K., Kim, H., Song, J., 2015. Design and analysis of enumeration attacks on finding friends with phone numbers: a case study with KakaoTalk. Comput. Secur. 52, 267–275.
Kohl, J., Neuman, C., September 1993. The Kerberos Network Authentication Service (V5). RFC 1510, ISI. http://www.rfc-editor.org/rfc/pdfrfc/rfc1510.txt.pdf. (Accessed 22 January 2019).
Lee, J.-K., Han, J.-S., Lee, S.-J., 2011. Method to extract communication history in instant messenger. J. Korea Inst. Inf. Secur. Cryptol. 21 (2), 49–60.
Microsoft Corp, 1999. OLE Concepts and Requirements Overview. http://support.microsoft.com/kb/q86008. (Accessed 24 September 2016).
Neuman, B.C., Ts'o, T., 1994. Kerberos: an authentication service for computer networks. IEEE Commun. Mag. 32 (9), 33–38.
Park, K., Kim, H., 2015. Encryption is not enough: inferring user activities on KakaoTalk with traffic analysis. In: International Workshop on Information Security Applications, pp. 254–265.
Park, S., Oh, D., Lee, B.G., 2011. Analyzing user satisfaction factors for instant messenger-based mobile SNS. In: Future Information Technology. Springer, pp. 280–287.
Rivest, R., April 1992. The MD5 Message-Digest Algorithm. RFC 1321. MIT Laboratory for Computer Science and RSA Data Security, Inc.. https://tools.ief.org/rfc/rfc1321.txt. (Accessed 2 November 2016)
Rogaway, P., 2004. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp. 16–31.
Shin, D., Choi, Y., Lee, Y., Park, S., Kim, S., Won, D., 2007. Security analysis on the authentication mechanisms of Korean popular messengers. In: International

Conference on Universal Access in Human-Computer Interaction. Springer, pp. 547–552.
Smith, C., 2017. 18 Amazing QQ Statistics (December 2016). https://expandedramblings.com/index.php/qq-statistics/. (Accessed 22 September 2017).
Tencent, Q.Q., 2017. Product & Services. https://www.tencent.com/en-us/system.html. (Accessed 22 September 2017).
Thakur, N.S., 2013. Forensic Analysis of WhatsApp on Android Smartphones. Master's thesis. University of New Orleans.
Wheeler, D.J., Needham, R.M., 1994. TEA, a tiny encryption algorithm. In: International Workshop on Fast Software Encryption. Springer, pp. 363–366.
Wikipedia, 2017. Tencent QQ. https://en.wikipedia.org/wiki/Tencent_QQ. (Accessed 22 September 2017).
You, Z.-Q., Han, X.-P., Lü, L., Yeung, C.H., 2015. Empirical studies on the network of social groups: the case of Tencent QQ. PLoS One 10 (7) e0130538.
Yu, F., Zhao, X., Ji, Q., Zhang, L., 2014. Security analysis of mobile QQ. In: Wireless Communications and Signal Processing (WCSP), 2014 Sixth International Conference on. IEEE, pp. 1–5.