



DEGREE PROJECT IN INFORMATION AND COMMUNICATION  
TECHNOLOGY,  
SECOND CYCLE, 30 CREDITS  
*STOCKHOLM, SWEDEN 2016*

# **A Security and Privacy Audit of KakaoTalk's End-to-End Encryption**

**DAWIN SCHMIDT**

KTH Royal Institute of Technology

School of Electrical Engineering

Double Degree Programme in Security and Mobile Computing  
(NordSecMob)

ABSTRACT OF  
MASTER'S THESIS

<b>Author:</b>	Dawin Schmidt		
<b>Title:</b>	A Security and Privacy Audit of KakaoTalk’s End-to-End Encryption		
<b>Date:</b>	September 30, 2016	<b>Pages:</b>	165
<b>TRITA:</b>	TRITA-EE 2016:131	<b>Code:</b>	EP248X
<b>Supervisors:</b>	Panos Papadimitratos N. Asokan		
<b>Advisor:</b>	Ronald Deibert		
<p>End-to-end encryption is becoming a standard feature in popular mobile chat applications (apps) with millions of users. In the two years a number of leading chat apps have added end-end encryption features including LINE, KakaoTalk, Viber, Facebook Messenger, and WhatsApp.</p> <p>However, most of these apps are closed-source and there is little to no independent verification of their end-to-end encryption system design. These implementations may be a major concern as proprietary chat apps may make use of non-standard cryptographic algorithms that may not follow cryptography and security best practices. In addition, governments authorities may force chat app providers to add easily decryptable export-grade cryptography to their products. Further, mainstream apps have a large attack surface as they offer a variety of features. As a result, there may be software vulnerabilities that could be exploited by an attacker in order to compromise user’s end-to-end privacy. Another problem is that, despite being closed-source software, providers often market their apps as being so secure that even the provider is not able to decrypt messages. These marketing claims may be potentially misleading as most users do not have the technical knowledge to verify them.</p> <p>In this Master’s thesis we use KakaoTalk – the most popular chat app in South Korea – as a case study to perform a security and privacy assessment and audit of its “Secure Chat” opt-in end-to-end encryption feature. Also, we examine KakaoTalk’s Terms of Service policies to verify claims such as “[...] Kakao’s server is unable to decrypt the encryption [...]” from a technical perspective.</p> <p>The main goal of this work is to show how various issues in a product can add up to the potential for serious attack vectors against end-to-end privacy despite there being multiple layers of security. In particular, we show how a central public-key directory server makes the end-to-end encryption system vulnerable to well-known operator-site man-in-the-middle attacks. While this naive attack may seem obvious, we argue that (KakaoTalk) users should know about the strength and weaknesses of a particular design in order to make an informed decision whether to trust the security of a chat app or not.</p>			
<b>Keywords:</b>	Secure Messaging, End-to-end Encryption, Android Security		
<b>Language:</b>	English		

Kungliga Tekniska högskolan

Skolan för elektro- och systemteknik

Double Degree Programme in Security and Mobile Computing SAMMANDRAG AV  
(NordSecMob) DIPLOMARBETET

**Utfört av:** Dawin Schmidt

**Arbetets namn:**

A Security and Privacy Audit of  
KakaoTalk's End-to-End Encryption

**Datum:** 30 september 2016

**Sidantal:** 165

**TRITA:** TRITA-EE 2016:131

**Kod:** EP248X

**Övervakare:** N. Asokan  
Panos Papadimitratos

**Handledare:** Ronald Deibert

End-to-end kryptering är en allt mer vanligt förekommande funktionalitet bland populära mobila chatttjänster (händanefter appar) med miljontals användare. Under de två senaste åren har många ledande chattappar, bland annat LINE, KakaoTalk, Viber, Facebook Messenger, och WhatsApp, börjat använda end-to-end kryptering.

Dock så är de flesta av dessa appar closed-source och det finns begränsad, eller ingen, fristående granskning av systemdesignen för deras end-to-end kryptering. Dessa implementationer kan innebära en stor risk då proprietära chattappar kan använda sig av kryptografiska algoritmer som inte följer best practice för säkerhet eller kryptografi. Vidare så kan statliga myndigheter tvinga de som tillhandahåller chattappar att använda lättdekrypterad export-grade kryptografi för sina produkter. Lägg till det att de flesta vanliga appar har många ytor som kan attackeras, till följd av all funktionalitet de erbjuder. Som ett resultat av detta finns en risk för mjukvarubrister som kan utnyttjas av en hackare för att inkräkta på en användares end-to-end integritet. Ytterligare ett problem är att trots att det är closed-source mjukvara så marknadsför ofta appleverantörerna sina appar som att vara så säkra att inte ens leverantörerna själva kan dekryptera användarnas meddelanden. Det som hävdas i marknadsföringen riskerar vara missledande eftersom de flesta användarna inte har den tekniska kunskap som krävs för att kunna verifiera att det som hävdas är sant.

I den här Master-uppsatsen använder vi KakaoTalk – den mest populära chattappen i Sydkorea – som en fallstudie för att granska och bedömma säkerhetens- och integritetsaspekterna hos deras valbara "Secure Chat" med end-to-end krypteringsfunktionalitet. Vi granskar även KakaoTalk's användarvillkor för att kunna verifiera påståenden som att "[...] Kakao's server is unable to decrypt the encryption [...]" från ett tekniskt perspektiv.

Det huvudsakliga syftet med denna studien är att belysa hur olika brister i en produkt sammantagna kan skapa en risk för allvarliga vektorattacker mot end-to-end integriteten även fast det finns flera skyddslager. Mer specifikt visar vi hur en central katalogserver för public-keys gör end-to-end krypteringssystemet sårbart mot välkända operator-site man-in-the-middle-attacker. Trots att denna naiva typ av attack kan verka uppenbar, argumenterar vi för att (KakaoTalk) användare borde veta om styrkorna och svagheter med en särskild systemdesign för att kunna göra ett informerat val för om de ska lita på säkerheten hos en chattapplikation eller inte.

**Nyckelord:** Secure Messaging, End-to-end Encryption, Android Security

**Språk:** Engelska

# Preface

All of the work presented in this Master’s thesis was conducted in the Citizen Lab at the University of Toronto, Canada. Several persons have contributed academically and practically to this study:

The Android security assessment checklist in Appendix A.1 was primarily compiled by myself, except for Section A.1.6, which was contributed by Jeffrey Knockel. The end-to-end encryption assessment checklist in Appendix A.2 was authored by Jedidiah Crandall, with assistance and comments from myself (e.g., Section A.2.8 was mainly contributed by myself). The rest of this Master’s thesis including scripts, documents, experimental setup, and data is my original work which is openly available on <https://github.com/stulle123>.

I would like to thank the staff and fellows from the Citizen Lab including Ronald J. Deibert, Masashi Crete-Nishihata, Jakub Dalek, Christopher Parsons, Sarah McKune, Irene Poetranto, Andrew Hiltz, Jedidiah Crandall, Antonio Espinoza and Jeffrey Knockel for their constructive comments and feedback throughout the entire research process. Furthermore I would like to thank Kelly Kim from Open Net Korea for her extensive feedback on Section 3 of this work.

Finally I would also express my gratitude to my two supervisors N. Asokan and Panos Papadimitratos for their thoughtful feedback which helped me to improve this Master thesis.

Stockholm, September 30, 2016

Dawin Schmidt

# Abbreviations and Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
APK	Android Application Package
CA	Certificate Authority
CCA	Chosen Ciphertext Attack
E2E	End-to-End
EdDSA	Edwards-curve Digital Signature Algorithm
ELF	Executable and Linkable Format
HMAC	keyed-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over TLS
IM	Instant Messaging
IP	Internet Protocol
IPC	Inter-Process Communication
ISP	Internet Service Provider
IV	Initialization Vector
JS	JavaScript
KCSC	Korea Communications Standards Commission
KPA	Known Plaintext Attack
MITM	Man-In-The-Middle
MSIP	Ministry of Science, ICT and Future Planning
NX	No-eXecute
OS	Operating System
PBKDF2	Password-Based Key Derivation Function 2
PFS	Perfect Forward Secrecy
PII	Personally Identifiable Information
PIN	Personal Identification Number
RSA	Rivest-Shamir-Adleman
SDK	Software Development Kit
SHA	Secure Hash Algorithm

SIM	Subscriber Identity Module
SMS	Short Message Service
SOP	Same-Origin Policy
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ToS	Terms of Service
UUID	Universally Unique Identifier
XSS	Cross-Site Scripting

# Contents

<b>Abbreviations and Acronyms</b>	<b>5</b>
<b>1 Introduction</b>	<b>11</b>
1.1 Problem Statement . . . . .	12
1.2 Evaluation Requirements and Techniques . . . . .	14
<b>2 Background</b>	<b>16</b>
2.1 Secure Instant Messaging . . . . .	16
2.2 Android's Security Model . . . . .	21
2.2.1 Android's System Architecture . . . . .	21
2.2.2 Android's Security Model . . . . .	24
2.2.3 Terminology of Attack Types . . . . .	27
2.3 KakaoTalk Application Overview . . . . .	30
<b>3 Kakao Terms of Service Analysis</b>	<b>32</b>
3.1 Methodologies . . . . .	32
3.2 Political and Historical Context of Communications Censorship and Surveillance in South Korea . . . . .	33
3.2.1 Internet Censorship . . . . .	34
3.2.2 Communications Surveillance . . . . .	35
3.2.3 A Timeline of Surveillance Scandals in South Korea and Their Implications on KakaoTalk . . . . .	37
3.2.3.1 2014 . . . . .	37
3.2.3.2 2015 . . . . .	41
3.2.3.3 2016 . . . . .	42
3.2.4 Conclusion . . . . .	43
3.3 Analysis of Kakao's Public Security and Privacy Statements . . . . .	44
3.3.1 Security Promises . . . . .	44
3.3.1.1 Encryption . . . . .	44
3.3.1.2 Data and Identity Protection . . . . .	45
3.3.2 Privacy Promises . . . . .	46

3.3.2.1	Collection of User Information . . . . .	46
3.3.2.2	Collection of User Information from Third- Parties . . . . .	48
3.3.2.3	Sharing of User Information . . . . .	48
3.3.2.4	User Control Over Information Collection and Sharing . . . . .	50
3.3.2.5	User's Access to Their Own Information . . .	50
3.3.2.6	Retention of User Information . . . . .	51
3.3.2.7	Spam Protection . . . . .	52
3.3.3	Summary of Resulting Research Questions . . . . .	53
<b>4</b>	<b>Technical Analysis</b>	<b>56</b>
4.1	Methodologies and Tools . . . . .	56
4.1.1	Assessment Checklists . . . . .	57
4.1.2	Experimental Setup . . . . .	59
4.1.3	Automated Analysis . . . . .	61
4.1.4	Manual Analysis . . . . .	62
4.1.4.1	Static Analysis . . . . .	63
4.1.4.2	Dynamic Analysis . . . . .	64
4.2	Information Gathering . . . . .	67
4.2.1	Application Details . . . . .	68
4.2.2	Mobile Data . . . . .	72
4.2.3	Messaging System Overview . . . . .	75
4.2.3.1	The LOCO Messaging Protocol . . . . .	77
4.2.3.2	Device Registration and Login . . . . .	79
4.2.3.3	The LOCO End-to-end Encryption Messag- ing Protocol . . . . .	83
4.3	Threat Analysis . . . . .	90
4.3.1	Attack Surface . . . . .	90
4.3.2	Assets . . . . .	93
4.3.3	Threat Agents . . . . .	94
4.3.4	Threats . . . . .	95
4.4	Vulnerability Analysis and Findings . . . . .	99
4.4.1	End-to-end Encryption Protocol Analysis . . . . .	100
4.4.2	Android Software Security Analysis . . . . .	105
4.4.3	Information Privacy Analysis . . . . .	113
4.5	Comparison Between Terms of Service Claims and Technical Findings . . . . .	115

<b>5</b>	<b>Evaluation</b>	<b>119</b>
5.1	Evaluation for Solution 1 . . . . .	119
5.2	Evaluation for Solution 2 . . . . .	123
<b>6</b>	<b>Discussion</b>	<b>125</b>
6.1	Significance of Our Results . . . . .	125
6.1.1	Terms of Service Claims Versus Real-World Technical Findings . . . . .	125
6.1.2	End-to-end Encryption Alone is not Sufficient . . . . .	126
6.1.3	Recommendations for High-risk Users . . . . .	127
6.2	Reflection on Our System Evaluation . . . . .	127
6.3	Future Work . . . . .	128
6.4	Responsible Disclosure and Notification . . . . .	130
<b>7</b>	<b>Conclusion</b>	<b>131</b>
<b>A</b>	<b>Security Assessment Checklists</b>	<b>148</b>
A.1	Checklist 1: Questions to Answer About Software Security in Any Given Android Application . . . . .	148
A.1.1	Questions Related to Insufficient Transport Layer Protection . . . . .	148
A.1.2	Questions Related to Access Control and Information Leakage . . . . .	149
A.1.3	Questions Related to Cryptography . . . . .	150
A.1.4	Questions Related to Improper Authentication Proce- dures . . . . .	151
A.1.5	Questions Related to Improper Data Validation . . . . .	153
A.1.6	Questions Related to Software Updates . . . . .	153
A.1.7	Miscellaneous Questions . . . . .	154
A.2	Checklist 2: Questions to Answer About End-to-end Encryp- tion in any Given Chat Application . . . . .	154
A.2.1	Questions Related to Marketing Claims, ToS, and EULA	155
A.2.2	Questions Related to Random Number Generation . . . . .	155
A.2.3	Questions Related to Symmetric Cryptography . . . . .	155
A.2.4	Questions Related to Asymmetric Cryptography . . . . .	156
A.2.5	Questions Related to Hash Functions or MACs . . . . .	157
A.2.6	Questions Related to Traffic Analysis . . . . .	157
A.2.7	Questions Related to Forensics . . . . .	158
A.2.8	Miscellaneous Questions . . . . .	158

<b>B</b>	<b>Application Information</b>	<b>160</b>
B.1	Permissions . . . . .	160
B.2	Application Details . . . . .	164

# Chapter 1

## Introduction

End-to-end encrypted instant messaging is becoming a standard feature in many popular mobile chat applications. In April 2016, WhatsApp, (as of September 30, 2016 the world's most popular messaging service), announced to use end-to-end encryption for all messages including multimedia messages and group chats by default [91]. Only one month later, Google released a new messaging application called "Allo" that also supports end-to-end encryption as an additional opt-in feature [54]. WhatsApp and Allo are using the *Signal* protocol which is a well known and well tested end-to-end encryption messaging protocol [53]. However, even though the protocol is open-source and available for independent review, the two messaging applications are not. In addition, both messengers either added end-to-end encryption retrospectively or do not use end-to-end encryption by default. This might be an indicator that messaging services added end-to-end encryption on top of their existing infrastructure without redesigning their systems from scratch. However, without a system redesign an end-to-end encryption system may be vulnerable due to weaknesses in the underlying existing messaging system. This may be a major issue as many of these applications have large user bases and in some cases dominate countries or regions as, for example, in South Korea.

South Korea is an interesting country to study the adoption of end-to-end encrypted messaging services. It is one of the most connected countries [107, 128] in the world but also a highly regulated environment with a long history of strict government control. Internet users face censorship and surveillance rooted in traditional social values and political tensions with North Korea [122].

Released as South Korea's first mobile instant messenger in March 2010, *KakaoTalk* is dominating the Korean market today. The application, which is available in 230 countries and 15 different languages, has a 200 million user base with almost 50 million monthly active users worldwide [76, 116].

In South Korea, KakaoTalk is the number one messaging application with a market penetration of 97 percent and more than 38 million monthly active users [26].

In recent years, the South Korean government authorities especially targeted KakaoTalk to eavesdrop on users' communications [34]. Concerns over privacy and security of KakaoTalk peaked in 2014, after president Park Geun-Hye announced a crackdown on government criticisms which were spread over the messaging service. As part of the investigations, a large number of KakaoTalk messages of critics and protesters were seized and searched. Shortly after KakaoTalk's cooperation with government agencies came to public, a large number of users moved to Telegram, which they perceived as more secure. Following these events, KakaoTalk released a "Secret Chat" end-to-end encryption feature in December 2014.

## 1.1 Problem Statement

End-to-end message encryption is an increasingly popular feature in mobile chat applications which have millions of users. Many of these Instant Messaging (IM) applications are closed-source and there is little to no independent verification of their end-to-end (E2E) encryption system design. Proprietary IM client and server software and any IM chat protocol libraries may use non-standard cryptographic algorithms that may not follow cryptography and security best practices. As a result, there may be vulnerabilities that could be exploited by an attacker in order to compromise user's E2E privacy. Additionally, governments or intelligence agencies may force IM service providers to add backdoors to their proprietary products. Despite being closed-source software, messaging service providers often market their applications as being so secure that even the service provider is not able to decrypt messages. These marketing claims may be potentially misleading as most users do not have the knowledge to verify them from a technical perspective. Therefore, high-risk or privacy-concerned users may find it hard to make an informed decision on whether to trust a provider's end-to-end encryption or not. Given this problem, we ask the following research question:

*When security and cryptography best practices are not followed in end-to-end encryption of KakaoTalk, would the result be the potential for practical attacks?*

KakaoTalk serves as an interesting case study for exploring our general research question due to a number of reasons:

- *Market dominance:* The application dominates the South Korean market with a user base of 38 million monthly active users in a country that has a population of roughly 50 million people.
- *Security measures:* Kakao, the company behind KakaoTalk, invests numerous efforts into the security and privacy of its products. For instance, the company holds a ISO/IEC 27001 security certificate and follows secure coding guidelines [32].
- *User demand:* End-to-end encryption was added to KakaoTalk as a “Secret Chat” feature retrospectively upon user demand.
- *Social and political context:* Even though South Korea has some of the strictest data privacy laws in Asia [56], it is also a country with a long history of strict government authorities.
- *Closed-source software:* Similar to most other end-to-end chat applications, KakaoTalk including its end-to-end messaging protocol is closed-source.
- *No independant code audit:* To the best of our knowledge, previous research that technically analyzed KakaoTalk’s “Secret Chat” feature, does not exist.

Choosing KakaoTalk as our case study poses the following challenges:

- *Closed-source software:* How do we reverse-engineer a mobile application given that the source-code is not available?
- *Finding attack vectors:* Which attack vectors exist to possibly compromise an end-to-end encrypted conversation?
- *Threat simulation:* Given a discovered vulnerability, how do we prove that the vulnerability can be practically exploited?

This Master thesis tackles the above challenges and provides an answer to our research question by making the following contributions:

### Context and Terms of Service Analysis

- We provide a detailed survey of the political and historical context of communications censorship and surveillance in South Korea and show their particular implications on KakaoTalk’s Terms of Service and privacy policy (Section 3.2 on page 33).

- We provide an in-depth examination of Kakao’s Terms of Service and privacy policy. We contribute a summary of the most important claims and divide them into several privacy and security categories (Section 3.3 on page 44). Finally, we derive a number of research questions which we will later use to compare our technical findings against Kakao’s marketing claims.

### **Technical Analysis**

- We explain our methods and toolchain that we used to reverse-engineer and analyse a closed-source mobile Android application (Section 4.1 on page 56).
- We list a number of possible weaknesses of Android end-to-end chat applications by compiling two distinct security assessment checklists (Section 4.1.1 on page 57).
- We provide a detailed description of KakaoTalk’s messaging system architecture and end-to-end encryption protocol (Section 4.2 on page 67).
- We perform a threat analysis of KakaoTalk’s messaging system by analyzing the system’s assets and attack surface, as well as multiple threat agents and possible system threats (Section 4.3 on page 90).
- We show that KakaoTalk’s end-to-end encryption system is vulnerable to man-in-the-middle (MITM) attacks on the operator-side (Section 4.4.1 on page 100). We also outline a number of software vulnerabilities in KakaoTalk 5.5.5 which may be used to compromise end-to-end encrypted user chat (Section 4.4.2 on page 105).

### **Comparison of Technical Findings and Kakao’s Terms of Service Claims**

- We compare our technical results against Kakao’s Terms of Service claims to show if what KakaoTalk is doing conforms or not with what Kakao states in its policies (Section 4.5 on page 115).

## **1.2 Evaluation Requirements and Techniques**

As described in the previous section, our main contribution of this work will be a security assessment and terms of service analysis of an end-to-end

encryption system. Following, we describe the requirements and techniques that we use in Section 5 on page 119 to evaluate our work:

**Goal 1:** Our security assessment is “profound” or “complete” in the sense that it includes the most important security analysis aspects.

**Expected solution 1:** A detailed technical security analysis of the Android version of KakaoTalk including its end-to-end encryption protocol in order to identify strength and weaknesses. The analysis includes the assessment of TLS and inter-process communications, as well as of end-to-end encrypted user chat. The security exercise uses methods including information gathering, static and dynamic analysis, reverse engineering, forensic analysis, and other techniques. Critical vulnerabilities (if any) should allow an attacker to read users’ end-to-end encrypted messages.

**Evaluation requirements:** The security analysis should identify security goals for the target application (KakaoTalk) and make a clear statement about how well the target application meets a particular goal. For any goal that is not met, the analysis should clearly explain why, and preferably include a Proof-of-Concept (PoC) exploit.

**Evaluation technique:** Comparative study.

**Goal 2:** KakaoTalk end users should know about the strength and weaknesses of the particular end-to-end encryption design that Kakao has chosen. With that knowledge they should be able to make an informed decision whether they want to trust Kakao or not.

**Expected solution 2:** A higher-level representation of the main outcomes of our work, easy to understand for an ordinary KakaoTalk user. We will come up with a scorecard, graph, website, or report that shows the strength and weaknesses in a clear and precise format.

**Evaluation requirements:** After consulting our higher-level report, users should be able to make an informed decision whether they want to trust Kakao or not.

**Evaluation techniques:** User study as a controlled experiment, cognitive walkthrough with an usability expert or qualitative feedback surveys.

## Chapter 2

# Background

This chapter provides some preliminary material for the reader in order to understand the technical content that follows in Chapter 4. As we will discuss in Chapter 3, one of Kakao’s marketing claims is that KakaoTalk enables secure instant messaging. Section 2.1 familiarizes the reader with the term of secure instant messaging and points out its important properties. Section 2.2 describes the fundamentals of Android’s system architecture and security model. We conclude by briefly introducing KakaoTalk in Section 2.3. Finally, we expect the reader to be familiar with basic cryptography concepts. For references, please see [110] or [96].

### 2.1 Secure Instant Messaging

Many of the popular Instant Messaging (IM) services today such as Microsoft’s Skype rely on the Transport Layer Security (TLS) protocol to encrypt private user chat. This approach may be secure against passive attackers, but may not protect against active man-in-the-middle attacks or eavesdropping on the IM service provider’s side. Further TLS weaknesses have been highlighted by recent Certificate Authority (CA) scandals [130, 85].

As the Snowden revelations have confirmed, intelligence agencies and governments are spying on IM conversations. For instance, in 2008, the National Security Agency (NSA) had access to up to 60,000 online sessions of Yahoo’s Web-Messenger per day [46]. In other cases, the NSA demanded access to internal IM service providers’ networks [111] or forced companies to provide chat history records [68], e.g., by the means of national security letters. Government agencies have also asked IM service providers to cooperate in message interception [114] or to add backdoors to their chat products [103].

Worse, intelligence agencies are not the only eavesdropper on IM communications. Often IM service providers themselves monitor encrypted or unencrypted network traffic for several purposes such as targeted advertising. This can be achieved by applying traffic analysis techniques to learn about who is talking to whom, from where and for how long [36]. In addition, the IM service provider may be able to gather information about users' actions and the actual length or language of particular messages. For example in the case of KakaoTalk, actions such as viewing other user's profiles – which could be an indicator for stalking – can be identified with 99.7 percent accuracy [98].

Following the Snowden leaks, there has been a rapid growth of IM chat protocols and applications that provide secure instant messaging by using techniques beyond Transport Layer Security. For instance, there is the Signal protocol which has been recently integrated into Facebook's Messenger [90] or the Off-The-Record (OTR) protocol which has been implemented by open-source IM applications such as Pidgin and ChatSecure.

However, most of these chat programs define their own set of secure IM rules and implement features differently. This is due to the fact that there is no open protocol standard for secure messaging which software developers could follow. Another reason is that major non-profit organizations that publish secure IM guidelines including the Electronic Frontier Foundation (EFF) and the Free Software Foundation (FSF) have contradicting opinions about secure IM, e.g., the FSF [49] promotes a more radical approach to secure IM than the EFF [48]. The last reason is that the term of secure instant messaging has yet not been defined and there is no consensus about which security and privacy features it should incorporate. The implications of these issues are that most secure IM solutions are either entirely proprietary (e.g., Threema) or are open-source but incompatible to each other, e.g., a Signal user cannot talk to a (OTR-enabled) Pidgin user.

Secure IM solutions are commonly modeled on two different communication architectures: client-to-server and client-to-client (peer-to-peer). In the client-to-server communication model, an IM server, which is typically hosted by an IM service provider, manages the availability of the clients and relays messages between them (store-and-forward). For authentication, each client shares a secret (e.g., a password) with the service provider. In addition, the IM server provides necessary information, such as the end-points' IP addresses, in case two clients communicate with each other in a peer-to-peer fashion (e.g., for audio/video calls or file transfers). Most public IM solutions use the client-to-server communication model in order to allow asynchronous messaging, i.e., a sender and a recipient do not need to be online at the same time to exchange messages. There are also fully decentralized peer-to-peer

IM networks in which a central IM server is not necessary (e.g., BitTorrent’s Bleep). These peer-to-peer IM networks are examples for synchronous messaging systems in which a sender needs to directly connect to an intended recipient before sending messages.

Unger et al. [126] identify three aspects of a secure instant messaging system: Trust Establishment, Conversation Security, and Transport Privacy. Trust establishment refers to the process where users exchange and authenticate any long-term key material. After a user adds another user to its buddy list, both parties must first go through the process of trust establishment before they engage in a secure conversation. After trust establishment has been achieved, a conversation security protocol ensures the security and privacy of a secret conversation. Eventually, the aspect of transport privacy defines how well any communication metadata is protected during encrypted message exchange.

In this work, we require a private and secure online chat to have the same privacy level as a face-to-face or “off-the-record” [10] conversation between two human beings<sup>1</sup>:

**Confidentiality:** Confidentiality guarantees that nobody else is able to listen to Alice’s and Bob’s private conversation. Secure IM chat protocols must encrypt the communication end-to-end in order to ensure confidentiality. The usage of end-to-end encryption requires that only Alice and Bob know about the encryption keys that are used to secure the communication channel.

**Perfect Forward Secrecy:** Perfect Forward Secrecy (PFS) [58] ensures that it is impossible for someone else to find out what Alice and Bob talked about after their conversation has happened even if Alice’s and/or Bob’s long-term secret key material are compromised by the adversary. Typically, this is achieved by short-term encryption/decryption keys that are generated on-the-fly, thrown away after usage, and are impossible to derive from any long-lived key material.

**Backward or “Future” Secrecy:** In a scenario where Eve obtained access to a long-term decryption key and *passively* listens to Alice’s and Bob’s encrypted conversation, this feature prevents her to read all subsequent messages that Alice and Bob will exchange in the future. However, secure IM

---

<sup>1</sup>These are just the fundamental security and privacy properties of a secure messaging system, for a more detailed evaluation framework refer to the survey of Unger et al. [126].

solutions with backward secrecy are still vulnerable by *active* attackers with access to any long-term key material.

**Authentication:** Another important property of private communications is entity and message authentication. Alice must be sure that she is really talking to Bob and not to someone else pretending to be Bob (entity authentication). For entity authentication a number of mechanisms exist, e.g., a secure chat application may show each entity’s public key fingerprint that could be then compared over an out-of-band channel. Typically, digital signatures or message authentication codes are used for message authentication.

**Offline Message Deniability:** After having received an authenticated message from Alice, offline message deniability or repudiability ensures that Bob cannot later prove to Eve that such a particular message was indeed send by Alice [13]. That means that Alice can deny the fact that she sent a particular message to Bob. For example, Alice might assume that her friend Bob is an FBI informant who secretly keeps track of all her messages [132]. Therefore, Bob should not be able to prove to a court of law that Alice sent any particular messages. If Alice and Bob use message authentication codes for message authenticity, both of them share the same symmetric key. Thus, Bob could have also created a fake message in Alice’s name, making it impossible for him to provide valid proof. This concept is known as “weak” offline message deniability/repudiability [10]. “Strong” (or plausible) offline message deniability or forgeability [10] is a related property, which assumes that not only Bob but everyone who listens to the communication channel can create fake messages, e.g., by using a malleable encryption scheme such as a stream cipher. As a result, Alice can deny having sent a certain message because anyone else could have created fake messages using her name. Finally, there is also the concept of offline participation deniability/repudiation in which Alice can deny the fact talking to Bob at all. As pointed out by Unger et al. the idea of *online* deniability/repudiation [39] will not be considered in this work due to incompatible definitions in the academic literature.

Whether message deniability is a useful feature or not is an active topic of discussion. For instance, there have not been any cases of deniable messaging properties influencing legal court decisions. Also, there may be cases in which message deniability is not useful, e.g., in situations where the mere suspicion of sending or receiving messages is harmful to an author. However, we agree with Unger et al. that secure IM conversations should still have the property of deniability.

**Metadata protection:** Metadata protection (“untraceability”) ensures that no one except Alice and Bob knows about the fact that they engage in a personal conversation. Anonymous instant messaging is about hiding metadata and complicating encrypted traffic analysis techniques such as timing and statistical attacks (“unobservability”). Accessible metadata could be otherwise easily correlated with other records to identify individuals. In many cases metadata is sufficient to either convict [6] or even harm individuals, e.g., the U.S. is using metadata to justify targeted drone strikes [109, 87]. Metadata protection can, for example, be achieved by decentralized architectures including peer-to-peer networks or by onion routing technologies such as Tor. When a secure messaging system utilizes these decentralized architectures it is crucial that the application/protocol does not leak any sensitive information that could potentially deanonymize a user. As for now, there is no practical secure and anonymous IM system that is resistant against global adversaries who control and monitor large network segments [126].

Apart from the protocol properties mentioned above, there are also more general criteria that are crucial for assessing secure IM chat protocols and applications (again, these are just the ones we consider as most important):

**Open-source IM software:** Proprietary IM client and server software and any IM chat protocol libraries may use non-standard cryptographic algorithms that may introduce software vulnerabilities. These vulnerabilities may then be exploitable in order to compromise user’s privacy. Additionally, governments or intelligence agencies may force IM service providers to add backdoors to their proprietary products. For these reasons, secure IM software must be open-source. If the IM client and server software and any IM chat protocol libraries are open-source, they are available to the public that may detect malicious behaviour.

**Open documentation:** Whether the secure IM chat protocol and application is well documented, complete, and publicly available. The documentation must be up-to-date and easily accessible.

**Recent independent review:** Whether the chat protocol design and the application source code of a secure IM solution have been audited independently. The full audit must be open to the public, and application/protocol maintainers must publish which of the proposed improvements have been integrated and which security bugs have been fixed as a result of the audit<sup>2</sup>.

---

<sup>2</sup>See the Electronic Frontier Foundation’s post on what makes a good security audit: <https://www.eff.org/deeplinks/2014/11/what-makes-good-security-audit>

## 2.2 Android’s Security Model

This section will cover Android’s system architecture (Section 2.2.1) and security model (Section 2.2.2) at a higher-level in order to better understand the security analysis of KakaoTalk in Chapter 4. The content and structure of the following discussion is mostly based on Nikolay Elenkov’s book “Android Security Internals” [42].

### 2.2.1 Android’s System Architecture

Android’s system architecture and fundamental parts can be best described by using a layered diagram starting from the bottom up (see Figure 2.1):

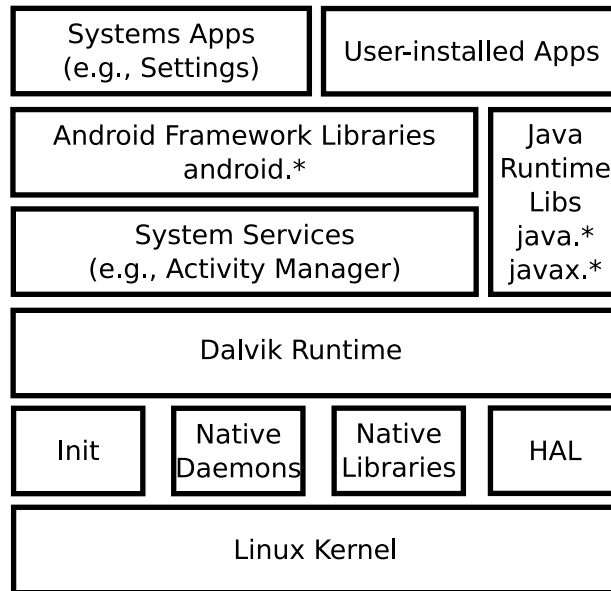


Figure 2.1: Android’s system architecture. Figure adapted from [42].

**Linux Kernel:** The Linux kernel provides hardware drivers, network and file system access, process management and other low-level system services. Android’s Linux kernel is different from any “standard” Desktop-PC Linux kernel. It supports additional features or “Androidisms” [135] that are necessary for mobile devices, e.g., features such as *Binder* for inter-process communication (IPC).

**Native User Space:** On top of the Linux kernel is the native user space which consists of the Init process (first process that starts all other processes), the Hardware Abstraction Layer (HAL), as well as native libraries and daemons.

**Dalvik Virtual Machine:** Apart from native libraries and daemons that are written in C/C++, the majority of Android source code is written in Java that needs a Java Virtual Machine (JVM) to be executed. The JVM in Android is called Dalvik Virtual Machine (DVM) which is not capable of executing “standard” Java bytecode (\*.class files), but Dalvik Executable bytecode (\*.dex files). Dalvik Executable bytecode is either packaged inside Java libraries (\*.jar files) or inside Android applications (\*.apk files). In addition, Dalvik Executable bytecode can be converted into device-dependant Optimized Dalvik Executable bytecode (\*.odex files) for performance optimizations. As from Android 5.5, the DVM was replaced by the Android Runtime (ART) which remains to use Dalvik Executable bytecode due to backward compatibility reasons.

**Java Runtime Libraries:** Most of the system services and applications require Java runtime libraries to import core functionalities (java.\* and javax.\* packages). Some Java runtime libraries call native C/C++ code by using the Java Native Interface (JNI) which also allows C/C++ code to call Java code.

**System Services:** System services implement an object-oriented operating system on top of the Linux kernel and provide fundamental Android features such as telephony or touchscreen support. Typically, each system service offers a remote interface that can be accessed by other services and applications. In Android, Binder implements the discovery, mediation and inter-process communication of system services.

**Binder:** Android uses a combination of the Binder kernel driver and the Binder userspace library to implement IPC. Binder assures that the User ID and the Process ID of the calling process cannot be forged. Binder’s security features will be more discussed in Section 2.2.2.

**Android Framework Libraries:** Android framework libraries include Java libraries that are not part of the standard Java runtime, e.g., base classes for Android application components such as Activities, Services or Content Providers (android.app.\* packages). The framework libraries also

include facade classes (“managers” or “Binder IPC proxies”) through which user-installed application access system services, e.g., the `ActivityManager` is a facade class of the `ActivityManagerService`.

**Applications or “Apps”:** Android applications can be divided into system and user-installed applications. On the one hand, system applications have more privileges than user-installed applications and are typically pre-installed under the `/system` or `/system/priv-app` directories to which a user has read-only access. In addition, system applications can be replaced by updates that are signed with the same developer signing key. On the other hand, user-installed applications have their own read/writable private data directory under `/data/data/app-package-name` and run in their own security sandbox in order to guarantee privilege separation.

Android applications consist of a set of loosely coupled components which typically have multiple entry points. These components and their entry points are defined in the application’s `AndroidManifest.xml` manifest file. The most important property defined in the `AndroidManifest.xml` is the application package name that serves as a unique identifier in the system. In order to prevent a malicious application to replace a legitimate application by using the same package name, each application must be signed with a private key that is controlled by the developer. The four different application components are as follows:

**Activities:** An Activity is an application component that provides a User Interface with which users can interact. If the activity has been declared as *exported* in the `AndroidManifest.xml`, it can be also started by other third-party applications.

**Services:** A Service is a component that has no user-interface and runs any long-running tasks in the background. Services can implement a remote interface to allow interaction with other components by using the Android Interface Definition Language (AIDL).

**Content Providers:** Content Providers such as databases allow an Android application to share data with other applications. Fine-grained data sharing permissions allow an application to share only a subset of its data.

**Broadcast Receivers:** Broadcast Receivers are components that respond to system-wide events (called broadcasts). Broadcasts such as “network interface down” can originate from the system or from user applications.

Now that we know about the fundamentals of Android's system architecture we can proceed with Android's security model in the next section.

### 2.2.2 Android's Security Model

In the following paragraphs we will discuss the fundamental properties of Android's security model including sandboxing, SELinux, permissions, IPC, code signing, system updates, and verified boot.

**Android Sandboxing:** Android creates a full kernel-level sandbox by isolating applications both at the process as well as at the file level. In a traditional Linux system, a User ID is typically given to a physical user that can log into the system. However in Android, the physical user is implicit and each application is executed with its own User ID (application ID). This means that each application runs in a dedicated process (process isolation). In addition, every application has its own private read/writable data directory under `/data/data/app-package-name` to which no other application has access to (file system isolation). Finally, there is the concept of a shared user ID in which two applications share the same application ID and run in the same sandbox. This is only possible if both applications have been signed with the same code signing key.

**SELinux:** SELinux is a Mandatory Access Control (MAC) implementation for Android. Traditionally, Android is based on a Discretionary Access Control (DAC) model which means that once an application has access to a particular resource, it can pass the resource to another application at its discretion. For example, an application may set one of its private files to world read/writable to allow other applications to access it. The delegation of private files to other applications would not be possible in a MAC model in which central policies define system-wide authorization rules. As of Android version 4.4, SELinux isolates the core system daemons by enforcing multiple SELinux policies.

**Permissions:** Android follows the principle of least privilege meaning that applications own only as many privileges as necessary. Additional privileges can be granted to applications by using the concept of *Permissions*. For instance, an application can be granted Internet access by giving it the `INTERNET` permission. Permissions are associated with a single Group ID (GID) and are defined in the application's `AndroidManifest.xml`. Prior Android 6.0, permissions are granted at installation time and cannot be revoked

after the fact. Permission checks are either performed by the Linux kernel, the Android OS or by individual application components such as Content Providers. As of version 6.0, Android changed its permission model to one that is similar to Apple's iOS: permissions are no longer assigned at application install-time, but at runtime, and can be also revoked later<sup>3</sup>.

**Inter-Process Communication (IPC):** In most Desktop-PC operating systems such as Microsoft's Windows, user-installed applications can directly interact with the OS kernel, e.g., by initiating system calls. However in Android, applications do not have direct kernel access, but communicate to it via the system services. All inter-process communication (IPC) between applications and system services is handled by Binder (higher-level IPC abstractions include Intents, Messengers, and Content Providers).

Binder is implemented by two major components: a userspace library `libbinder.so` and the Binder kernel driver. The userspace library is loaded into most processes (user-installed applications and system services) and is responsible for marshalling objects into *Parcels*<sup>4</sup>. In addition, the library makes the necessary `ioctl()` system calls with the file descriptor `/dev/binder` as a parameter. The `/dev/binder` device itself is the interface to the Binder kernel driver that handles all of the inter-process communications. The Binder kernel driver maintains a memory chunk of every process. Each particular chunk is read/writable for Binder, but readable-only for the process.

When a process sends a message to another process, the Binder kernel driver copies the message and pastes it into the Binder-maintained memory chunk of the receiving process (recipient). The Binder kernel driver then notifies the recipient that there is new data available. When the recipient has finished processing the data, it notifies the Binder kernel driver which then frees its maintained memory chunk.

When a process calls a function of another process, the Binder kernel driver automatically adds the Process ID (PID) and the Effective User ID (EUID) – which is composed of the Android user ID and the application ID – to the Binder IPC call (Binder transaction). The callee can then decide whether to allow or drop the call by checking the PID and EUID of the caller. This prevents privilege escalation attacks because it is impossible for the calling process to spoof or fake its PID and EUID.

Binder objects that are passed between processes maintain an unique

---

<sup>3</sup>See Android's Developer documentation for how to request permissions at runtime: <http://developer.android.com/training/permissions/requesting.html>.

<sup>4</sup>A Parcel is a container for a message that can be sent through an IBinder interface, see <http://developer.android.com/reference/android/os/Parcel.html>.

identity which guarantees that all calls to the object will always be processed by the same object. The Binder kernel driver maintains a record of which handles correspond to which Binder object in a specific process. For any userspace process it is impossible to create a fake copy of a Binder object or to obtain a reference for it unless they have been assigned a handle through Binder.

As a result, any Binder object is unique and unforgeable, and can act as a security token or Binder token. These tokens in turn act as capabilities and enable a capability-based security model in Android where a capability references the target object or resource and encapsulates a set of access rights to it. The mere possession of a capability/Binder token grants a process full access to another Binder object. In order to discover and access Binder objects, processes can query the *servicemanager* native daemon which is responsible for maintaining all Binder object references.

Besides its useful features, there are also security concerns that are introduced with Binder. One of the major problems is that *all* inter-process communications go through Binder. As a result, sensitive data may leave an application's sandbox when it communicates with another process. The main issue is that transmitted data is typically sent in plain Binder transactions (Parcels) that can be easily sniffed by a man-in-the-middle attacker who has root access to the mobile device. In 2014, Artenstein et al. [3] hooked Binder's userspace library at the point where it issues the `ioctl()` system call to the Binder kernel driver in order to intercept IPC messages. As part of a proof-of-concept, they were then able to get a hold of plaintext data before it was sent over a HTTPS network connection by sniffing IPC messages between an Android application and the Network Manager system service.

**Code Signing and Platform Keys:** Code signing assures that an Android application cannot be replaced by a malicious update that uses the same package name. Thus, code signing guarantees the same-origin policy concept as well as trust relationships between applications. When a user updates an application, the signing certificate of the already installed application is compared against the certificate that is shipped with the update. User-installed applications are typically signed with a developer-controlled key whereas system updates are signed with device-dependant platform keys. Applications and system services are running in the same sandbox if they are signed with the same signing key. However, there have been a number of events where attackers were able to circumvent signature checks by forging a valid signature in order to install malicious updates [51, 50, 52, 134].

**System Updates:** Security updates for Android<sup>5</sup> are typically installed automatically via over-the-air (OTA) updates or manually via the Android Debug Bridge (ADB) interface or other vendor-specific update tools.

**Verified Boot:** Verified boot assures that the system partition and the kernel have not been tampered with. This is achieved by a cryptographic hash tree: Each node in the tree is a cryptographic hash. Leaf nodes contain the hash value of a physical data block, and parent nodes contain the hash value of their child nodes. As a result, the root node is based on all other hash values in the tree and therefore only the root node hash needs to be verified with a RSA public key. This verification step is performed by the kernel whose integrity is also being verified by a signature verification key that is securely stored on the mobile device.

### 2.2.3 Terminology of Attack Types

Android applications can be attacked in many different ways. In order for the reader to fully understand the threat analysis of KakaoTalk in Section 4.3, we hereby provide a list of the most important attack types<sup>6</sup>:

**Remote code execution attacks:** Remote code execution attacks are attacks where a remote adversary is able to execute her own arbitrary code on a victim's mobile device without having physical access. These attacks typically aim to gain root privileges on the target device by exploiting a software vulnerability. For instance, some of the previous root exploits exploited vulnerabilities such as CVE-2013-6282<sup>7</sup> or CVE-2014-3153<sup>8</sup>. The goal of a remote code execution attack is usually to perform some of the following actions: Disclose sensitive user information; capture screenshots; monitor clipboard content; collect passwords from WiFi networks and other (online) accounts; record voice and real-time phone calls using the microphone; collect contacts from address book, SMS, MMS, emails, and chat messages; record location; gather device information; capture pictures using the front or back camera. In order to run a remote code execution attack, an adversary must meet two conditions:

---

<sup>5</sup><https://groups.google.com/forum/#!forum/android-security-updates>

<sup>6</sup>We leave out sophisticated attack types such as hardware side channel attacks. Some of the attack types were taken from David Thiel's book "IOS Application Security: The Definitive Guide for Hackers and Developer" [123].

<sup>7</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-6282>

<sup>8</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3153>

1. *There must be a software vulnerability on the mobile device that can be exploited.* Such a vulnerability could be a memory corruption bug in KakaoTalk such as a stack or heap overflow, a use-after-free or a Java object deserialization bug.
2. *The remote code execution exploit must be delivered to and executed by the victim.* There are multiple channels for an attacker to deliver her remote execution exploit to the user. For instance, an adversary could trick the user to unintentionally execute malicious payload by using a tapjacking attack<sup>9</sup> or by sending a maliciously crafted file attachment via KakaoTalk's user chat. Another delivery method could be the mobile phone's baseband CPU that may allow unauthorized third-party access via the cellular network.

**Web-based attacks:** Many Android applications make use of WebViews that allow an application to display web content. Using web-based attacks, an adversary may be able to steal user data or to fully compromise a user's device by presenting a maliciously crafted HTML and/or JavaScript file. Upon tricking the user to visit a maliciously crafted website, an attacker may be able to execute injection attacks such as client-side cross-site (XSS) scripting, SQL or XML injection, or other injection attacks. These injection attacks may be then used to steal sensitive information from local data storages, read and copy session cookies, or present a fake login website to steal users' login credentials. In 2011, Phil Purviance was able to steal another Skype contact's address book by creating a fake user name which had malicious JavaScript code embedded<sup>10</sup>. When the malicious Skype user name was displayed on another Skype client, the embedded JavaScript executed and sent the victim's address book to a remote server. There are also web-based remote code execution attacks that are more powerful than just stealing a user's private address book. For instance, an exploit created by Hacking Team is able to fully compromise an Android device by just tricking the user to browse to a maliciously crafted website<sup>11</sup>.

**Network-based attacks:** These types of attacks are performed by local or remote adversaries that attack the communication channel of the (KakaoTalk) messaging system. There are passive and active network attacks: In a

---

<sup>9</sup>[https://media.blackhat.com/ad-12/Niemietz/bh-ad-12-androidmarcus\\_niemietz-WP.pdf](https://media.blackhat.com/ad-12/Niemietz/bh-ad-12-androidmarcus_niemietz-WP.pdf)

<sup>10</sup><https://superevr.com/blog/2011/xss-in-skype-for-ios/>

<sup>11</sup>[http://archive.hack.lu/2015/HT\\_Android\\_hack\\_lu2015\\_v1.0.pdf](http://archive.hack.lu/2015/HT_Android_hack_lu2015_v1.0.pdf)

passive network attack, an adversary is passively listening to the communication channel and usually tries to obtain information that is transmitted in cleartext. Active network attackers, on the other hand, try to actively mangle the victim's network traffic by performing a man-in-the-middle attack. Once being the man-in-the-middle possible attacks include: code injection attacks into unencrypted APK file downloads, downgrade attacks against TLS connections, or replay attacks against KakaoTalk's chat messages. Other active network attacks include Denial of Service (DoS) attacks that flood an endpoint with a large number of messages in order to exhaust the endpoint's network link or computing power. This way, an endpoint might become unreachable for other network clients to connect. For example, a DoS attack against a KakaoTalk server could block other KakaoTalk users from participating in the messaging system.

***Inter-process-communication based attacks:*** These types of attacks try to exploit vulnerabilities of the mobile application's inter-process communication (IPC) implementation. Many Android applications do not verify the integrity of incoming IPC messages and are therefore vulnerable to injection attacks. IPC attacks are typically performed by third-party user-installed application that aim to steal sensitive user data but do not have root-privileges. By sending a specially crafted IPC message, a malicious application could perform an SQL injection attack in order to steal user data from the target application's private SQLite database. Other IPC attacks include attacks against exported Activities: If the application's exported Activity is not protected by a restrictive permission, a malicious application could send a IPC message to open the Activity to the foreground. It could then capture a screenshot to steal sensitive information, for example, a private chat conversation.

***Physical access based attacks:*** There are also attacks that require brief or permanent physical access to the user's mobile device. In some cases a brief physical interaction is enough to compromise a victim's mobile device. For instance, an exposed Bluetooth or NFC interface may allow an attacker to deliver her exploit just by passing by. However, in many cases an adversary typically tries to get a permanent hold of the victim's mobile phone. Having permanent physical access to the device, an attacker may be able to access online services that require two-factor authentication. In addition, an adversary may be able to execute a number of "offline" and "online" forensic attacks: In an online forensic attack, an adversary may be able to extract sensitive data from the phone's physical memory because the device is still

powered on. If the mobile phone is powered off, an attacker may be able to examine the phone's unencrypted block storage to extract critical private information such as login credentials.

## 2.3 KakaoTalk Application Overview

Released as Korea's first mobile instant messenger in March 2010, KakaoTalk is one of the global messaging service leaders today. The application, which is available in 230 countries and 15 different languages, has a 200 million user base with almost 50 million active users worldwide [76, 116]. In Korea, KakaoTalk is the number one messaging application with a market penetration of 97 percent and more than 38 million monthly active users [26]. Besides other messengers such as Facebook Messenger more than 70 percent of Koreans use KakaoTalk everyday [41]. The reasons for KakaoTalk's success are several-fold: First, KakaoTalk is freely available and does not cost money. The second reason is KakaoTalk's efficient approach of finding and adding new contacts to the messaging system.

KakaoTalk is based on a platform business model and is transitioning from an IM-only service provider into a mobile social platform [59, 138]. This platform ecosystem consists of a number of additional services which are added to KakaoTalk as part of new version releases or separate mobile applications. These services, which include Kakao Story, Kakao Taxi, or Kakao Game are linked to the KakaoTalk ecosystem via a central Kakao user account. Moreover, Kakao provides a Software Development Kit (SDK) to enable users to develop their own services by using Kakao's APIs.

In contrast to many other mobile instant messenger (IM) services, Kakao has been successful in monetizing KakaoTalk without requiring user fee payments. KakaoTalk's main sources of revenue consist of advertising and e-commerce. As for advertising, KakaoTalk's in-app advertising takes the forms of "Plus Friend", "Yellow ID", and "Brand Emoticons". Plus Friend and Yellow ID are paid corporate accounts whereas Brand Emoticons are brand-specific emoticons. These types of marketing and advertising opportunities are attractive to many corporations due to KakaoTalk's large user community. KakaoTalk's e-commerce business is driven by the application's "Item Store" service through which users can purchase items such as special emoticons or stickers.

KakaoTalk provides a variety of features and is available on most mobile and Desktop operating systems. The mobile messenger runs on mobile operating systems including Apple's iOS, Google's Android and Microsoft's Windows Phone. PC operating systems including Windows and MAC OS

are supported as well. Using a single account, a KakaoTalk user can use both the mobile version and the PC version at the same time in order to sync messages or files<sup>12</sup>. However, multiple mobile devices per account are not supported (see Section 4.2.3.2). The majority of the different KakaoTalk versions supports the following most important features:

**Multimedia messaging and calling:** KakaoTalk provides three different types of both one-on-one and group chats: Open, regular, and private. An open chat room is a public chat room that is searchable and can be joined by everyone. In a regular chat all messages are encrypted with a provider-shared key whereas in a private chat messages are encrypted end-to-end. KakaoTalk’s private “Secret Chat” feature is available on Android and iOS only. While in a chatroom, users can share images, videos, contact information, URLs, location information and other data. However, KakaoTalk does not permit sharing audio or APK files. Finally, the application allows users to participate in SRTP-based video group calls.

**Friend stickers, emoticons, and emojis:** KakaoTalk’s emoticons include friend stickers, animated emoticons as well as sound emoticons featuring popular Korean characters or celebrities. Furthermore, users can customize their KakaoTalk application by choosing own wallpapers, chat bubbles, fonts, or color schemes. Themes which bundle all these customizations can be either created by the user or downloaded as pre-packaged APK files. In order to prevent that a user installs a malicious third-party theme, Kakao provides a service to scan APK files for malicious activities<sup>13</sup>. Emoticons and themes can be purchased through KakaoTalk’s Item Store which provides in-app payments via the Google Play billing service or Kakao Pay<sup>14</sup>.

---

<sup>12</sup><http://www.kakao.com/services/8/pc>

<sup>13</sup><http://www.kakao.com/services/talk/theme>

<sup>14</sup><http://www.kakao.com/kakaopay>

## Chapter 3

# Kakao Terms of Service Analysis

We begin this chapter by describing our methodologies that we used for our context and terms of service analysis (Section 3.1). In Section 3.2 we describe the political situation as well as the communications surveillance and censorship landscape of South Korea. This environment makes it interesting to analyse a popular messaging application such as KakaoTalk. For instance, on the one hand there are strict data privacy laws that protect user’s chat from government eavesdropping. On the other hand, there are significant surveillance and censorship mechanisms that enable communications wiretapping. It is interesting to analyse how Kakao behaves in such an environment and how it runs a messaging service in such a contradicting situation (Section 3.2.3). In Section 3.3 we examine Kakao’s public security and privacy statements. We collect the most important statements, categorize them and derive a number of research questions. These research questions are the main motivation for our technical analysis in Chapter 4 in which we verify Kakao’s claims for correctness.

### 3.1 Methodologies

The main method we applied in this chapter was a rigorous review of existing literature. We also spoke to the non-government organization Open Net Korea which led to valuable input. Nevertheless, the main approach was reading, collecting, and categorizing information from various online sources:

- For our analysis in Section 3.2 we gathered information from the international and Korean press, from non-government organizations and advocacy groups as well as from the research community.
- For KakaoTalk’s marketing claims analysis in Section 3.3 we collected

statements from Kakao’s Terms of Service (ToS), privacy policy, operation policy, transparency report, press releases, FAQs, as well as from news articles.

Our literature review has a number of limitations. First of all, we did not use any “offline” material such as books or articles, but relied on online sources only. Also, we used just one online search engine to search for relevant material. We did not use any other search engines which may would have provided different results. Also, because of language barriers, we mainly gathered sources written in the English language. However, some of our sources included primary sources written in Korean.

## 3.2 Political and Historical Context of Communications Censorship and Surveillance in South Korea

From 1961 to 1987, South Korea was under military dictatorship of Park Chung-hee and Chun Doo-hwan with restrictions on freedom of speech and widespread surveillance. Since then, South Korea has rapidly transitioned from a controlled society into a vibrant democracy and into one of the most connected countries in the world. Not only has South Korea the highest average Internet connection speed globally [8, 115], it also has the highest Internet penetration rates [107, 128] with most Koreans owning more than one mobile phone [127]. The constitution of South Korea guarantees independence of judiciary and freedom of speech, the press, assembly, and association to all citizens. Moreover, the country has one of the strictest data privacy laws in Asia [56].

However, South Korea is also a country with a long history of strict government authorities whose behaviour is rooted in political tensions with North Korea and traditional social values. Over the years, especially since the conservative party returned to power in 2008, the government has employed censorship and surveillance techniques to prosecute citizens for criticizing the government or digital defamation. For instance, criticism on government policies posted on the Internet portal “Agora” between March 2008 and January 2009 led to lawsuits and investigations against a famous Korean blogger<sup>1</sup>.

---

<sup>1</sup>See Wikipedia article on Minerva: [https://en.wikipedia.org/wiki/Minerva\\_%28Internet\\_celebrity%29](https://en.wikipedia.org/wiki/Minerva_%28Internet_celebrity%29).

Moreover, conservatives have used restrictions and other articles in the South Korean constitution as ways to attack freedom of speech. For example, Articles 17 and 18 prohibit the infringement of “privacy” and “privacy of correspondence”, but those guarantees can be bypassed by invoking Article 37 (2), which allows to limit rights “when necessary for national security, the maintenance of law and order or for public welfare”. In addition, restrictions exist in that “neither speech nor the press may violate the honor or rights of other persons nor undermine public morale or social ethics” [84].

Furthermore, laws such as the National Security Act which originates from the military dictatorship period, have been used as tools to justify restrictions and suppression of different aspects of digital activity. As a result, the United Nations Human Rights Council has recommended to delete Article 7 of the National Security Act because the article criminalizes freedom of speech itself without requiring a criminal act [5].

Multiple human rights organizations and advocacies have been criticizing South Korea for its censorship and surveillance practices. For example in 2012, the United Nations Human Rights Council’s Special Rapporteur on Freedom of Expression warned that South Korea’s defamation laws are often used to punish statements “that are true and are in the public interest” [105]. In addition, the South Korean Human Rights Organizations Network criticized in a statement to the United Nations Human Rights Committee that “[...] The Government of the Republic of Korea does not comply with the Covenant and does not fulfill its commitment as a member of the Human Rights Council. [...]” [5]. Freedom House, an independent organization dedicated to worldwide freedom and democracy, pointed out that South Korea blocks political and social content and that critical Internet bloggers and users get arrested. As a result, Freedom House marked South Korea’s press freedom only as “Partly-free” [65]. Finally, Amnesty International called on South Korea’s authorities, among other claims, to select South Korea’s National Human Rights Commission independently and transparently [69].

In the following two sections we will describe the Internet censorship (Section 3.2.1) and communications surveillance (Section 3.2.2) issues in South Korea.

### 3.2.1 Internet Censorship

In South Korea a wide range of information, from election-related discourses to discussions about North Korea, is censored, completely blocked or deleted [112]. Besides being the first Asian country to introduce a data privacy law, it was supposedly also the first country worldwide that introduced an Internet-specific censorship law (“Telecommunications Business Act”) [66]. What is

also unique compared to other democratic countries is that South Korea has a central censorship body called the “Korea Communications Standards Commission” (KCSC) which was created in 2008 by the new conservative government. Its nine members, who are appointed by the cabinet, can regulate content and ethical standards of broadcasting and telecommunication activities without judicial prior review. The commission does not directly carry out censorship, but its recommendations are almost never rejected. Internet service providers face large fines if they do not comply, and message board operators can be jailed. In 2011, the Electronic Frontier Foundation (EFF) criticized the KCSC for recommending censorship of an Internet activist’s blog [47].

Internet censorship in South Korea is increasing each year according to the Korea Internet Transparency Report<sup>2</sup> published by the Korea Internet Transparency Reporting Team and Open Net Korea. The report says that from 2011 to 2014 the number of Internet content takedowns increased from 53,485 to 132,884 [122]. For instance, a number of Twitter accounts that criticized the government were blocked or deleted [60, 106]. Other “offline” content such as artistic expressions that satirizes political elites is also often banned [104].

### 3.2.2 Communications Surveillance

Internet mass surveillance between 2011 and 2014 is on the rise according to the Korea Internet Transparency Report. The report compiles data from the South Korean government as well as from private Internet corporations on government requests to private user information [122]. The report categorizes four major surveillance requests or techniques that are possible in South Korea:

***Communication-Restricting Measures:*** These eavesdropping requests can be performed by government agencies in order to access the content of communications. These actions require court permission.

***Communication Confirmation Data:*** This refers to the acquisition of communication metadata from telecommunication companies. The requests include information such as who is talking to whom, from where and for how long. In recent events, metadata requests occurred in the form of so-called “cell tower dumps” in order to obtain mobile phone metadata from citizens participating in protests. The request for communication confirmation data requires a court permission.

---

<sup>2</sup>[www.transparency.kr](http://www.transparency.kr)

**Communications Data:** Communications Data refers to the acquisition of a subscriber identifying information. Subscriber identifying information such as name, identification number, address, date of subscription and unsubscription, telephone number and other data can be requested *without* a court permission [15]. Most of the subscriber information is requested from telecommunication service providers – KT, SK Telecom, and LG Uplus – and not from Internet companies such as Internet portals [122, 83].

**Search and Seizure Warrants:** This is the most powerful technique in which investigatory agencies are able to request all communication data – message content, metadata, and subscriber information – in accordance with the Criminal Procedure Act. The number of requests for communications is not disclosed by the government for public review. However, the two major South Korean Internet corporations, Kakao and Naver, publish numbers on received and compiled search and seizure warrants in their own transparency reports.

The findings of the transparency report suggest that Internet surveillance is gaining popularity among government authorities (see Table 3.1). One major finding of the report is that Internet communication wiretapping is increasing (taking both datasets from government authorities as well as from Kakao [34] and Naver [35] into account). Interceptions on KakaoTalk, which holds more than 90 percent of instant messaging services in South Korea, has increased from the first half of 2012 to the first half of 2014.

	2011		2012		2013		2014	
	Docs	Accounts	Docs	Accounts	Docs	Accounts	Docs	Accounts
All comm.	707	7167	447	6087	592	6032	570	5846
All Internet	446	1815	265	1654	401	1887	372	1748
Naver	n/a	n/a	30	79	72	195	56	193
Daum <sup>3</sup>	n/a	n/a	53	324	68	272	47	237
Kakao	n/a	n/a	41	47	81	89	78	117

Table 3.1: Status of interceptions between 2011 and 2014. Table adapted from [122].

Another important finding of the report is that the number of requests for communication metadata is slowly decreasing whereas the number of requests for subscriber information is on the rise. According to [83], South Korea’s three main Internet service providers (ISPs) have been handing over

---

<sup>3</sup>Daum was a Korean Internet company that was merged with Kakao in 2015.

subscriber information to more than six million phone numbers in the first half of 2014 alone. Taking different Internet companies including the ISPs into account, nearly 13 million accounts have been seized in 2014 in a country with just 50 million citizens. Most worryingly, data published by Kakao and Naver reveals that the number of search and seizure warrants have increased three times after the start of the new Park administration in 2013, i.e., from the first half of 2012 to the first half of 2014 [34].

The results of the transparency report are limited and incomplete for several reasons. One reason is that the Ministry of Science, ICT and Future Planning (MSIP) only discloses the total numbers of data requests to telecommunication and Internet companies and that those numbers are not accurate enough to provide a detailed evaluation. Another reason is that the statistics on search and seizure warrants entirely rely on the private Internet service providers' transparency reports because the ministry does not disclose those numbers. Finally, only two major Internet companies – Kakao and Naver – are publishing numbers of government requests to user data. Other telecommunications and Internet service providers do not release any statistics on government requests to the public directly, but through the MSIP only.

In the next section we analyse several surveillance scandals in South Korea from 2014 to 2016 and discuss how they impacted Kakao's privacy policy and terms of service.

### **3.2.3 A Timeline of Surveillance Scandals in South Korea and Their Implications on KakaoTalk**

In this section we provide a timeline of surveillance scandals between 2014 and 2016 and analyse the different implications and changes on KakaoTalk's privacy policy and terms of service. We also show to what extent Kakao cooperated with government authorities to comply with user data and wiretapping requests. The goal of this section is to help users understand why Kakao has been moving towards certain directions and how the policies around it have changed over time.

#### **3.2.3.1 2014**

2014 was the year in which Kakao's privacy policies positively changed the most due to a series of political and legal events in South Korea. An important user privacy improvement occurred in August 2014, when Kakao and other Internet businesses were legally required to stop storing users' Resident

Registrations Numbers (RRNs<sup>4</sup>) and to destroy those already collected [79]. However, mobile service providers are still allowed to require users' RRNs to verify their identities for service registration.

In October 2014, Kakao went through its major privacy policy changes. All started in April 2014 with a disaster, when a ferry carrying mostly school students capsized and more than 300 passenger and crew members died<sup>5</sup>. The official handling and response to the ferry disaster provoked widespread criticism towards the government of president Park Geun-Hye and led to major demonstrations throughout the country. Worryingly, the criticism prompted a series of legal and political consequences.

On September 16, president Park Geun-Hye told in a cabinet meeting that “profanity towards the president had gone too far”, that “insulting the president is equal to insulting the nation”, and that false rumours “divided [the] society”<sup>6</sup>. As a result, Park Geun-Hye pledged to prosecute people spreading rumours about her or the government on Twitter, social media and instant messenger services such as KakaoTalk.

Shortly after Park's speech, the Public Prosecutor's Office established an anti-defamation special investigation team which monitored and censored inappropriate public and private online content and also prosecuted suspects. The investigation team especially monitored KakaoTalk to seek for violations by reading chat messages in real-time, even though Kakao denied that it provided real-time wiretapping access. Nevertheless, Kakao showed collaboration by holding a closed-door meeting with prosecutors to discuss how online rumours and criticism can be stopped [70].

Kakao's cooperation and the active monitoring of KakaoTalk led to unjustified indiscriminate data collection and surveillance of protesters for violations of the Assembly and Demonstration Act [137]. In October 2014, Jin Woo Jung, a vice representative of the Labor Party, announced in a press conference that he was accused by the government “causing public unrest” for organizing a demonstration. Jung said that prosecutors had accessed his private KakaoTalk conversations as well as the personal details of his 3,000 contacts over a period of 40 days. In another case, Hye In Yong, a university student who organized a solidarity march for the victims of the ferry tragedy, was also a subject of KakaoTalk surveillance. The search and seizure warrant against her included personal information, message content (texts, pictures, and videos) and metadata (e.g., KakaoTalk ID, phone number used for au-

---

<sup>4</sup>An RRN is a 13-digit number that is assigned to a South Korean citizen at birth and can be used to uniquely identify a person.

<sup>5</sup>[https://en.wikipedia.org/wiki/Sinking\\_of\\_MV\\_Sewol](https://en.wikipedia.org/wiki/Sinking_of_MV_Sewol)

<sup>6</sup>Full text of the president's speech to the cabinet available in Korean at: <http://www.hani.co.kr/arti/politics/bluehouse/655420.html>.

thentication, IP addresses, and other metadata). Among the metadata was supposedly also her cellular modem’s MAC address that may had been used by authorities for tracking Yong’s location [65, 122]. Similar to Jung’s case, the surveillance of Yong led to unjustified indiscriminate data collection of innocent people because all of her contacts’ conversations and metadata were also seized.

The revelations of the different surveillance cases and Kakao’s participating role in them caused major concerns in the South Korean public. As a reaction, around 400,000 KakaoTalk users and 1.5 million South Koreans signed up for more secure non-South Korean-based chat applications within seven days [75]. In addition, asked by a public survey conducted between October and November 2014, more than seven out of ten South Koreans answered that they were worried about secret government surveillance [64].

In order to regain trust, Kakao’s co-CEO Sirgoo Lee reacted with a press conference saying “We stopped accepting prosecution warrants to monitor our users’ private conversations from 7 October, and hereby announce that we will continue to do so.” [94, 12]. Kakao also said that it had to comply with the law and that it was not able to deny governments’ requests due to search and seizure warrants [119].

As the Korea Internet Transparency Report shows, Lee’s announcement was indeed put into practice: The report indicates that the number of processed requests for message interception, message metadata, subscriber personal information, and search and seizure warrants decreased in the second half of 2014 [122, 34]. Furthermore, the next day after Lee’s press conference, Kakao announced a number of new security and privacy features which we will now describe in the following paragraphs:

***Reduced time of message storage:*** Kakao stores KakaoTalk messages on a server in order to provide asynchronous messaging. As part of Kakao’s commitments, this storage time was reduced from 3-7 to 2-3 days. The company also announced to introduce a new “Privacy Mode” in which all *read* messages are automatically deleted from Kakao’s servers. Thus, Kakao stops storing the chat history when a sender and a recipient are online at the same time [120].

***End-to-end encrypted chat and decline invites:*** KakaoTalk added one-on-one end-to-end encryption support to the Android version on 8 December [31] and to the iOS version on 17 December 2014; a group chat end-to-end functionality was added in March 2015<sup>7</sup>. It also introduced a “Decline

---

<sup>7</sup><https://blog.kakaocorp.com/?p=1410>

Invites” feature which allows users to permanently leave a chatroom.

**Privacy Policy Advisory Committee:** Kakao established a committee<sup>8</sup> that consists of outside security and privacy advisors and other professionals from different fields. For instance, advisors include members of the Korean Internet Security Agency<sup>9</sup> and one of the founders of Open Net Korea<sup>10</sup>. Kakao is also a member of an association that engages with non-industry and non-government organizations on privacy and freedom of expression.

**Terms of service and privacy policy:** Kakao’s policies received good results in Ranking Digital Rights’s Corporate Accountability Index [102]. Most notably, the company provides notifications for any policy changes [27].

**Technical measures:** As part of its efforts, Kakao introduced a number of technical measures to protect user’s personal information<sup>11</sup>. For instance, the company established tighter access control mechanisms for its personal information processing system and introduced intrusion detection systems to monitor its internal networks. Kakao also holds a number of accreditations from government organizations and information security certifications such as the ISO/IEC 27001 certificate<sup>12</sup>.

**Kakao Transparency Report:** In January 2015, Kakao revealed its first transparency report on user information eavesdropping and search warrants [19, 34]. Kakao was the first Asia-based and the first South Korean telecommunication company to release such a report. Regarding KakaoTalk, the company revealed that it received 2,467 metadata requests, 147 warrants for real-time monitoring of user chat, and 4,807 warrants to access past user conversations [71]. In addition to government requests, the report also includes information about private requests to block or restrict content. In this regard, the report contains more data than any other transparency report published by other Internet companies [102].

Kakao states that requests for message content and metadata are only provided if they have legal justification [20]. Also, Kakao restricts the amount

<sup>8</sup><http://privacy.kakaocorp.com/en/protection/commonssion>

<sup>9</sup>[https://en.wikipedia.org/wiki/Korea\\_Internet\\_%26\\_Security\\_Agency](https://en.wikipedia.org/wiki/Korea_Internet_%26_Security_Agency)

<sup>10</sup>[https://en.wikipedia.org/wiki/OpenNet\\_%28organization%29](https://en.wikipedia.org/wiki/OpenNet_%28organization%29)

<sup>11</sup><http://privacy.kakaocorp.com/en/protection/tech> and <http://privacy.kakaocorp.com/en/protection/certification>

<sup>12</sup><http://www.iso.org/iso/home/standards/management-standards/iso27001.htm>

of metadata it provides as defined in the Protection of Communication Secrets Act. As for subscriber identifying information, the company does not accept any warrantless requests since the Seoul High Court ruled in 2012 that Internet companies are liable for providing personal information without carefully reviewing investigatory agencies' warrantless requests. However, subscriber information can be still obtained from government authorities by the use of a search and seizure warrant. Finally, Kakao does not process a request if the request document misses important details or contains any errors (e.g., document has not been officially stamped).

Kakao's transparency report has been criticized for a number of reasons. Firstly, it does not include statistics about which different investigative agencies requested access to user information and therefore it is not clear who filed the most data requests.

Secondly, since the report shows the data requests for Daum and Kakao separately, a KakaoTalk user does not know which statistics to consider because KakaoTalk uses both company's online storage systems. Since the merger of Kakao and Daum the company has stated that "[...] Any user information, including location information, maintained by Kakao Corp. will be securely transferred to Daum Communications Corp. to ensure service quality. [...]" [27]. This is critical because the data request statistics for Daum and Kakao vary significantly. For example, Daum has been seeing substantially more search and seizure warrants in the first half of 2015 than Kakao.

Thirdly, Kakao may not always prevent any unjustified indiscriminate data collection by accepting request documents that also contain account requests for the suspects' chat partners [137].

### 3.2.3.2 2015

In October 2015, KakaoTalk's privacy policy changed radically after Kakao's new CEO Ji Hoon Rim<sup>13</sup> announced to start cooperating with government agencies again [67]. This decision marked a 180-degree turnaround after Kakao's co-CEO Sirgoo Lee had promised one year earlier to not collaborate with government officials (see Section 3.2.3.1). Rim's statement prompted other Internet businesses to move and host their services outside South Korea [74]. Through this "server exile" movement some Internet corporations wanted to avoid the same fate as KakaoTalk in 2014 when it lost a large number of its user base to Telegram due to a surveillance scandal (see Section 3.2.3.1).

---

<sup>13</sup>Ji Hoon Rim was announced in August 2015, see [http://www.kakaocorp.com/en/pr/pressRelease\\_view?page=2&group=1&idx=8318](http://www.kakaocorp.com/en/pr/pressRelease_view?page=2&group=1&idx=8318).

As Kakao's transparency report indicates, Rim's announcement was immediately put into practice. The report states that "[...] Kakao resumed cooperation with requests for communication-restricting measures in October 2015 [...]" which can be verified from the fact that processed interception-requests went up from zero in the first half of 2015 to eight in the later half of 2015. As one can see from the report, Rim's announcement may be also an explanation for two other alarming developments. First, the number of processed search and seizure warrants rose from 1,040 in the first half to 1,261 in the second half of 2015. Second, the amount of copyright infringement requests went up significantly from almost 200 in the first half of 2015 to more than 16,000 requests in the second half of 2015. What is also worth noting is that Rim's radical decision does not quite fit into Kakao's privacy philosophy which claims to "[...] proactively participate in critical discourse on privacy protection" [29].

In November 2015, Kakao's co-CEO Sirgoo Lee resigned from the company after allegations he did not do enough to prevent child pornography being spread over Kakao Group [77]. According to the Youth Protection Act, Kakao must block illegal obscene content, even though the company argued that the law does not provide any clear guidelines on how this should be done. Kakao said that it scans for malicious keywords and URLs and filters them accordingly.

### 3.2.3.3 2016

In March 2016, the government of president Park Geun-hye passed a new anti-terror bill that greatly increased the power and authority of the National Intelligence Service (NIS) [44, 113]. With the new law the NIS is allowed to eavesdrop on terror suspects' communications, access their financial transactions, and to collect their personal information. In the past, the agency had already actively monitored the Internet and telecommunications in South Korea. According to the Korea Internet Transparency Report, 90 percent of all wiretapping and interception requests between 2011 and 2014 were made by NIS. Before president Park's election in 2012, the agency also monitored Internet discussion boards to discredit content of opposition parties. For these reasons, there are legitimate concerns that the spy agency could abuse its newly gained powers for domestic surveillance under the name of counter-terrorism. Worse, there are fears that the NIS could require information technology companies such as Samsung to demand weak or "export-grade" cryptography that would be breakable by the agency [136].

### 3.2.4 Conclusion

South Korea is a heavily connected and monitored country that eavesdrops on Korean citizens' Internet communications on a large scale. Internet surveillance from 2011 to the first half of 2014 was expanding until the practice decreased in late 2014 due to the controversy surrounding the wiretapping scandal of KakaoTalk. However, after a one-year surveillance decline, search and seizure and communication interception requests rose again after Kakao's new CEO said to resume cooperation with government officials.

Many of the spying requests were issued because of violations of the Assembly and Demonstration Act (Sewol ferry protests in 2014) and of the interference of business (railway union strikes in 2013 [122]). This is a serious problem because all of these violations relate to freedom of expression restrictions. The Internet censorship and surveillance mechanisms do not only have a "chilling effect" [84] on the free flow of information, but also make Internet business growth difficult (see Sections 3.2.3.1 and 3.2.3.2). Worse, surveillance is often conducted not only on the target, but also on numerous innocent people who participate in conversations with the suspect [122]. This means that simply being in the same KakaoTalk chat room, is enough to disclose innocent users' private chats to the investigatory authorities. And even though officials are required by law to notify potential suspects that they are being monitored, many targets do not receive such a notice and thus are not aware that private communications are being acquired [122].

From the year of 2014, Kakao improved the protection of user's private information substantially. These enhancements probably did not result from users' protests, but rather from the fact that Kakao lost a large number of its KakaoTalk user base to its competitor Telegram within a week. The introduction of end-to-end encrypted one-on-one and group chat was Kakao's major commitment to user security and privacy, and according to Ranking Digital Rights "Kakao's [...] disclosures related to freedom of expression and privacy are significantly stronger than any other non-Western company [...]" [102]. However, Kakao's commitment towards user privacy is under recent internal and external pressure. On the one hand, there is the new CEO Ji Hoon Rim who made clear that he does not care about privacy as much as his predecessors by allowing message-interception requests from October 2015. On the other hand, there is increasing legal pressure such as the new anti-terror law that may demand Kakao to ease its surveillance restrictions.

In the next section we will analyse and structure Kakao's main public security and privacy statements.

### 3.3 Analysis of Kakao’s Public Security and Privacy Statements

Kakao makes a number of public statements about the security and privacy of KakaoTalk. In the following sections we provide a summary of the most important claims and categorize them similar to Ranking Digital Rights’s “Corporate Accountability Index” [102]. We divided the statements into the categories *Security Promises* (Section 3.3.1) and *Privacy Promises* (Section 3.3.2). Each category is then split into several subcategories.

#### 3.3.1 Security Promises

Kakao makes numerous security statements which we divided into the security categories “Encryption” (Section 3.3.1.1) and “Data and Identity Protection” (Section 3.3.1.2). We structured each category in the following way:

**Description:** A short description of the security category.

**Statements:** A list of the most important statements.

**Resulting research questions:** A list of resulting research questions that are interesting for the technical analysis of KakaoTalk.

##### 3.3.1.1 Encryption

**Description:** Does Kakao deploy encryption and security mechanisms for KakaoTalk? Does KakaoTalk provide any information about its “Secure Chat” feature?

**Statements:**

Guaranteed End-to-end Encryption:

“[...] Secret Chat ensures heightened confidentiality of user conversations by providing end-to-end encryption, where the decryption key for chat messages is stored in the user’s device making the messages only readable by the users involved in the conversation. Since the decryption key is stored only in the device, other parties cannot access conversations through any outside point – even through servers. [...]” [31]

“[...] Kakao’s server is unable to decrypt the encryption [of the ‘Secret Chat’ chatroom]” [32]

“[...] It also explained this new mode will fundamentally make it impossible for the authorities to inspect conversations from the server, with or without a warrant. [...]” [117]

#### Trusted Contacts:

“[...] If the public key image is identical to your friend’s, that means your chat is secured. [...]” [21]

#### Miscellaneous:

“[...] Safe transfer of information is ensured [...]” [32]

“Lee said the company had “top security technology” to prevent leaks and only stored messages for a short time” [137]

### **Resulting Research Questions:**

- Despite Kakao’s claims, would the company be able to decrypt end-to-end encrypted messages?
- Does KakaoTalk use encryption for all communication channels?

#### **3.3.1.2 Data and Identity Protection**

**Description:** Does the company protect user’s personal information and any other critical data?

#### **Statements:**

##### Data protection:

“[...] We at Kakao will never neglect our obligation to protect our user’s personal information, which is protected by law, as well as our user’s privacy-related information from third parties. [...]” [29]

“Kakao makes efforts to protect users’ personal information from being leaked with technology and policy measures” [30]

“[...] Here at Kakao, we make it our utmost priority to keep user’s personal information safe and secure [...] We undertake all necessary measures to prevent unauthorized access to the personal information, [...]” [118]

#### Implications of Impersonation and Identity Theft:

Ban from the network: “[...] If the user attempts to subscribe under a false identity or the identity of another person [...]” [33]

No refunds: “[...] Refund requests or request for the paying individual’s personal information, in response to identity theft or payment fraud experienced by the Subscriber, may be rejected if circumstances does not fall under those prescribed by law. [...]” [33]

#### **Resulting research questions:**

- Does KakaoTalk protect against user impersonation or identity theft?

### **3.3.2 Privacy Promises**

Compared to other technology companies, Kakao’s commitment to user privacy is quite strong [102]. The company makes a number of privacy statements which we divided into seven different privacy categories. We structured each category in the following way:

**Description:** A short description of the privacy category.

**Statements:** A list of the most important statements.

**Resulting research questions:** A list of resulting research questions that are interesting for the technical analysis of KakaoTalk.

#### **3.3.2.1 Collection of User Information**

**Description:** Does Kakao disclose what user information it collects, how it collects this information, and why?

**Statements:**Data minimization:

“[...] Kakao only collects personal information that is absolutely needed to provide our services. Kakao requests the user for their name, password and contact information (email, telephone number) once a user registers for our service. Some services may verify the identity of the user or collect the user’s delivery, payment, location and device information. Information including IP address, cookie and usage records is collected while the user uses the service [...]” [20]

“[...] Only the most necessary information is collected [...]” [32]

“[...] Limited information such as ID, password and contact information [...]” [28]

Data utilization:

“[...] Your personal information may be used to search, register or notify other Service users who are in your contact list or who may be your KAKAO friends. Furthermore, your personal information may also be used to give any individual notifications, deal with any inquiries or disputes arising in connection with your use of the Services, send content for paid services, or process shipping and payment. In addition, your personal information may be used to perform statistical analysis for new customized services, provide event and advertising information, comply with obligations required under applicable laws and regulations, and prevent any improper use of your personal information that may cause harm to you in violation of law or the Terms of Service. [...]” [24]

“[...] Personal information is used to provide paid services, for marketing/advertising, [...]” [28]

Targeted advertising based on the user’s current location: “[...] an information provision service through which you can [...] be provided with advertisement information using your current location [...]” [25]

Kakao extracts EXIF location data from pictures: “[...] your location data that is recorded with your photos [...]” [25]

User consent:

“[...] Before you get started with an application or program, KAKAO will inform you of the personal information we collect to which you must consent in order to use the Service. [...]” [24]

“Kakao notifies the user of personal information items that are collected, the purpose of use and the information retention period in advance and acquires the user’s approval for the collection and use of such information. (Details are provided in both Daum and Kakao’s Privacy Policies.)” [20]

**Resulting research questions:**

**Data minimization:** Does KakaoTalk collect more information than it claims?

**Data utilization:** Does KakaoTalk utilize user data for any other purposes?

**User consent:** Does KakaoTalk require user consent? Are the policies the same as the ones published on Kakao’s website?

**3.3.2.2 Collection of User Information from Third-Parties**

**Description:** Does KakaoTalk publish clear information about whether it collects user information from third parties?

**Statements:** No statements found.

**Resulting research questions:**

- Does KakaoTalk collect any user information from third-parties (e.g., by using third-party advertising libraries)?

**3.3.2.3 Sharing of User Information**

**Description:** Does Kakao disclose if and how it shares user information with third-parties?

**Statements:**User consent:

“[...] Information is provided [to third-parties] only after acquiring user’s consent. [...]” [28]

“As a rule, a user’s personal information is not provided to a third party without the user’s consent.” [20]

Third-parties:

1. Transcosmos Korea & Metanet MCC & DK Service, DK techin, Danal, NICE Investors Service, LG U+, Podotree, SureM, DreamSecurity, SK-telink, NOSTech, KT, KTH, Korea Smart Card Co. Ltd, NICE Information & Telecommunication Inc
2. Credit card companies: Samsung, Shinhan, KB, Hana, BC, Citi, Lotte, Hyundai, NHnonghyup
3. Business partners<sup>14</sup>
4. Shipping companies<sup>15</sup> [24]
5. Up to 18,000 “Connected Services”<sup>16</sup>

Miscellaneous:

“[...] KAKAO will never disclose your personal information to a third party unless otherwise required by the relevant laws and regulations or consented by you. [...]” [24]

**Resulting research questions:**

- Does KakaoTalk share more information with third-parties than officially claimed?
- Does KakaoTalk share user information with any other third-party not mentioned in the privacy policy?

<sup>14</sup>[http://www.kakao.com/en/policy\\_agree](http://www.kakao.com/en/policy_agree)

<sup>15</sup>[http://www.kakao.com/delivery\\_partners](http://www.kakao.com/delivery_partners) (in Korean) and <http://makers.kakao.com/deliver> (in Korean).

<sup>16</sup>[http://www.kakao.com/provide\\_layer?page=1794&privacySearch=&lang=en#none](http://www.kakao.com/provide_layer?page=1794&privacySearch=&lang=en#none)

- Tracking: What about third-party cookies that are used by KakaoTalk? For what are they used?
- Which data does KakaoTalk share with other Kakao services? According to [102], Kakao does not clearly disclose whether and how it shares user information between different Kakao services.

### 3.3.2.4 User Control Over Information Collection and Sharing

**Description:** Does KakaoTalk provide users with options to control the company’s collection and sharing of their information?

**Statements:**

“[...] If you post any content on Services, the content may be exposed to Kakao services, and you will be deemed to have granted to Kakao and its partners a worldwide and permanent license to use, store, copy, modify, publicly transmit, display and distribute such content within the required extent. [...]” [25]

**Resulting research questions:**

- What online storage system does KakaoTalk use, Daum’s or Kakao’s? This is critical because the number of data requests for Daum and Kakao vary significantly [34].

### 3.3.2.5 User’s Access to Their Own Information

**Description:** Are KakaoTalk users able to access information about them that the company holds?

**Statements:**

Miscellaneous:

“[...] Sign In > My Information > Personal Information Usage Status [...]” [20]

View the list of linked “Services”:

“[...] go to “Connected Apps” in the KAKAO Account menu in your app [...]” [24]

**Resulting research questions:**

- What kind of data is accessible by a KakaoTalk user?

**3.3.2.6 Retention of User Information**

**Description:** Does KakaoTalk disclose how long it retains user information? Does Kakao allow users to selectively remove information collected about them? What information is retained after a user deletes her account?

**Statements:**Miscellaneous:

Depending on which personal information, Kakao deletes data after 3 or 6 months, or after 1, 3, or 5 years (e.g., records on service visits are deleted after 3 months) [24].

Kakao keeps user location information for at least six months: “[...] Kakao keeps your personal location data and the information confirming the use and provision of the location data for a period of at least 6 months pursuant to the Act on the Protection, Use, Etc. of Location Information. [...]” [25]

“[...] Personal information is destroyed without delay once it is used for its intended purposes. [...]” [28]

“All messages are saved on our servers for 2 3 days to ensure sufficient time for the intended party to receive the message, even if their phone is turned off for a day.” [121]

KakaoTalk’s privacy mode: “KakaoTalk will offer a new Privacy Mode which will allow for off-the-record chatting. Activating Privacy Mode will also delete all read messages from the server. In addition, all read messages will be deleted automatically from the servers and Daum Kakao plans to gradually stop storing the chat history in cases where the sender and the recipient are both online. KakaoTalk plans to add Privacy Mode by the end of 2014.” [120]

**Resulting research questions:**

- What about any data that is left on the SD card after KakaoTalk was removed from an Android device?
- KakaoTalk’s privacy mode: Does Kakao store unread KakaoTalk messages longer than three days?

**3.3.2.7 Spam Protection**

**Description:** Does KakaoTalk protect from spam, harassment or other malicious content?

**Statements:**Miscellaneous:

“[...] The administrator will hide inappropriate posts from other users. [...]” [23]

“[...] The Operation Policies of YellowID<sup>17</sup> and StoryChannel<sup>18</sup> are not stated in this Kakao Operation Policy [...]” [23]

“If Kakao determines that the content you have provided violates relevant laws or Kakao’s policies, Kakao may remove, delete or refuse to post such content [...]” [25]

“We are making a genuine effort, by scanning for keywords, looking for malicious links, and allowing users to report objectionable material, but requiring us to filter out even more obscene material on a private service necessarily implies a degree of censorship that would infringe on the privacy of users.” [77]

“[...] the Korean government enforces Juvenile Protection Policy under the Act on Promotion of Information and Communications Network Utilization and Information Protection [...]” [25]

“[...] We will conduct real-time monitoring if there is a public consensus to put that responsibility on operators [...]” [86]

---

<sup>17</sup><https://yellowid.kakao.com/public/policy> (in Korean)

<sup>18</sup><https://ch.kakao.com/terms/rule> (in Korean)

“The Company may remove or refuse to display content that we believe violates our policies or the law. However, that does not necessarily mean that the Company has an obligation to monitor all the content. If you think somebody is violating your rights, you can let us know through the Customer Service.” [22]

#### Resulting research questions:

- Does KakaoTalk or any other associated third-party monitor users’ communications to enforce its operation policy? If yes, how do they block/filter/delete spam or other malicious content?
- Juvenile protection: How does KakaoTalk enforce Articles 41-43 of the *Promotion of Information and Communications Network Utilization and Information Protection Act*<sup>19</sup>?

### 3.3.3 Summary of Resulting Research Questions

Following we summarize the research questions that arose in the previous sections 3.3.1 and 3.3.2:

#### End-to-end encrypted chat

- Despite Kakao’s claims, would the company be able to decrypt end-to-end encrypted messages?

#### Encryption and data protection

- Does KakaoTalk use encryption for all communication channels?
- Does KakaoTalk protect against user impersonation or identity theft?

#### Collection of user information

**Data minimization:** Does KakaoTalk collect more information than it claims?

**Data utilization:** Does KakaoTalk utilize user data for any other purposes?

**User consent:** Does KakaoTalk require user consent? Are the policies the same as the ones published on Kakao’s website?

---

<sup>19</sup>[http://elaw.klri.re.kr/kor\\_service/lawView.do?hseq=32543&lang=ENG](http://elaw.klri.re.kr/kor_service/lawView.do?hseq=32543&lang=ENG)

**Collection of user information from third-parties**

- Does KakaoTalk collect any user information from third-parties (e.g., by using third-party advertising libraries)?

**Sharing of user information**

- Does KakaoTalk share more information with third-parties than officially claimed?
- Does KakaoTalk share user information with any other third-party not mentioned in the privacy policy?
- Tracking: What about third-party cookies that are used by KakaoTalk? For what are they used?
- Which data does KakaoTalk share with other Kakao services? According to [102], Kakao does not clearly disclose whether and how it shares user information between different Kakao services.

**User control over information collection and sharing**

- What online storage system does KakaoTalk use, Daum's or Kakao's? This is critical because the number of data requests for Daum and Kakao vary significantly [34].

**User's access to their own information**

- What kind of data is accessible by a KakaoTalk user?

**Retention of user information**

- What about any data that is left on the SD card after KakaoTalk was removed from an Android device?
- KakaoTalk's privacy mode: Does Kakao store KakaoTalk messages longer than three days?

**Spam protection**

- Does KakaoTalk or any other associated third-party monitor users' communications to enforce its operation policy? If yes, how do they block/filter/delete spam or other malicious content?
- Juvenile protection: How does KakaoTalk enforce Articles 41-43 of the *Promotion of Information and Communications Network Utilization and Information Protection Act*<sup>20</sup>?

---

<sup>20</sup>[http://elaw.klri.re.kr/kor\\_service/lawView.do?hseq=32543&lang=ENG](http://elaw.klri.re.kr/kor_service/lawView.do?hseq=32543&lang=ENG)

## Chapter 4

# Technical Analysis

In this chapter we provide a detailed technical security assessment of KakaoTalk’s end-to-end encryption protocol and “Secret Chat” feature. We start by explaining our methodologies and tools that we used to examine KakaoTalk (Section 4.1). Section 4.2 outlines KakaoTalk’s messaging system architecture as well the internals of the application’s end-to-end encryption protocol. After describing the system we proceed to analyze KakaoTalk’s attack surface and propose possible system threats (Section 4.3). Section 4.4 provides an overview of vulnerabilities of KakaoTalk’s application and messaging protocol. Finally, we conclude by comparing our technical findings with Kakao’s public marketing claims from Section 3.3.

### 4.1 Methodologies and Tools

In this section we describe the methodologies and tools that we used for our technical analysis of KakaoTalk. It is organized as follows:

First, we outline for *which* technical evidence and system vulnerabilities we were looking for by providing two security assessment checklists (Section 4.1.1).

Second, we explain the automated (Section 4.1.3) and manual (Section 4.1.4) analysis methods that we applied in our technical analysis of KakaoTalk. For the automated and manual analysis methods we used different static and dynamic analysis techniques and tools. Static analysis refers to the process of analyzing KakaoTalk’s source code and any other related files (e.g., manifest and resource files) when the application is not running. By dynamic analysis we mean debugging KakaoTalk at runtime and to also monitor logging events and any network and IPC traffic.

Third, we used reverse engineering methods to learn about KakaoTalk’s

end-to-end encryption protocol. Eventually, we applied a number of digital forensics techniques to search for any information leakage issues.

As of June 2016 the latest version of KakaoTalk is 5.7.0. For most of our analysis work we used KakaoTalk 5.5.5 since we started our project in February 2016. However, we used KakaoTalk 4.7.0 for reverse-engineering the application’s end-to-end encryption protocol since this older version was less obfuscated and therefore faster to analyse than version 5.5.5.

### 4.1.1 Assessment Checklists

In this section we present two different security assessment checklists that we used for the technical analysis of KakaoTalk (see Appendix A). The checklists aim to provide a non-exhaustive overview of possible software security weaknesses in Android applications. The first checklist was compiled from many different sources [17, 131, 93, 14, 82] and lists the most common “low-hanging fruits” or security flaws. The second checklist tries to provide a more specialized set of weaknesses that are common for secure IM applications. This list was mainly authored by Jedidiah Crandall [37] and was additionally influenced by Unger’s evaluation framework [126], and Green’s “Cryptographic Engineering” blog<sup>1</sup>. Due to timing constraints and also because of ethical reasons, we limited the scope of our checklists by intentionally leaving out a number of attack vectors. For instance, we did not search for any server-side weaknesses in KakaoTalk’s online backend system. In the following, we briefly explain the main assessment categories of each checklist:

***Checklist 1: Insufficient transport layer protection.*** The majority of Android applications use the Transport Layer Security (TLS) protocol to encrypt the communication channel. However, this type of encryption may not provide confidentiality if the application’s developer does not know how to implement TLS in a correct way. For instance, Android applications may not validate the TLS endpoint’s certificate correctly.

***Checklist 1: Access control issues and information leakage.*** Many Android applications suffer from different access control flaws that often result in sensitive information disclosures. One example flaw are world readable files inside the application’s private directory. As a result, this may allow other user-installed applications to read from or write to files that are normally protected by the application’s sandbox. Information leakage may also

---

<sup>1</sup><http://blog.cryptographyengineering.com/>

occur due to improperly protected IPC interfaces or system-wide readable log files.

**Checklist 1: Vulnerable cryptographic implementations.** Android application developers may use their own non-standard cryptographic implementations rather than relying on well tested cryptography standards. The possibility of finding a software vulnerability in such “hand-baked” cryptographic implementations is typically high because most programmers struggle to implement cryptography in a correct way. For instance, developers may create vulnerable cryptographic functions by using broken cryptography standards or weak encryption keys.

**Checklist 1: Improper authentication procedures:** Most Android applications use server-side rather than client-side mechanisms for authentication. These online authentication procedures can be used to authenticate the mobile phone’s user and/or the application itself. For instance, the user may use her username and password to log into KakaoTalk’s online account whereas KakaoTalk may use a secret token to authenticate itself against a third-party “cloud” service such as Google Cloud Messaging. There may be a number of weaknesses if the authentication procedures are implemented in a flawed way. For example, server API calls may be unauthorized or the application may use other poor authentication methods such as allowing weak user passwords.

**Checklist 1: Improper data validation.** An Android application typically has three major data entry points: IPC interfaces, network interfaces, and user interfaces. If the application fails to properly sanitize any inputs through these entry points, a number of client-side injection attacks may be possible, for example, cross-side scripting (XSS) or SQL injection attacks.

**Checklist 2: Trust establishment.** Before two KakaoTalk users engage in an end-to-end encrypted conversation, they may authenticate each other to make sure that they are really talking to the intended person. Using this category, we examine the process of how two parties exchange any long-term key material. We also analyse how users can make sure that these keys belong to the correct entities.

**Checklist 2: Conversation security.** The category of conversation security includes the analysis of all cryptographic functions that are needed to

engage in an end-to-end encrypted conversation after trust establishment has been achieved (i.e., message encryption and any short-term key exchanges).

**Checklist 2: Transport privacy.** As described in Section 2.1, we argue that a mobile chat application must support the property of “anonymity” in order to mimic a real-world off-the-record conversation. In this assessment category we therefore analyse KakaoTalk in terms of its metadata protection properties.

### 4.1.2 Experimental Setup

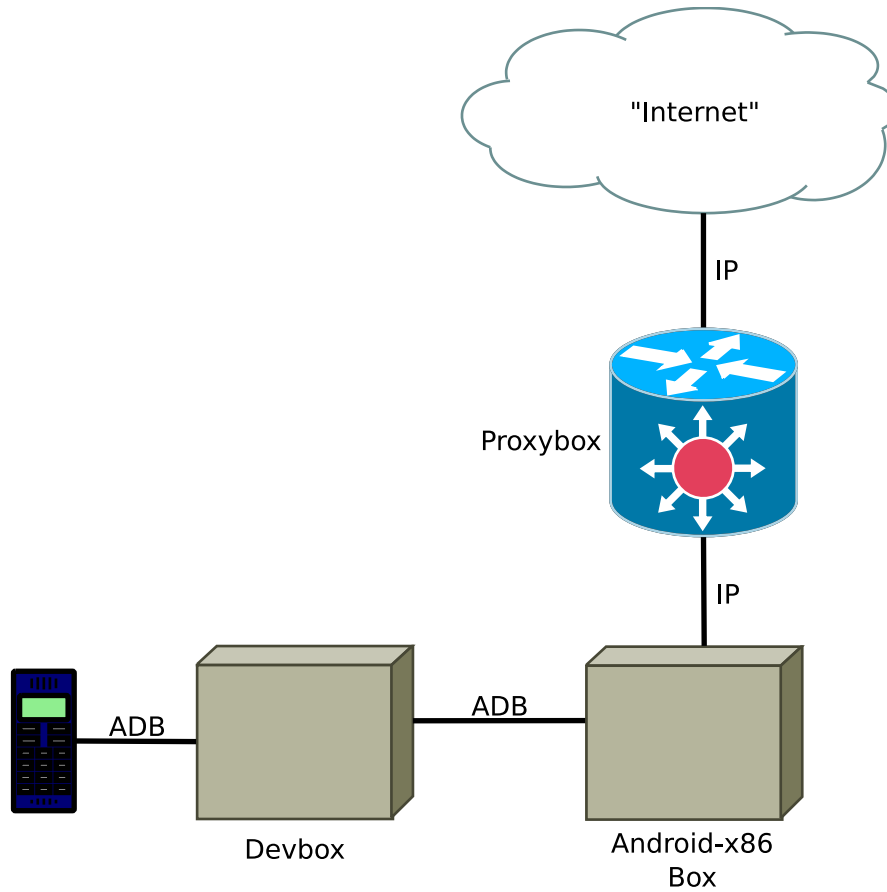


Figure 4.1: Virtual experimental setup based on Vagrant.

As illustrated in Figure 4.1, we used a virtual experimental setup in order to technically analyze the security of KakaoTalk. The main goal of the test

bed was to have a reproducible and portable experimental setup that requires only one physical host. Another goal was also to test and debug KakaoTalk on two different physical mobile devices as well as on an emulator. These goals were accomplished by using *Vagrant*<sup>2</sup> which is a wrapper for virtualization software and which can be used to easily setup a virtual development environment<sup>3</sup>. We configured Vagrant with *Virtualbox*<sup>4</sup> as the virtualization hypervisor and created a virtual network consisting of the following components:

**Devbox:** This *box* has all the tools installed that are required to debug and reverse-engineer an Android application. The virtual machine (VM) is connected to the *Android-x86* box as well as to a physical mobile handset via the Android Debug Bridge (ADB). The software repository of this box includes the following tools: android-sdk<sup>5</sup>, android-ndk<sup>6</sup>, apktool<sup>7</sup>, dex2jar<sup>8</sup>, enjarify<sup>9</sup>, androguard<sup>10</sup>, drozer<sup>11</sup>, pidcat<sup>12</sup>, apkdifff<sup>13</sup>, meld<sup>14</sup>, jadx<sup>15</sup>, jd-gui<sup>16</sup>, simplify<sup>17</sup>, CodeInspect<sup>18</sup>, Bytecode Viewer<sup>19</sup>, frida<sup>20</sup>, Eclipse Memory Analyzer (MAT)<sup>21</sup>, SQLite Database Browser<sup>22</sup>, and 010 Editor<sup>23</sup>.

**Android-x86 box:** The main issue of the standard Android emulator is its slow performance. For this reason, we chose *Android-x86* which is an

---

<sup>2</sup><https://www.vagrantup.com/>

<sup>3</sup>We made our Vagrant setup open-source so that other Android security analysts may benefit from it.

<sup>4</sup><https://www.virtualbox.org/>

<sup>5</sup><https://developer.android.com/studio/index.html>

<sup>6</sup><https://developer.android.com/ndk/index.html>

<sup>7</sup><http://ibotpeaches.github.io/Apktool/>

<sup>8</sup><https://github.com/pxb1988/dex2jar>

<sup>9</sup><https://github.com/google/enjarify>

<sup>10</sup><https://github.com/androguard/androguard>

<sup>11</sup><https://labs.mwrinfosecurity.com/tools/drozer/>

<sup>12</sup><https://github.com/JakeWharton/pidcat>

<sup>13</sup><https://github.com/neosilky/apkdifff>

<sup>14</sup><http://meldmerge.org/>

<sup>15</sup><https://github.com/skylot/jadx>

<sup>16</sup><https://github.com/java-decompiler/jd-gui>

<sup>17</sup><https://github.com/CalebFenton/simplify>

<sup>18</sup><https://codeinspect.sit.fraunhofer.de/>

<sup>19</sup><http://bytecodeviewer.com/>

<sup>20</sup><http://www.frida.re/>

<sup>21</sup><http://www.eclipse.org/mat/>

<sup>22</sup><http://sqlitebrowser.org/>

<sup>23</sup><http://www.sweetscape.com/010editor/>

open-source project that allows to run Android 4.4 on the x86 hardware architecture. Android-x86 has the advantages that it executes much faster than the Android emulator and that it is capable of running inside a virtual machine. We deployed the Android-x86 box with the following tools: Drozer<sup>24</sup>, Xposed framework<sup>25</sup>, Xposed module JustTrustMe<sup>26</sup>, NetGuard<sup>27</sup>, Shark for Root<sup>28</sup>, and frida-server<sup>29</sup>.

**Proxybox:** This box serves as the default gateway for the Android-x86 VM which allowed us to capture and analyse any KakaoTalk network traffic with *Wireshark*. The proxy NAT-traverses all incoming traffic from Android-x86 to an external public network interface. However, HTTP and HTTPS application traffic is forwarded to a software intercepting proxy in order to edit, forward, and drop certain requests and responses made by KakaoTalk. Custom binary network packets such as KakaoTalk's chat messages are forwarded to *Mallory* which is capable of editing and replaying such traffic flows on the fly. The proxy also serves as a virtual WiFi access point in order to sniff KakaoTalk network traffic of the physical mobile testing devices. We installed the following tools on the proxy VM: Wireshark<sup>30</sup>, mallory<sup>31</sup>, Burp Suite<sup>32</sup>, mitmproxy<sup>33</sup>, and create\_ap<sup>34</sup>.

**Android mobile testing phones:** Testing KakaoTalk also on a physical mobile device has several advantages over debugging the application on an Android-x86 emulator only. For example, we can sniff any network traffic that leaves the cellular network interface which is usually not available on an Android emulator. We used a rooted as well as non-rooted mobile phone and deployed both with almost the same toolchain as Android-x86.

### 4.1.3 Automated Analysis

We used a number of automated static and dynamic analysis techniques for the two main reasons: First, we needed to save time and reduce efforts

---

<sup>24</sup><https://labs.mwrinfosecurity.com/tools/drozer/>

<sup>25</sup><http://repo.xposed.info/>

<sup>26</sup><https://github.com/Fuzion24/JustTrustMe>

<sup>27</sup><https://play.google.com/store/apps/details?id=eu.faircode.netguard>

<sup>28</sup><https://play.google.com/store/apps/details?id=lv.n3o.shark>

<sup>29</sup><http://www.frida.re/>

<sup>30</sup><https://www.wireshark.org/>

<sup>31</sup><https://github.com/intrepidusgroup/mallory>

<sup>32</sup><https://portswigger.net/burp/>

<sup>33</sup><https://mitmproxy.org/>

<sup>34</sup>[https://github.com/oblique/create\\_ap](https://github.com/oblique/create_ap)

in the information gathering stage due to timing constraints. By using a manual analysis only, a security analyst may not detect all crucial information and may spend too much time in finding relevant details. Second, we used automated tools that scan for common Android application vulnerabilities in order to establish a baseline.

For the information gathering stage in Section 4.2 we used various on-line tools such as Akana<sup>35</sup>, SandDroid<sup>36</sup>, and Virustotal<sup>37</sup>. In order to find common Android security vulnerabilities we ran static analysis tools such as mallodroid [43], Mobile Security Framework<sup>38</sup>, Quick Android Review Kit<sup>39</sup>, and AndroBugs<sup>40</sup>. Moreover, we used a numerous automated tools for the dynamic analysis of KakaoTalk. For instance, we used *Droidbox*<sup>41</sup> to dynamically trace function calls and *Haystack* [100] to analyze KakaoTalk for any sensitive data leakage during runtime.

#### 4.1.4 Manual Analysis

The main body of this thesis is a manual technical analysis of KakaoTalk. While performing an automated analysis can be beneficial for several reasons, a manual analysis is still crucial in order to achieve reliable results. An automated analysis may result in a number of false positives or false negatives. For instance, Reaves et al. found that the TLS vulnerability scanning tool mallodroid creates a number of false positives [101]. For this reason, we used manual static and dynamic methods to verify any results from the automated analysis.

For our attempts to reverse-engineer KakaoTalk’s end-to-end encryption protocol, we followed all sequences of callback methods<sup>42</sup> a user can possibly take from starting the application until sending an end-to-end encrypted chat message. For this, we started at the `onCreate()` method of KakaoTalk’s launcher activity and went all the way through until KakaoTalk’s *ChatRoomActivity*. This approach allowed us to identify the entry points of the application and ensured that our findings are actual part of live code.

<sup>35</sup><http://akana.mobiseclab.org>

<sup>36</sup><http://sanddroid.xjtu.edu.cn/>

<sup>37</sup><https://www.virustotal.com/>

<sup>38</sup><https://github.com/ajinabraham/Mobile-Security-Framework-MobSF>

<sup>39</sup><https://github.com/linkedin/qark>

<sup>40</sup>[https://github.com/AndroBugs/AndroBugs\\_Framework](https://github.com/AndroBugs/AndroBugs_Framework)

<sup>41</sup><https://github.com/pjlantz/droidbox>

<sup>42</sup>[http://developer.android.com/training/basics/activity-lifecycle/s  
tartting.html](http://developer.android.com/training/basics/activity-lifecycle/startting.html)

#### 4.1.4.1 Static Analysis

In this section we discuss the static analysis of different file types packaged inside KakaoTalk’s Android Application Package (APK<sup>43</sup>). KakaoTalk’s APK includes, among other files, the application’s manifest (`AndroidManifest.xml`), the Dalvik bytecode (`classes.dex`) as well as native libraries (`*.so`). Prior to the analysis, we first needed to unzip and convert these files into human-readable formats. For this, we followed the following steps:

The first step was to use *apktool* to convert the binary XML manifest file into a text-based XML format. We then examined the manifest for KakaoTalk’s usage of permissions and application components. Especially, we searched for any exported application components and dangerous system permissions. These permissions may be an indicator for sensitive information leakage, e.g., by using a permission such as `WRITE_EXTERNAL_STORAGE` an application may store private data on the mobile phone’s SD card.

The second step was to disassemble the Dalvik bytecode into the more human-readable data formats Smali<sup>44</sup> and Java. We first used *Baksmali*<sup>45</sup> to convert the Dalvik bytecode into the Smali assembly language. Then, we used *enjarify* to convert KakaoTalk’s `*.dex` files into Java bytecode (`*.class` files). Lastly, we ran the *Java Decompiler* (JD-Gui) to decompile the Java bytecode into Java code. After the file conversions, we analysed the `*.smali` and `*.java` files for the following most important artifacts:

- Hard-coded secret strings (e.g., static encryption keys, credentials, or API keys)
- Application’s intent behaviour
- API calls (e.g., REST login calls)
- Library usage
  - Networking libraries
  - Cryptography libraries (e.g., java.security or Bouncy Castle)
  - Advertising/Analytics libraries
  - Other third-party libraries
- Imported packages (e.g., UI, HTTPS, and other packages)

<sup>43</sup>[https://en.wikipedia.org/wiki/Android\\_application\\_package](https://en.wikipedia.org/wiki/Android_application_package)

<sup>44</sup>Smali is an assembler for the DEX format and its syntax is loosely based on the Jasmin/dedexer syntax.

<sup>45</sup>Smali/baksmali is an assembler/disassembler for the Dalvik bytecode, see <https://github.com/JesusFreke/smali>.

- Anti-forensics (e.g., Java class loaders that are used to load external code during runtime)
- Embedded native code
- Software development toolkits (SDKs)

The final step was to perform a high-level analysis of any native libraries that are stored in the binary `*.so` file format. Due to timing constraints, we did not disassemble these binaries by using tools such as IDA Professional. However, we used the *strings*<sup>46</sup> tool in order to search for any sensitive hard-coded strings.

#### 4.1.4.2 Dynamic Analysis

By dynamic analysis we mean the analysis of KakaoTalk when the application is running. We split the dynamic analysis into four main parts: Application debugging, digital forensics, network traffic analysis, and inter-process-communication (IPC) analysis. These parts are now explained into more detail:

**Application debugging:** Static code analysis is usually not sufficient to fully understand the behaviour of an (Android) application. Actions such as dynamic loading of external code or calls to native methods can be often spotted during a dynamic code analysis only. We decided to use *CodeInspect* as our main live debugging tool. CodeInspect is a reverse engineering framework for Android applications and comes as an Eclipse-based Integrated Development Environment (IDE). The IDE uses Jimple [129] as an intermediate representation (IR) to convert the Dalvik bytecode into a human-readable format which is easier and faster to analyze. One of Jimple's advantages is that it is a typed language which means that it supports typed variable names. Other representations such as the IR used by smali/baksmali are not typed and are therefore harder to read and to understand. The main reason why we did not consider other debuggers such as IDA PRO or JEB2 was because they only support debugging on the Dalvik bytecode level.

In order to debug an Android application the *debuggable* flag in its `AndroidManifest.xml` must be set to `TRUE`. Even though KakaoTalk had this flag set to `FALSE`, we were able to circumvent this issue by debugging the application on an emulator and on a rooted phone. The Android-x86 emulator has the *ro.debuggable* property set to 1 which means that all applications are

---

<sup>46</sup><http://man7.org/linux/man-pages/man1/strings.1.html>

debuggable regardless of the debuggable flag's value. On a rooted phone the `ro.debuggable` property can be changed from 0 to 1. This way we were able to attach a debugger because KakaoTalk does not perform any root, emulator, or debugger checks. Nevertheless, KakaoTalk implements health checks such as checking the application's signature at runtime. The application throws an error message and refuses to start if it detects that it has been signed with a different developer key. However, this check could be easily bypassed by modifying the signature verification function to always return a boolean value of `TRUE`.

As stated earlier, we used KakaoTalk 4.7.0 to debug and reverse-engineer KakaoTalk's end-to-end encryption protocol. However, this older version is not officially supported and does not connect to KakaoTalk's messaging system. We were able to circumvent this behaviour by updating the version string in the application's `AndroidManifest.xml` to the lowest supported version which is 5.1.0. After these modifications we were able to connect and to chat with other KakaoTalk clients that were running the latest version.

**Digital forensics:** In our dynamic analysis we also examined the physical storage and memory of the mobile device. KakaoTalk stores numerous artifacts on internal (private application file directory) and external (SD card) storage. We especially searched for sensitive information stored inside SQLite databases, SharedPreferences, cookies, system log files and crash reports. In addition, we analyzed the filesystem for world readable/writeable files that may be accessible by other third-party applications. Moreover, we developed custom scripts that dump the system state of the device including system log, open files, network sockets, and running processes. Finally, we obtained a copy of the physical heap memory and analyzed it with Eclipse Memory Analyzer (MAT) using object query language (OQL) queries such as `select * from java.lang.String where toString().startsWith("https://")`.

**Network traffic analysis:** An important aspect of the dynamic analysis was to examine the security of the communication channel between the KakaoTalk application and the KakaoTalk backend data servers. In the case of KakaoTalk, we especially looked at any HTTPS and LOCO related communications. KakaoTalk uses HTTPS mainly to secure remote REST API calls, while LOCO is the application's custom binary messaging protocol. We used our Vagrant experimental setup from Section 4.1.2 in order to intercept such encrypted data communications.

As described earlier, the setup consists of the following components: An Android device (mobile phone and Android-x86 VM) running KakaoTalk,

an intercepting proxy machine, and a KakaoTalk Internet “cloud”. On the intercepting proxy machine we installed *mitmproxy* and *Burp* for mangling with HTTP/HTTPS application traffic. These tools are capable of performing HTTPS man-in-the-middle attacks on the fly by automatically replacing a domain’s TLS certificate with a custom generated one. Since Burp and mitmproxy are only able to deal with HTTP application traffic, we also ran *Mallory* to tamper with any LOCO protocol messages. Besides actively intercepting network traffic, we also passively sniffed network communications with *Wireshark*.

On the Android device we installed a number of additional applications in order to eavesdrop on the communication channel. The most important ones to mention are *tshark* and *NetGuard*. Tshark, which is a derived version of Wireshark for Android, was used to capture any network communications leaving the cellular network interface. Additionally, NetGuard, which is a network firewall manager for Android, was configured to only permit network access for KakaoTalk. All other system and user-installed applications were blocked to use the network in order to filter out all unrelated network traffic.

In order for our HTTPS man-in-the-middle attack to work we needed to further configure the Android device. KakaoTalk performs numerous server certificate checks when establishing a TLS connection:

- 1 It checks if the server hostname matches the certificate’s common name (CN).
- 2 It verifies if the certificate has been signed by a valid certificate authority (CA).
- 3 It verifies the certificate chain by maintaining a whitelist of public keys that are trusted to sign certificates (“Certificate Pinning”).

For the first check, Burp automatically adjusts the CN of its fake certificate to the server endpoint’s domain name. For the second verification, Burp allows to export a CA certificate that can be imported into the Android device’s trusted CA store. Since Burp’s CA signature is trusted after the import, any certificates signed by this authority will be accepted by KakaoTalk. For the last check, Certificate Pinning can be defeated in several ways. For example, one could exploit vulnerabilities in HTTPS libraries (e.g., CVE-2016-2402<sup>47</sup>) or modify the Dalvik executable code to disable Certificate Pinning. We used the least time consuming approach by installing the

---

<sup>47</sup>[https://bugzilla.redhat.com/show\\_bug.cgi?id=1308851](https://bugzilla.redhat.com/show_bug.cgi?id=1308851)

*Xposed Framework* and its *JustTrustMe* module on the Android device. JustTrustMe bypasses Certificate Pinning checks by hooking all related function calls upon application start.

In the last step, we performed some high-level tests against KakaoTalk’s backend data servers. We did not carry out a detailed analysis because we did not have approved access to Kakao’s live-systems. For instance, we did not run any pervasive network scans in order to not possibly disturb KakaoTalk’s operations. Nevertheless, we used a number of third-party services such as Qualys’s<sup>48</sup> SSL server test to scan for known TLS security weaknesses. In addition, we made use of an open-source LOCO protocol implementation<sup>49</sup> to send a number of requests to KakaoTalk’s messaging backend. Finally, we used different Google dorks such as “kakaotalk site:domaintools.com” to enumerate server domains and IP ranges. We also used Shodan’s<sup>50</sup> and Censys’s<sup>51</sup> search engines to find additional data servers to map out KakaoTalk’s network architecture.

***Inter-process Communication (IPC) analysis:*** An important part of the dynamic analysis was to investigate how KakaoTalk communicates on the mobile platform itself. Many Android application unintentionally share data to the system so that other third-party applications may obtain it. For the IPC analysis we used *Drozer* to map out all the IPC interfaces defined for KakaoTalk and to search for weaknesses. Drozer behaves similar to any other user-installed Android application and does not require root privileges. Therefore, any action that Drozer is able to perform (e.g., starting an exported Activity) can be accomplished by any other application as well.

## 4.2 Information Gathering

The first phase of our security assessment focused on gathering as much information about the KakaoTalk application as possible. Information gathering is one of the most crucial parts in any security assessment as it maps out the application’s attack surface. This section is organized as follows: In Section 4.2.2 we list the mobile data that KakaoTalk is storing on the client as well on the server-side. Section 4.2.3 describes KakaoTalk’s system architecture and messaging protocols.

---

<sup>48</sup><https://www.ssllabs.com/ssltest/>

<sup>49</sup><https://github.com/HallaZzang/pykakao>

<sup>50</sup><https://www.shodan.io/>

<sup>51</sup><https://www.censys.io/>

As emphasized previously, most of our work is based on KakaoTalk 5.5.5 while the analysis of the application’s messaging protocol is based on KakaoTalk 4.7.0.

### 4.2.1 Application Details

KakaoTalk 5.5.5 is a hybrid mobile application which displays web content in native Android WebViews. The application downloads HTML5, CSS, JavaScript, image files and other data from online back-end servers and caches them on the mobile phone. As shown in Figure 4.2, KakaoTalk’s main User Interface (UI) `com.kakao.talk/.activity.TaskRootActivity` is divided into four main tabs. We hereby briefly discuss each one of them:

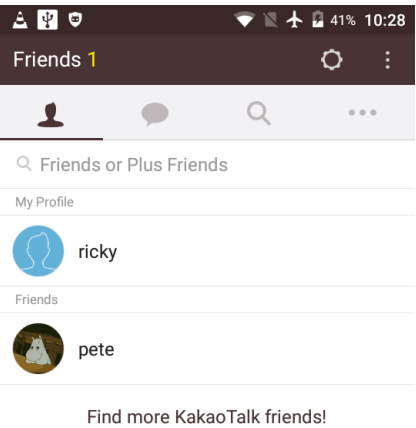
***Friends:*** Under the `FriendSettingsActivity` (Figure 4.2a) the user is able to configure a number of privacy-related settings. For instance, the user can disable the default option “Add Friend Automatically” which adds other KakaoTalk users to the friends list without the user’s consent. Moreover, the user has the chance to disable the “Recommend Friends” option which allows Kakao to constantly search the user’s social graph<sup>52</sup> for possible contacts that the user might know. The user has also the possibility to manage hidden and blocked friends as well as to sync the mobile phone’s address book with KakaoTalk’s servers. Another Activity on the “Friends” tab is the `MiniProfileActivity` on which the user is able to manage her public KakaoTalk profile. The main privacy setting on this Activity is to disable an option that makes users publicly searchable.

***Chats:*** On this tab (Figure 4.2b) the user is able to create open, regular and secret chat rooms and engage in IM chats. The responsible UI for chatting is the `ChatRoomActivity`. Most notably, most of the multimedia messaging features that are available in a regular chat room are not available in a secret chat room.

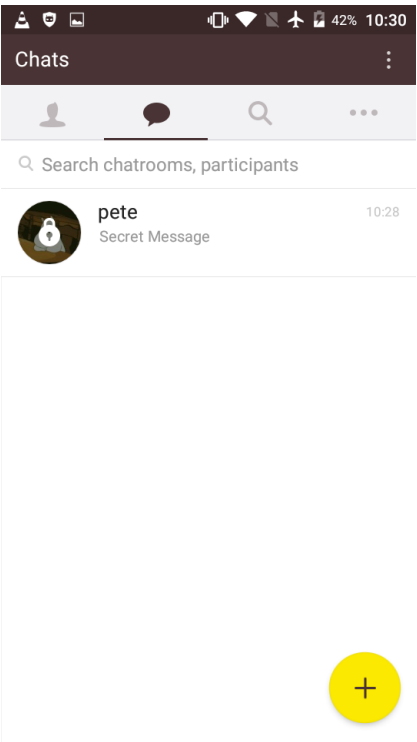
***Find:*** On the `FindFriendsActivity` (Figure 4.2c) the user is able to add new contacts through various channels. Contacts can be added via scanning QR codes, sending invitation SMS or by searching the user’s phone number or KakaoTalk ID (nickname). In addition, the mobile handset’s accelerometer can be used to add a contact who is in close physical proximity. Using this feature, two users need to shake their phones and hold them together closely.

---

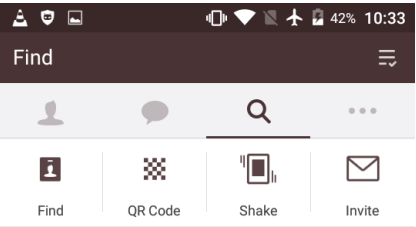
<sup>52</sup>[https://en.wikipedia.org/wiki/Social\\_graph](https://en.wikipedia.org/wiki/Social_graph)



(a) “Friends” screen.

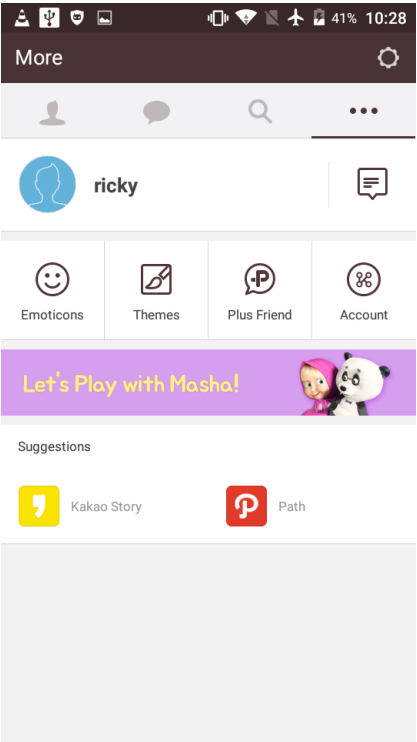


(b) “Chats” screen.



You have no recommended friends.

(c) “Find” screen.



(d) “More” screen.

Figure 4.2: KakaoTalk’s four main Activities.

**More:** The “More” tab (Figure 4.2d) provides the user with a number of links to other services that aim to generate revenue for Kakao. For instance, there are advertisements to emoticons that can be purchased in KakaoTalk’s Item Store or links to KakaoTalk’s “Plus Friend” corporate marketing channel. Under the “Settings” tab the user can perform a number of privacy-related settings. For example, there is a “Do Not Disturb” setting which allows a user to disable chat message notifications. Finally, on the PassLock-Activity the user is further able to set a four-digit pin code which can be used to lock the application in order prevent unauthorized misuse.

After having discussed the main UI elements of KakaoTalk we continue by listing various static information about the application. As most of this data can be found online<sup>53</sup>, we hereby provide only the most important application artifacts (see Appendix B.2):

**Application permissions:** As listed in Appendix B.1, KakaoTalk requires a total number of 36 permissions. 15 of them are dangerous Android system permissions whereas 14 belong to the normal Android system permission group. Interesting dangerous system permissions include `BOOT_COMPLETED` (KakaoTalk may execute code on bootup), `RECEIVE_SMS` (KakaoTalk executes code when a SMS is received) or `RESTART_PACKAGES` (KakaoTalk may close other applications and background processes). In addition, KakaoTalk declares and uses seven custom permissions. The usage of certain permissions indicate that KakaoTalk interacts with various hardware components including Bluetooth, GPS, telephony (phone, SMS), camera, microphone, network interfaces, vibration sensor, fingerprint sensor and the phone’s accelerometer.

**Exported application components:** KakaoTalk exports numerous application components that are not protected by any permissions. This allows other third-party applications on the same device to start a particular component. The application exports 47 Activities, 31 of them do not require any permissions (the launcher activity is included in this number). KakaoTalk also export six Broadcast Receivers (four do not require a permission) and two Services (both require a permission to bind to the particular Service). It is worth noting that KakaoTalk does not implement a Content Provider to share data with other applications.

---

<sup>53</sup><https://virustotal.com/en/file/ca5e56b8ea1e53bad462e2561028685fdc0abc839f678af538a409a047e477ec/analysis/1459472054/> (KakaoTalk version 5.3.0)

**Certificates and public keys:** We found six different files inside KakaoTalk’s APK which contained certificates or public keys. Mostly, we used the “keytool”<sup>54</sup> and “strings” commands to investigate the files. Following, we provide a brief list of the most relevant file properties:

File name	Description
CARoot.der	Root certificate, self-signed, CN=GeoTrust Global CA
c1.der	Class 1 certificate from Symantec <sup>55</sup> , signer of c2.der, certificate issued by Equifax, CN=GeoTrust Global CA
c2.der	Class 2 certificate from Symantec, CN=GeoTrust SSL CA - G3
kakao_c	X.509 certificate, could not be parsed properly
nettest_pubkey.der	Could not be parsed properly
S.bks	Bouncy Castle keystore, no password protection

Table 4.1: KakaoTalk’s certificates and public keys.

**External libraries:** KakaoTalk uses 107 open-source and numerous proprietary software libraries<sup>56</sup>. Most of the libraries are included as standard Java packages which load additional native C/C++ code. We found that KakaoTalk only uses one shared system library which is the Google Maps library. Moreover, we counted a total number of thirteen native libraries that are loaded via the `System.load` or `System.loadLibrary` Java methods. The most interesting libraries are used for mobile payment, software security hardening, and performant public-key cryptographic computations. Following, we provide an overview of the most important third-party libraries:

**Mobile payment:** Most of KakaoTalk’s native libraries are used by Kakao Pay which is a mobile payment service with 8.5 million users registered in May 2016<sup>57</sup>. In order to use Kakao Pay, a user needs to register her credit or debit card and set a pin number. Once a user is registered, she only needs to enter the pin in order to pay. Kakao Pay uses

<sup>54</sup>Keytool comes with the Java Development Kit (JDK).

<sup>55</sup><https://www.symantec.com/ja/jp/pki-class-cert/>

<sup>56</sup><https://katal.kakao.com/android/cs/licenses?prev=1>

<sup>57</sup><http://www.edaily.co.kr/news/NewsRead.edy?SCD=JE41&newsid=01289046612654496&DCD=A00504&OutLnkChk=Y> (in Korean)

a payment technology from LG CNS<sup>58</sup> which provides Korean users to perform credit card transactions. The main Java packages include `com.cns.mpay`, `com.lgcns.kmpay` and `com.lgcns.map`. The first two packages import third-party cryptography libraries offered by Dreamsecurity<sup>59</sup>. The third and later library comprises LG CNS’s MPay module which serves as the back-end payment gateway to Kakao Pay. Finally, Kakao Pay makes use of `com.ahnlab.v3mobileplus`<sup>60</sup> which provides an identity authentication service for financial transactions.

**Software security hardening:** KakaoTalk includes three Java packages by NSHC<sup>61</sup> to improve software security. The library “nSafer” provides an additional cryptographic module whereas “Antivirus” is used as an anti-virus software for Android. The package “nFilter” provides a secure Android keypad which possibly prevents tapjacking attacks.

**Performant public-key cryptographic computations:** KakaoTalk is making use of an open-source<sup>62</sup> Ed25519 library for creating and verifying public-key signatures. The Ed25519 public-key signature system is used during end-to-end encrypted instant messaging.

**Security measures:** As already indicated by KakaoTalk’s usage of certain libraries, one can see that KakaoTalk is taking several security measures. For instance, NSHC’s Antivirus library provides a root checking functionality which, however, was disabled. In addition, KakaoTalk uses Proguard to obfuscate source code by, for example, renaming file, class, and variable names. Finally, KakaoTalk performs health checks such as checking the application’s code signature at runtime.

## 4.2.2 Mobile Data

The KakaoTalk application generates and stores mobile data on the server as well on the client side. We recognized that KakaoTalk stores the following artifacts on the server-side:

---

<sup>58</sup>[http://www.kakaocorp.com/en/pr/pressRelease\\_view?page=1&group=1&idx=7996](http://www.kakaocorp.com/en/pr/pressRelease_view?page=1&group=1&idx=7996)

<sup>59</sup><http://www.dreamsecurity.com/>

<sup>60</sup><https://play.google.com/store/apps/details?id=com.ahnlab.v3mobileplus>

<sup>61</sup><http://www.nshc.net/wp/en/>

<sup>62</sup><https://github.com/dazoe/Android.Ed25519>

**Identifiers:** KakaoTalk nickname, KakaoTalk user ID, Android device UUID, Google GCM ID which is linked to the KakaoTalk username and used for push messaging, phone number, user's email address, and other identifiers.

**Authentication tokens and cryptographic keys:** OAuth refresh and access tokens, LOCO authentication session key, signing public keys and versions, encryption public keys and versions, provider-shared symmetric AES key, HTTPS header authorization token (concatenation of access token and device UUID), user's password used for Kakao account, session cookies, and other tokens.

**Message content:** Unread non-end-to-end encrypted messages (readable by Kakao), unread end-to-end encrypted messages (not readable by Kakao), encrypted file attachments (images, videos, voice, and other file types; readable by Kakao if these attachments were sent in a regular non-end-to-end encrypted chat room), user's phone address book if the user has agreed to share it with Kakao, and other information.

**Communication metadata** Friend relationships (including blocked, hidden, recommended, and deleted friends list), conversation relationships (e.g., how many friends and who participated in the same chat room), user's actions (e.g., creating a chat room, adding a friend, posting a message, and all other actions), total amount of messages sent per user, total message file attachments sent per user, date of when a particular message was sent, size of the encrypted attachment, sender's and receiver's IP address, sender's and receiver's geolocation, and other metadata.

**Miscellaneous:** User avatars and friend aliases that are assigned to certain KakaoTalk usernames, unread sent and received emoticons and stickers, number of bought emoticons and stickers, profile picture and status message, mobile operating system name and version, Android hardware configuration, mobile phone model, screen resolution, SIM operator, phone type, country code, country ISO, KakaoTalk application settings, linked Kakao applications and services, LOCO protocol version, and other data.

KakaoTalk not only stores mobile data on the server but also generates numerous files on the user's handset. Table 4.2 lists the most important artifacts that we found during our forensic analysis of the mobile phone's file system and physical memory.

Artifact	Private data directory	External data directory	Memory	Encrypted
Session cookies	X	X		
OAuth tokens	X		X	X <sup>63</sup>
LOCO authentication session key			X	
Chat message content	X			X
Chat message metadata	X			
Contact list	X			X
Call details	X			
ED25519 private key	X			
ED25519 public key	X			
RSA private key	X			
RSA public key	X			
Email address	X		X	
Device UUID	X			
Credit card information <sup>64</sup>	X			
User activity tracking data	X			
Browser history	X			
WiFi networks	X			
Hardware preferences	X			
Application preferences	X			
User's phone number	X			

Table 4.2: Mobile data stored by KakaoTalk on the mobile handset.

### 4.2.3 Messaging System Overview

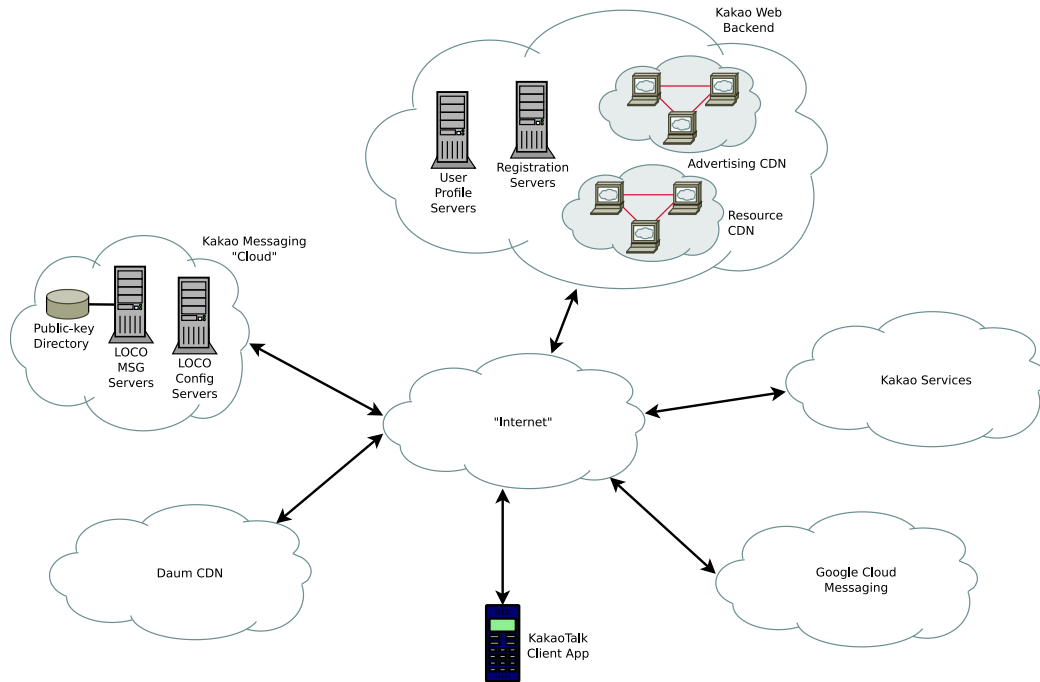


Figure 4.3: KakaoTalk's messaging system.

KakaoTalk's messaging system is based on a classical client-server model. All client communications are relayed through a central messaging server which unpacks, creates, and forwards messaging packets. KakaoTalk uses a custom binary messaging protocol called LOCO which uses the TCP ports 995, 5223, and 5228 (see Section 4.2.3.1 for a detailed analysis of the protocol). All LOCO messaging packets are encrypted with a provider-shared symmetric AES key. The messaging server provides asynchronous messaging by using the store-and-forward concept. In case a recipient is offline, the messaging server stores unread messages and forwards them once the destination is back online. As one can see in Figure 4.3, KakaoTalk's messaging system consists of various distributed components and networks. The most important entities are Kakao's messaging "cloud" and Kakao's Web API backend. As we will discuss in Section 4.2.3.2, Kakao's messaging "cloud" network consists of a LOCO configuration server, a LOCO messaging server and a

<sup>63</sup>Not all of the OAuth tokens were encrypted.

<sup>64</sup>We did not find actual data, but only an empty SQLite table that may store the information.

public-key directory database. The other important entity of the messaging network is Kakao’s Web API backend. The KakaoTalk application primarily operates through Web API calls in order to register and sign-up for a new user account, manage friends lists, adjust settings, or download resources. Most of these calls, which are sent to kakao.com domains, consist of HTTPS POST requests that include JSON objects to transmit data in attribute-value pairs. However, we found that all of these calls also work over HTTP which may be used due to fallback reasons. All remaining components of the messaging system will be now explained briefly:

**Google Cloud Messaging:** Google Cloud Messaging (GCM) is a service provided by Google to enable push notifications. In order to publish a mobile messaging application on the Play Store, Google requires developers to include GCM. KakaoTalk is communicating to GCM via TCP port 5228.

**Daum CDN:** During our analysis we found that KakaoTalk downloads web resources from Daum’s Content Delivery Network (CDN). Notably, most content is downloaded in plaintext over unauthenticated channels.

**Kakao’s messaging “cloud”:** The architecture of Kakao’s Messaging “cloud” will be discussed in Section 4.2.3.2.

**Kakao’s Web API backend:** The most important parts of the Web API backend are an user profile server, a registration server, as well as a resource and advertising CDN. The user profile server handles the KakaoTalk user account, for example, by regularly refreshing the user’s OAuth access token. The server also stores the client application’s configuration settings which are regularly synced and updated on the server-side. In addition, the server stores the user’s friend lists and various account-related settings. The registration server manages the registration process when a new user is signing up with the messaging service. The server validates the user’s phone number, handles the SMS user authentication, and generates session and OAuth tokens (see Section 4.2.3.2). The Web API backend also holds an advertising CDN which is used for user-tracking. Finally, Kakao’s resource CDN serves a variety of emoticons, stickers, and other assets.

**Kakao services:** Kakao services include all services that are linked to KakaoTalk in order to establish Kakao’s mobile platform ecosystem. Ser-

vices such as Plus Friend<sup>65</sup> or Kakao Story<sup>66</sup> can be reached through KakaoTalk and help to monetize the messaging application. We found evidence that KakaoTalk may automatically log into up to fourteen different Kakao services.

#### 4.2.3.1 The LOCO Messaging Protocol

The LOCO messaging protocol builds the foundation of the KakaoTalk messaging system. LOCO is a proprietary request-response BSON<sup>67</sup> protocol based on TCP and mainly operates through the usage of sending custom commands. Each participant in the messaging system can send a LOCO command which is tied to a specific action. For example, the `SWRITE` command writes an end-to-end encrypted message to a chatroom and the `LOGINLIST` command is used to authenticate a KakaoTalk client against a LOCO messaging server. Most of the LOCO protocol reverse engineering work has been already done by Brian Pak in 2012 [97]. During our analysis we found that the LOCO protocol has not changed significantly since then. We detected only a small set of modification, e.g., a slightly different LOCO packet structure. Also, older open-source LOCO implementations such as *pykakao*<sup>68</sup> continue to work as we were able to use *pykakao* to send non-end-to-end encrypted chat messages to another Android KakaoTalk 5.5.5 client. As for completeness, the LOCO messaging protocol encompasses the following three different packets:

ID (4 bytes)	Status (1 byte)	Command (11 bytes)	Type (1 byte)	Body Length (4 bytes)	Body Length (4 bytes)	Payload (N bytes)
--------------	-----------------	--------------------	---------------	-----------------------	-----------------------	-------------------

Figure 4.4: The LOCO “plaintext” packet.

**LOCO “plaintext” packet:** The LOCO “plaintext” packet (Figure 4.4) is a packet that is transmitted unencrypted. However, most LOCO “plaintext” packets are encapsulated into a LOCO “encrypted” packet and are therefore secured against passive network attacks. The most important data field of the packet is the 11 bytes “command” section as well as the “body payload” section. The “body payload” section is encoded in Binary JSON

<sup>65</sup><http://m.kakao.com/plusfriend/en>

<sup>66</sup><http://www.kakao.com/story>

<sup>67</sup><http://bsonspec.org/spec.html>

<sup>68</sup><https://github.com/hallazzang/pykakao>

(BSON) and its length is dependant on the specific command. Note, that the “body length” packet section occurs twice.

**LOCO “encrypted” packet:** The payload of the LOCO “encrypted” packet (Figure 4.5) is an AES encrypted LOCO “plaintext” packet. In case the used block cipher mode requires a padding, the plaintext payload is first PKCS7 encoded and then encrypted with a symmetric key that is shared between the KakaoTalk client and the messaging server.

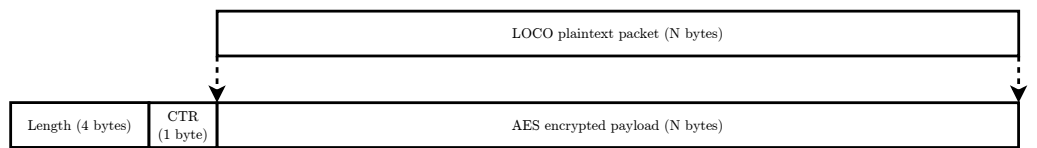


Figure 4.5: The LOCO “encrypted” packet.

**LOCO “handshake” packet:** The “handshake” packet (Figure 4.6) is sent by a KakaoTalk client during the session establishment with a LOCO messaging server. Starting at the fifth byte of the packet, the first four bytes indicate the type of asymmetric encryption used for encrypting the provider-shared symmetric AES 128-bit key (e.g., a type of 0x4 refers to RSA with Optimal Asymmetric Encryption Padding). The next four bytes store the type of block cipher mode that is used for AES encryption. A value of 0x1 refers to Cipher Block Chaining (CBC) with PKCS7 padding, a value of 0x2 to Cipher Feedback (CFB), and a value of 0x3 to Output Feedback (OFB). The next 256 bytes of the packet are used for storing the 128-bit AES key as well as the 128-bit Initialization Vector (IV). Both values are encrypted with the LOCO messaging server’s static 1024-bit public RSA key. The payload of the packet comprises a LOCO “encrypted” packet which in turn contains the LOGINLIST command.

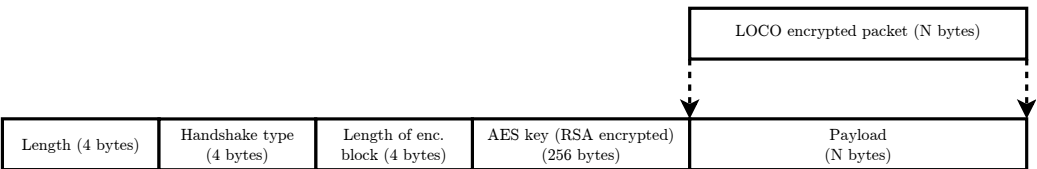


Figure 4.6: The LOCO “handshake” packet.

### 4.2.3.2 Device Registration and Login

All users need to first register and sign-up with the KakaoTalk messaging service before they are allowed to participate in the chat system. One goal of the registration process is to provide a phone-based user authentication model via SMS. Another goal is to compute a number of shared secret tokens which are later used to authenticate the user against the Web API as well as the LOCO messaging back end. As depicted in Figure 4.7, the KakaoTalk application starts the registration process by initiating the following six HTTPS protected Web API calls:

- 1 During the first request the KakaoTalk application posts the user's phone number, device UUID and other data to a server. The server verifies whether the number is reachable and whether the user's mobile handset supports interactive voice response, voice calls, and SMS. In addition, the server maps the user's phone number to her device UUID. If the number is reachable the server responds with the message "We will send you an SMS with a 4-digit verification code to the number above". This message is returned as a JSON object and displayed in a WebView.
- 2 Once the user accepts the dialog, the KakaoTalk application sends a second HTTPS POST request. Similar to the previous HTTPS POST, this request contains the user's phone number, the phone's device UUID, and other data. The server then generates a new four digit PIN code and sends it in a SMS to the user's phone number. If the SMS could be delivered successfully, the server replies with a nonce which is later used by the client in the fifth POST request.
- 3 In the third and fourth requests, the KakaoTalk application issues two GET requests to fetch a HTML login form, a session cookie, a HTML5 cache manifest file, and other data. The HTML5 cache manifest file contains links to client-side JavaScript libraries ("kakao\_accounts\_libs" and "oauth-kakao-accounts-mobile") which the application downloads in later requests. KakaoTalk also downloads JavaScript libraries from third-parties, e.g. from [cdn.ravenjs.com](https://cdn.ravenjs.com)<sup>69</sup>.
- 4 In the meantime, the user received the four digit PIN number that was sent by the server. Once the user submits the PIN number, the KakaoTalk application sends a fifth HTTPS POST request. This request contains the PIN number, the device UUID, and the server nonce that was received in the second response. In case an existing KakaoTalk user

---

<sup>69</sup><https://github.com/getsentry/raven-js>

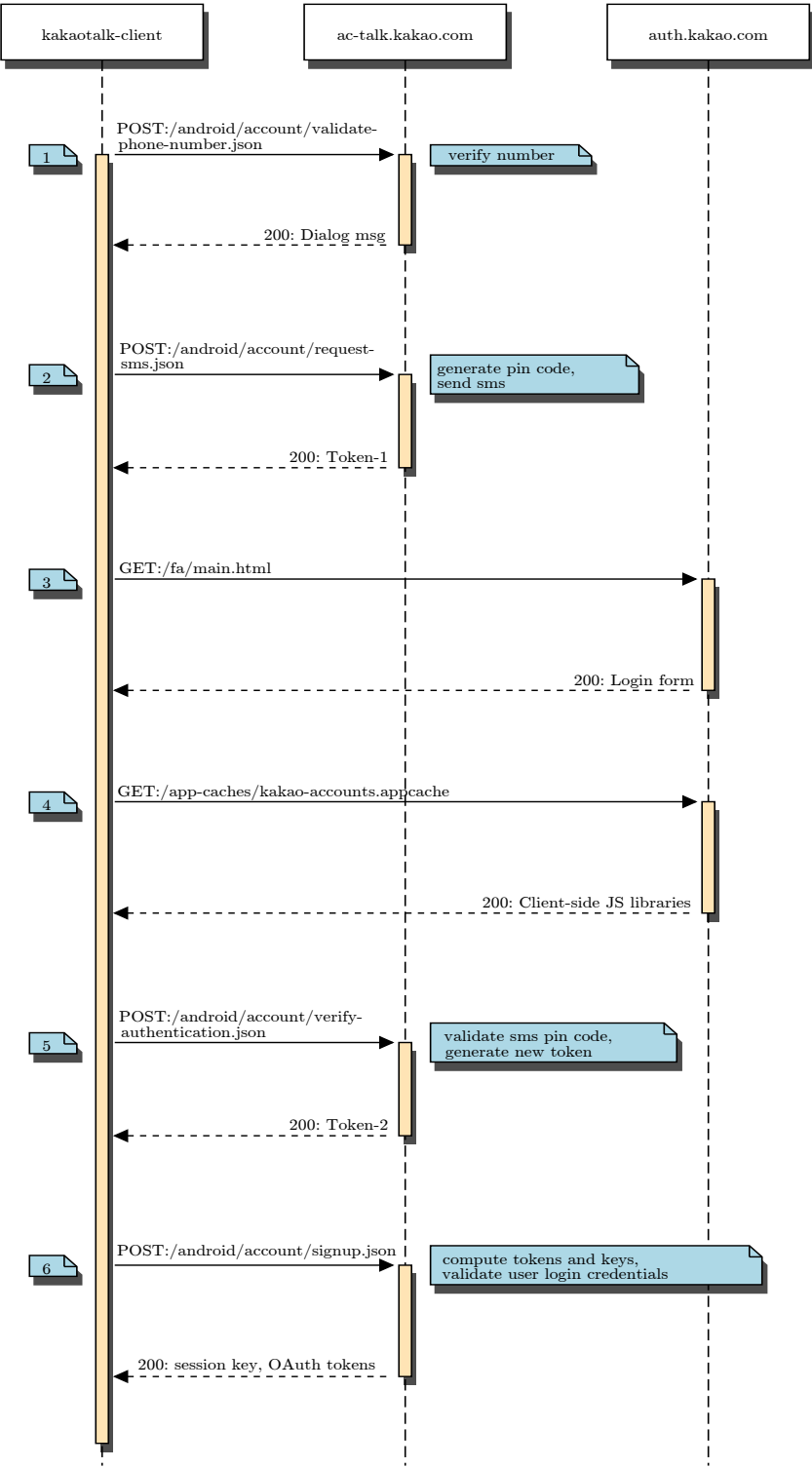


Figure 4.7: The KakaoTalk registration process.

has to re-authenticate, the application also posts the old OAuth access token that has been used by a previous KakaoTalk installation. In this particular case, the KakaoTalk server revokes and deletes the old association of the device UUID and access token. The server also verifies if the phone number is currently active on another mobile phone. In this case, the server responds with a warning message saying that the phone number is currently active on another device and that KakaoTalk only supports one phone number on one device. If the user continues and accepts this warning dialog, the server disconnects the other currently active mobile device from the messaging system. As a result, the system establishes an one-to-one mapping between the user and the mobile device, multi-device support is not supported even though KakaoTalk PC<sup>70</sup> allows to sync messages between the user's PC and mobile device. If the user enters the PIN code incorrectly for too many times, the KakaoTalk application sends a crash report over an unauthenticated channel to <http://group1.magpie.daum.net/magpie/put/>. If the PIN code was entered correctly the server responds with a new nonce which is used in sixth and last HTTPS POST request.

- 5 Now the login form, which was downloaded in the third request, is presented to the user. The user can choose to log in with her Kakao account, which consists of an e-mail address and password, or skip the login by creating a new chat account. Also, the user has the chance to allow or deny KakaoTalk to upload the user's mobile phone's address book. In case the user grants permission, the server would scan the address book and automatically add friends that own a KakaoTalk account. Once the user proceeds, the application sends another HTTPS POST message containing the previously received server nonce, the KakaoTalk nickname, and the old OAuth access token (in case the user has to re-authenticate). Among other data, the server further replies with a KakaoTalk user ID, a bearer OAuth access and refresh token, an OAuth token expiration date, as well as a "LOCO session key". A String concatenation of the OAuth access token and the device UUID is from now on used in the HTTPS Authentication header to authenticate all further Web API calls. The KakaoTalk application uses the "LOCO session key" to authenticate the chat client against the LOCO messaging backend.

In summary, the process of how a KakaoTalk client obtains an OAuth access token works as follows:

---

<sup>70</sup><http://www.kakao.com/services/8/pc>

- 1 The client sends a HTTPS POST request to `https://ac-talk.kakao.com/android/account/request_sms.json` and receives a nonce.
- 2 The client sends the server nonce to `https://ac-talk.kakao.com/android/account/verify_authentication.json` in order to obtain a new nonce.
- 3 The client sends the server nonce to `https://ac-talk.kakao.com/android/account/signup.json` to obtain the OAuth key material.

Since the key generation algorithm is implemented on the server-side we do not know how the access tokens are generated. We did not try to feed the key generator with different input (e.g., a different device UUID or phone number) or otherwise investigate any further. However, by performing a simple String comparison of different access tokens we found potential evidence that the access tokens may lack randomness. From our basic analysis it seems that an access token may be a concatenation of random and static Strings. As illustrated in Figure 4.8, we assume that an access token may be a concatenation of a random 32 character String (red color), a static six digit sequence of zeros (blue color), a thirteen digit timestamp in milliseconds (olive color), a static four digit sequence of zeros (black color), and a random sequence of ten Base64 encoded characters (orange color).

```
b6a6a5efa91d45b8bb91df86673da24c00000014589403796850000Ab6rEoqxUn
dde7c13b3fd41bea37bd1ef1856452400000014580886920360000kz0h09Jmw1
c4461ab6c06a4bcaa3dd2455c8942c1d0000001459286033727000064Pt63C9BJ
b2f0742ca6e2489182e74a73bd302a7f00000014617841824550000D4tubTHa1k
```

Figure 4.8: Similar patterns in KakaoTalk’s OAuth access tokens.

**AES key agreement during LOCO handshake** After the registration process with the Web API backend has finished, the KakaoTalk application uses the device UUID as well as the freshly obtained “LOCO session key” for authenticating itself against a LOCO messaging server. During the session establishment the application exchanges a locally computed 128-bit AES key and Initialization Vector (IV) with the server. As shown in Figure 4.9, the KakaoTalk application first establishes a HTTPS connection with a configuration server. For this, the application sends a LOCO “encrypted” packet containing the **CHECKIN** command. The configuration server replies with a number of LOCO messaging server IPv4 and IPv6 addresses as well as port numbers. After receiving the response, the KakaoTalk application establishes a connection with a server IP and port number provided by the configuration server. For this step, the application sends a LOCO “handshake” packet

containing the AES encrypted payload as well as the AES key and the Initialization Vector (IV) which are both encrypted with the server's public RSA key. The encrypted payload contains the LOGINLIST command, the "LOCO session key", the device UUID, the KakaoTalk user ID as well as other data. If the login was successful, the LOCO messaging server replies with a LOCO "encrypted" packet which contains a status code of 0x0, otherwise the server closes the TCP connection. After that, the server sends a BLSYNC packet which the KakaoTalk application replies with a SET\_PK LOCO packet. Before sending the SET\_PK packet, the KakaoTalk application computes an Ed25519 [9] signature key pair as well as a RSA key pair that is used for signing and encrypting. The private as well as the public key values of both key pairs are then stored in the application's shared preference file `TalkKeyStore.preferences.xml`. After that, the SET\_PK packet is sent. It contains the sender's public Ed25519 and public RSA key as well as the RSA (SHA256) and Ed25519 signature of the phone's device UUID. If the signature verification was successful the server acknowledges the packet with a status code of 0x0. After the handshake procedure the KakaoTalk client is allowed to fully participate in the messaging system (i.e., creating chatrooms, writing messages, and other actions). The KakaoTalk client and messaging server have now exchanged a shared secret that is used for non-end-to-end client-server chat communications. The AES key is newly generated each time when the KakaoTalk application is establishing a connection with a messaging server.

#### 4.2.3.3 The LOCO End-to-end Encryption Messaging Protocol

The LOCO end-to-end (E2E) encryption messaging protocol is based on the LOCO messaging protocol. The E2E "Secret Chat" feature was officially announced on the 8th of October 2014 [120] and was released roughly two month later on the 9th December of 2014<sup>71</sup>. As at the time of writing, only the Android and iOS KakaoTalk version support the "Secret Chat" feature. The E2E protocol introduced a number of new commands for the symmetric key exchange as well as for secure messaging. The most important new LOCO commands are as follows:

**Public key exchange:** The GET\_PK, GET\_LPK, and SET\_PK are getter and setter commands for exchanging public Ed25519 and RSA key material.

**Private key exchange:** The GET\_SK and SET\_SK commands are used to exchange a shared secret that is known by the two KakaoTalk clients only. The messaging server does not have access to this secret.

---

<sup>71</sup><https://blog.kakaocorp.com/?p=943>

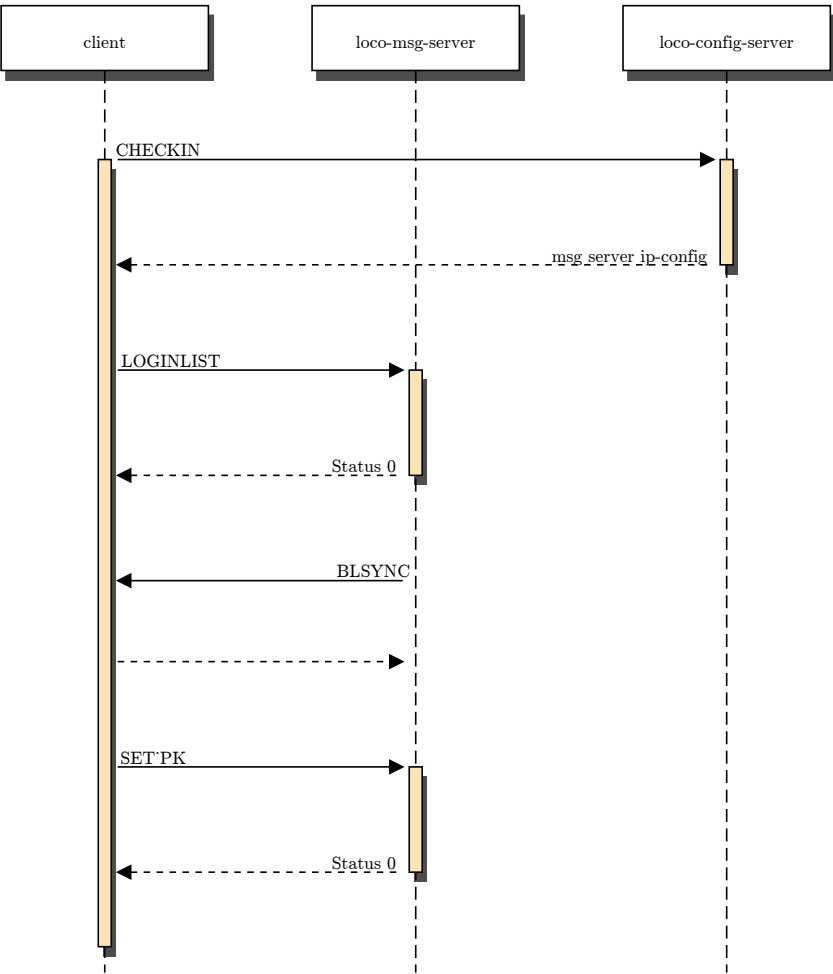


Figure 4.9: The KakaoTalk handshake process.

**E2E messaging:** The **SCREATE** command is sent by the sender when a new secure chat room is created. If the user selects an already created chat room the KakaoTalk application sends a modified version of the **CHATONROOM** command. The **SWRITE** command was introduced to send E2E encrypted messages. A modified version of the **MSG** command is used to receive E2E encrypted messages.

**Protocol Overview** The message communication flow of the LOCO E2E encryption messaging protocol is illustrated in Figure 4.10. For a sender it takes the following three requests from creating a new secret chat room until sending an E2E encrypted message:

- 1 The sender sends a **SCREATE** LOCO command to the messaging server in order to create the secret chat room on the server-side as well as to retrieve the recipient's public key.
- 2 In the second request, the sender sends a **SET\_SK** LOCO message to the server. The packet includes the (RSA encrypted) shared secret value and the Ed25519 signature of the plaintext value's MAC. The server further accepts the message with a status code of 0x0.
- 3 During the third request, the sender sends a **SWRITE** LOCO message to the server. This message includes the E2E encrypted chat message, a Ed25519 signature of the plaintext message's MAC, and a nonce for computing the symmetric MAC key. The server accepts the **SWRITE** message with a status code of 0x0, transforms the **SWRITE** packet into a **MSG** packet and relays it to the recipient.

If the recipient does not obtain the shared secret in order to decrypt the message, she sends a **GET\_SK** request to the server. In addition, the recipient sends a **GET\_PK** request in case she does not possess the sender's public keys. After having obtained the key material, the recipient can decrypt and verify the authenticity of the message.

**Trust Establishment and Key Exchange** KakaoTalk uses an authority-based trust model in combination with optional manual key-fingerprint verification. By an authority-based trust model we mean that KakaoTalk holds a key-directory server that maps the user's mobile phone's device UUID to their Ed25519 and RSA and public keys. The KakaoTalk public-key directory service verifies the ownership of public keys through SMS verification which occurs during the registration process (see Section 4.2.3.2).

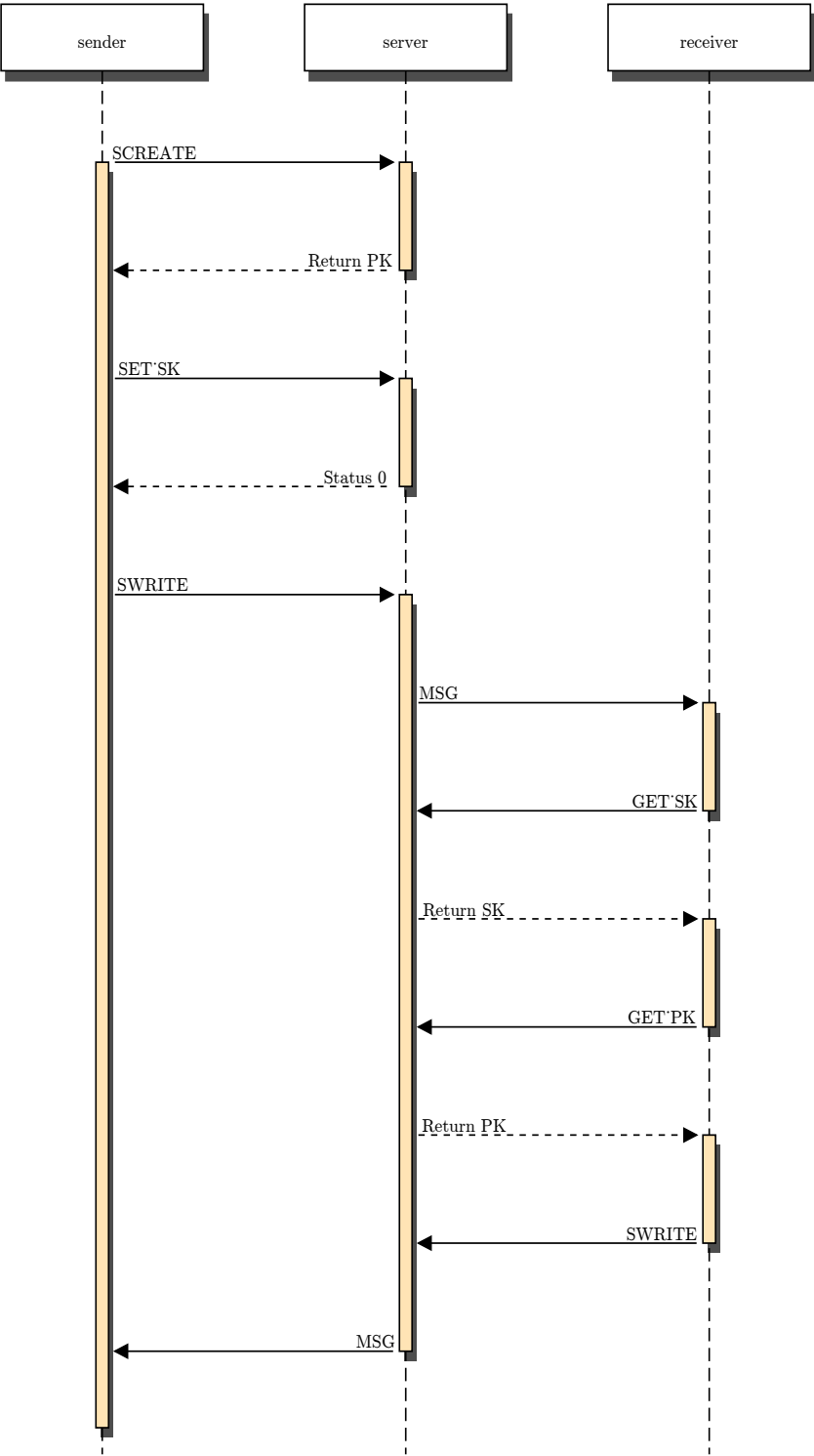


Figure 4.10: Sending and receiving an end-to-end encrypted chat message.

As already discussed in the previous section, the KakaoTalk application uploads its public Ed25519 and RSA key by using the `SET_PK` command. The `SET_PK` LOCO packet is typically sent after the user has established a session with a LOCO messaging server. The packet also contains a Ed25519 as well as a RSA signature of the mobile handset's device UUID.

There are multiple LOCO commands that allow the client to retrieve the recipient's public keys. For instance, the KakaoTalk client can use the `GET_PK`, `GET_LPK`, `SCREATE`, and `CHATONROOM` commands to receive the other party's RSA and Ed25519 public keys.

The `GET_SK` and `SET_SK` LOCO commands are used to exchange a random 256-bit shared secret value that is used for creating a symmetric AES and MAC key. After generating the shared secret the sender encrypts it with the recipient's public RSA key. The sender also creates a HMAC (SHA256) of the plaintext shared secret value. This HMAC is further signed with the sender's private Ed25519 signing key. The `SET_SK` LOCO packet finally comprises the RSA encrypted shared secret, the Ed25519 signature of the plaintext shared secret's HMAC, and a nonce. The receiver obtains the shared secret by sending a `GET_SK` LOCO packet to the messaging server. The recipient can then decrypt the shared secret, verify the Ed25519 signature of the sender, and finally compare the HMAC. In the paragraph after next, we describe how the recipient is able to compute the same MAC and verify the authenticity of the message.

In addition to the authority-based trust model, users can manually verify each other's public key fingerprint. This manual verification process guarantees that a user is talking to the intended person and that the contact is not pretending to be someone else. On the *EncryptionKeysInformationActivity* the user is able to compare an eight times eight grid image of the public key whereas on the *EncryptionKeysInformationDetailActivity* the user has the chance to verify the other party's fingerprint. The fingerprint is composed of the 16 most significant bytes of a SHA1 hash of the user's public key. If a participant's key-material has changed, the other party receives a warning message mentioning that the secret chat room is no longer available. However, we recognized that in some cases one party is still able to send E2E messages while the other party is no longer able to write messages as the message input field disappears from the chatroom.

**Message Authenticity** For every message, the KakaoTalk application creates a new symmetric MAC key and computes a new HMAC of the message with that key. The MAC key generator function is feeded with a random nonce. This nonce is computed by using the shared secret as well as the mes-

sage ID as input parameters. The KakaoTalk application obtains the message ID from the current time in milliseconds. By using another custom function, the application further reduces the number of digits of the message ID from a 13 to a 10 digit number. This 10 digit message ID is then included into the E2E message, so that the receiver is able compute the same nonce and hence the same symmetric MAC key to verify the integrity of the message. We can express the message authentication mechanism more formally:

**Definition 1.** The KakaoTalk message authentication process.

Let  $HMAC_k$  denote a message authentication function with a symmetric key  $k$ . Further, let  $kdf$  denote a key derivation function which takes a password and an optional salt value as inputs to compute a cryptographically strong random key. Then the computation of a HMAC of an E2E encrypted chat message can be described in the following three steps:

1. Computation of random nonce  $n$ :  $n = kdf(shared\_secret, message\_id)$
2. Generation of HMAC key  $k$ :  $k = kdf(n)$
3. Computation of HMAC  $mac$  over message  $m$ :  $mac = HMAC_k(m)$

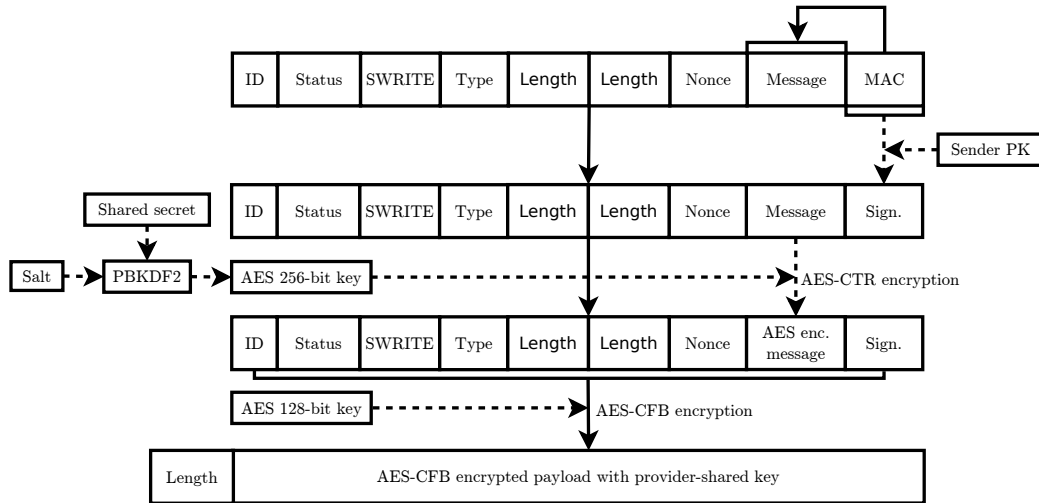


Figure 4.11: The KakaoTalk E2E encryption mechanism.

**Message Encryption** Upon creating a new secret chat room, the KakaoTalk application sends a **SCREATE LOCO** command to the messaging server. The reply of this command contains the recipient's RSA public key. The

chat application continues to create a new random 256-bit shared secret value that is used to compute the symmetric AES and MAC keys. After sending the (RSA encrypted) shared secret via the `SET_SK` command to the server, the KakaoTalk application then computes the AES and MAC keys. The AES encryption key is generated by using a Password-Based Key Derivation Function (PBKDF2) which applies a SHA-1 hash function to the shared secret along with a static 34 character String salt value. PBKDF2 repeats this process 2048 times to produce a 512 bit random value. The first left-most 256 bits of this value result in the final AES symmetric key that is used for E2E message encryption. The recipient is able to compute the same AES key as she possesses the same shared secret as well as the static salt value.

For the random MAC key, the sender first generates a new random nonce which is also being used as the IV during AES encryption. Next, the KakaoTalk application computes a new HMAC (SHA256) of the plaintext message and signs this MAC with the sender's private Ed25519 key. Then it encrypts the chat message using AES in Counter mode (CTR) along with the 256-bit symmetric key as well as the IV. Finally, the application composes the `SWRITE LOCO` packet by including the encrypted message, the Ed25519 signature of the plaintext message's HMAC as well as the nonce. The overall message encryption process is shown in Figure 4.11 and formally summarized in Definition 2.

**Definition 2.** The KakaoTalk E2E message encryption process.

1. Computation of HMAC  $mac$  over message  $m$ :

$$mac = HMAC_k(m)$$

2. Computation of signature  $s$  over input  $mac$ :

$$s = Ed25519Sign_{sender-pk}(mac)$$

3. AES 256-bit key generation of  $k$ :

$$k = PBKDF2(SHA-1, shared\_secret, salt, 2048, 512)$$

4. AES-CTR encryption of message  $m$ :

$$c_{CTR} = e_k(m)$$

5. AES-CFB encryption of LOCO “plaintext” packet with provider-shared key  $k$ :

$$c_{CFB} = e_k(ID || Status || \dots || Nonce || c_{CTR} || s)$$

**Message Decryption** The message decryption process performs the reverse actions of the message encryption process. The E2E encrypted message, the Ed25519 signature of the plaintext message’s HMAC, as well as the nonce for computing the MAC key are received in a **MSG LOCO** packet. If the KakaoTalk application does not possess the shared secret for computing the AES and MAC keys, it sends a **GET\_SK** packet. The KakaoTalk application also sends a **GET\_PK LOCO** packet in case it does not obtain the sender’s public keys. After receiving the key material, it decrypts the shared secret with its private RSA key.

After decryption, the application verifies the integrity of the E2E message by first verifying the Ed25519 signature with the recipient’s public Ed25519 signature key. Further, the application computes a new MAC key using the shared secret and the nonce as inputs. Finally, the chat client computes a HMAC (SHA256) of the plaintext message and verifies whether the locally computed MAC matches the MAC send by the sender.

## 4.3 Threat Analysis

In this section we perform a high-level threat analysis of KakaoTalk’s instant messaging system. We do *not* use a systematic threat model approach such as Microsoft’s STRIDE model [61], but instead follow Tuomas Aura’s lecture material of the CSE-C3400 course at Aalto University<sup>72</sup>. In addition, we exclude topics such as risk assessment, threat prioritization, or countermeasure specification. This section is organized as follows: First, we analyse the attack surface (Section 4.3.1) and highlight the valuable assets that require protection (Section 4.3.2). Section 4.3.3 deals with the analysis of potential threat actors and their particular motivations. Finally, we conclude with a non-exclusive list of possible threats (Section 4.3.4).

### 4.3.1 Attack Surface

After we have described KakaoTalk’s messaging system architecture in Section 4.2 we can proceed to analyze the system’s attack surface. The goal of defining the attack surface is to break the application into a high-level set of distinct system components and to analyze corresponding attack vectors for each of them. As described in the previous section, the main components of KakaoTalk’s messaging system include the *KakaoTalk client application*, the *KakaoTalk messaging and Web API backend*, as well as the *communication*

---

<sup>72</sup>The lecture slides on threat analysis are available at <https://mycourses.aalto.fi/course/view.php?id=4617>.

*channel* which connects the two entities. However, due to ethical reasons and timing constraints we only focused on the KakaoTalk client application and excluded the KakaoTalk messaging and Web API backend from our attack surface analysis. Furthermore, we excluded several other system components and made a number of security assumptions:

First, we excluded the KakaoTalk user who may be a potential target for social-engineering attacks. We also assumed that the user is using KakaoTalk in a good faith and does not seek to cause any harm. Second, we assumed that the mobile platform (hardware and software), on which the KakaoTalk application is running, is *trusted* even though most threat models of secure instant messaging applications treat the mobile platform typically as *untrusted*. Third, we did not analyse the attack surface of KakaoTalk's server backend because we did not have permitted access to Kakao's systems. Fourth, we assumed that the communication link between the KakaoTalk client and the KakaoTalk IT backend is *untrusted*. Finally, we excluded any third-party entities such as Google Cloud Messaging, KakaoTalk services and advertising networks from our attack surface analysis.

As described above, we excluded the user, the mobile phone's hardware platform and the Android operating system and any other third-party software from our attack surface analysis. Instead, we concentrated on the KakaoTalk application which we broke into the following main high-level components:

- *Registration and sign-up*: This component includes the user registration process, user authentication, and sign-up.
- *User authentication after registration*: This component includes all authentication processes after a user has successfully registered with the messaging system.
- *End-to-end encrypted messaging*: This is the major system component that we focused on in this work. End-to-end encrypted messaging includes actions including symmetric key-exchanges and message encryption/decryption processes.
- *In-app purchases*: This component mainly comprises KakaoTalk's Item Store and Kakao Pay which handles credit card transactions for Korean users.

After we broke the KakaoTalk application into several distinct system components we defined the attack surface by analyzing each component's remote and local data entry and exit points:

**IP network sockets:** HTTP/HTTPS Web API calls and LOCO protocol messages are sent and received through IP network sockets either via the WiFi or cellular network interface.

**Telephony:** KakaoTalk receives data through inbound SMS messages and incoming phone calls. Data is leaving the telephony hardware by placing phone calls or sending SMS messages.

**Hardware interfaces:** KakaoTalk interacts with various hardware components including Bluetooth, GPS, camera (e.g., data entry point through scanning QR codes), microphone, vibration sensor, fingerprint sensor and the phone's accelerometer.

**Multimedia file parsers:** Multimedia file parsers process sent and received media files such as videos or images that are transmitted as file attachments. Android's multimedia parser component *Stagefright* was known to be exploitable when processing malformed MP4 files<sup>73</sup>.

**Local file system:** KakaoTalk reads and writes files from and to the application's private and external data directories (e.g., the application caches web content on the phone's SD card).

**User Interface:** Through KakaoTalk's UI users may be able to inject arbitrary code that may be executed by the application if the input is validated improperly.

**Inter-process communication interfaces:** KakaoTalk shares and receives data from other applications and on-device services (e.g., GCM broadcast receiver). A malformed IPC message may inject arbitrary code into the KakaoTalk application.

**Third-party themes:** The application allows users to install third-party themes that can be installed as separate APKs. Since KakaoTalk has been already targeted by trojanized applications in the past<sup>74</sup>, customized themes may be a new attack vector to spread malware.

---

<sup>73</sup>[https://en.wikipedia.org/wiki/Stagefright\\_\(bug\)](https://en.wikipedia.org/wiki/Stagefright_(bug))

<sup>74</sup><http://blog.trendmicro.com/trendlabs-security-intelligence/kakaotalk-targeted-by-fake-and-trojanized-apps/>

**External software dependencies:** KakaoTalk uses over one hundred<sup>75</sup> open-source and numerous closed-source native libraries. In addition, there are various external Kakao services<sup>76</sup> which can be connected to KakaoTalk by a central Kakao user account. According to Kakao<sup>77</sup>, there are up to 18,000 of such “Connected Services”. All these external services and software dependencies increase KakaoTalk’s attack surface by adding additional data entry and exit points.

### 4.3.2 Assets

In this section we deal with the question of which assets of the KakaoTalk messaging system do we want to protect against a potential threat. We answer this question by providing the following non-exhaustive list of assets that we believe are the most important:

1. User’s privacy
  - 1.1. Right to have control over Personally Identifiable Information (PII)
    - 1.1.1. Identifiers, contact and location information (login credentials, Kakao-ID, name, phone number, email address, device ID, and other identifiers)
    - 1.1.2. Private message content (text, voice, video, image, file, chat history, and other message content)
    - 1.1.3. Message and communication metadata (IP addresses, amount of sent/received messages, user’s social graph, and other meta-data)
  - 1.1. Right to be left alone
    - 1.1.1. User’s right to chat at any time
    - 1.1.2. User’s right to not be discriminated by other users
    - 1.1.3. Freedom of speech (user’s right to not be censored by any third-parties)
2. User’s money
  - 2.1. Money spent on in-app purchases

<sup>75</sup><https://katalk.kakao.com/android/cs/licenses?prev=1>

<sup>76</sup><http://www.kakaocorp.com/en/about/service>

<sup>77</sup>[http://www.kakao.com/provide\\_layer?page=1794&privacySearch=&lang=en#none](http://www.kakao.com/provide_layer?page=1794&privacySearch=&lang=en#none)

3. Kakao's reputation
4. Secondary assets (parts of the technical system): KakaoTalk/third-party back-end IT system, application source code, the user's mobile phone, and other secondary system assets.

### 4.3.3 Threat Agents

In this section we provide an overview of potential threat agents and their motivation of attacking the KakaoTalk messaging system. Similar to [126], we assume that all threat agents are part of KakaoTalk's chat community. Therefore, they can perform the same actions as regular chat users, e.g., adding and searching contacts or sending and receiving messages. In the following we list various possible attackers:

***Malicious/nosy KakaoTalk users or script kiddies:*** Want to save money, do not want to pay for in-app purchases. Want to commit random harm and annoy other KakaoTalk users. Want to track or stalk other users, want other users' personally identifiable information.

***Insiders:*** Insiders such as rogue Kakao employees. Want to make money or get other benefits. Want to harm the company's reputation.

***Kakao Corp:*** Wants to make money (e.g., wants the KakaoTalk user to sign up and pay for other Kakao services). Wants to collect personally identifiable information for analytic purposes. Wants to comply with national and international (surveillance) laws.

***Kakao Corp. competitor:*** Wants to increase market share. Wants to get access to KakaoTalk's source code. Wants to harm Kakao's reputation.

***Cooperating advertising companies:*** Want KakaoTalk users' personally identifiable information for targeted advertising.

***Local area network attackers (active/passive):*** Want KakaoTalk users' personally identifiable information. Want to exploit an KakaoTalk application bug to compromise the user's mobile device.

**Global network attackers (active/passive):** Attackers such as powerful government agencies or compromised Internet service providers. Want to gather evidence such as users' personally identifiable information. Want to exploit an KakaoTalk application bug to compromise the user's mobile device.

**Internet service provider:** Wants to make money. Wants KakaoTalk users' transport communication metadata to comply with (surveillance) laws.

**Mobile phone manufacturer:** Wants to make money. Wants KakaoTalk users' "hardware-burned" personnel identifies for tracking purposes.

**Google:** Wants to make money. Wants KakaoTalk users' personally identifiable information for tracking and advertising purposes.

**Criminals/organized crime:** Want to make money. Want KakaoTalk users' personally identifiable information. Want to exploit an KakaoTalk application bug to compromise the user's mobile device.

**A random attacker with physical access to the mobile phone:** Wants to make money. Wants to steal users' personally identifiable information. Wants to bypass two-factor authentication methods.

#### 4.3.4 Threats

Instant messaging systems such as KakaoTalk face a number of specific security threats [62, 89, 88]. In this section we provide a high-level list of security and privacy threats against the KakaoTalk messaging system. We summarize different threats in terms of what each potential attacker may achieve. Before we continue we make various security assumptions:

- 1 The user obtains an authentic copy of KakaoTalk.
- 2 The user's mobile phone is not compromised by malware.
- 3 The security assumptions of Ed25519, HMAC-SHA256, AES, and other cryptographic standards are valid.

***All attackers:***

- 1 Account sharing (not allowed according to Kakao's Terms of Service)
- 2 Spoofing/impersonation/identity theft
- 3 Sending malicious chat messages or files (e.g., spam)
- 4 Uninformed gathering of personally identifiable information (e.g., retaining other users' messages)
- 5 Uninformed disclosure or publication of personally identifiable information (e.g., sharing contact's private messages with third-parties)
- 6 False representation of personally identifiable information
- 7 Spreading harmful false information about an user
- 8 Social-Engineering attacks
- 9 Provide evidence to a third-party that a message came from a specific user (Non-Repudiation)
- 10 Learn about that a user has blocked another user
- 11 Learn about a user's social data that the user has chosen to make available

***By Kakao Corp or a compromised KakaoTalk server:***

- 1 Persistent tracking, learning user identity and route (e.g., storing cookies without user consent)
- 2 Deletion of user chat history that is stored on the server
- 3 Tampering of KakaoTalk messages (edit, replay, drop)
- 4 Blocking a user from participating in the messaging system
- 5 Authoritarian measures (e.g., "In order to use KakaoTalk you must agree to our privacy policy.")
- 6 Plaintext access to non end-to-end encrypted user chat
- 7 Changing the client applications' configuration settings remotely

- 8 Adding backdoors upon government requests (e.g., end-to-end encryption backdoor in KakaoTalk)
- 9 Attacks against key exchanges of KakaoTalk's end-to-end encryption feature
- 10 Kakao Corp. knows about the social graph of an user
- 11 Kakao Corp. can learn when a user is online (presence status)
- 12 Kakao Corp. can learn who sends messages
- 13 Kakao Corp. can learn when a user sends/receives messages
- 14 Kakao Corp. can learn how many messages a user sends and receives
- 15 Kakao Corp. can learn about the size of any transferred files
- 16 Kakao Corp. can learn about the file upload/download time
- 17 Kakao Corp. can spam a user with invalid messages for which it has the decryption keys
- 18 Kakao Corp. can drop or corrupt any uploaded files
- 19 Kakao Corp. can learn about users' actions

***By insiders:***

- 1 Can do everything what Kakao Corp. is able to do as well
- 2 Damaging Kakao Corp. reputation

***By Kakao Corp. competitor:***

- 1 Industrial espionage (e.g., reverse engineering of application source code)
- 2 Blocking a user from participating in the messaging system (e.g., DoS attack against a KakaoTalk server)

***By cooperating advertising companies:***

- 1 Persistent tracking, learning user identity and route (e.g., setting persistent third-party cookies)
- 2 Sending malicious advertisements (malvertising)

***By mobile phone manufacturer:***

- 1 Persistent tracking, learning user identity and route (e.g., by unique identifiers “burned” into the mobile phone’s hardware)
- 2 Adding backdoors upon government requests (e.g., baseband backdoors)

***By Google:***

- 1 Persistent tracking, learning user identity and route (e.g., through in-app purchases via Google Play or through Google Cloud Messaging)
- 2 Authoritarian measures (e.g., “If you want to publish your chat application on the Play Store you have to support Google Cloud Messaging.”)
- 3 Adding backdoors upon government requests (e.g., backdoor in Google Cloud Messaging)

***By local network attackers (active/passive):***

- 1 Learning about users’ actions (e.g., by performing traffic analysis)
- 2 Observing when a user is using KakaoTalk
- 3 Observing when a user is using KakaoTalk’s secret chat feature
- 4 Tampering of KakaoTalk messages (edit, replay, drop)
- 5 Blocking a user from participating in the messaging system (e.g., DoS attack against the KakaoTalk client or server)
- 6 Attacks against key exchanges of KakaoTalk’s end-to-end encryption feature
- 7 Network-based remote code execution attacks

***By global network attackers (active/passive):***

- 1 Can do everything what a local network attacker is able to do as well
- 2 Authoritarian measures (e.g., “In order to use KakaoTalk you must go through proxy X”)
- 3 Learning about who is using KakaoTalk and who is taking part in the same conversation

- 4 Learning about when messages are sent and when they are received
- 5 Observing metadata of file attachment uploads/downloads

***By a random person with physical access to the mobile phone:***

- 1 Obtaining users' encrypted messages stored in SQLite database if the device is offline and the block storage is not encrypted
- 2 Obtaining users' secret keys and access tokens from memory if the device is online and the lock screen can be bypassed
- 4 Privilege escalation attacks by using the phone in two-factor authentication procedures

***By criminals/organized crime/malware developers:***

- 1 Can do everything what an insider, a local network attacker and a person with physical access to the phone is able to do as well
- 2 Targeted malware attacks in order to achieve a mobile handset compromise
- 3 Plaintext access to *non* end-to-end encrypted user chat
- 4 Plaintext access to end-to-end encrypted user chat
- 5 Remote backdoor access to the mobile phone
- 6 Use KakaoTalk to spread malware to other contacts
- 7 Deletion of user chat history

## 4.4 Vulnerability Analysis and Findings

In this section we list the results from our automated and manual security analysis of the KakaoTalk chat application. We start by discussing KakaoTalk's end-to-end encryption protocol (Section 4.4.1) and proceed by analyzing the application's specific software vulnerabilities in Section 4.4.2. Finally, we conclude our vulnerability analysis by presenting KakaoTalk's information privacy issues in Section 4.4.3.

### 4.4.1 End-to-end Encryption Protocol Analysis

If the reader recalls the goals of a secure instant messaging system from Section 2.1 we argue that it is fair to say that KakaoTalk including its “Secret Chat” feature is *not* a secure instant messenger. We speculate that the main reason for this fact is Kakao’s rush of implementing an E2E encryption protocol within two months in order to not possibly lose more users to other competing messengers such as Telegram (see Section 3.2.3.1). Also, Kakao built the E2E encryption feature on top of their existing messaging infrastructure and did not implement an entirely new protocol from scratch. This approach led to a system design which is prone to multiple attacks that may lead to the decryption of end-to-end encrypted messages. In the following we list a number of flaws that we have discovered in KakaoTalk’s end-to-end encryption messaging system:

**System architecture:** The probably most obvious design flaw is that Kakao uses a central public-key directory server which makes the E2E encryption messaging system prone to MITM attacks on the operator-site. This attack may remain undetected if users do not compare their public key fingerprints. In addition, MITM attacks can be also performed *after* two clients have agreed on a shared secret key since Kakao has the ability to delete E2E encrypted chat rooms on the server-side. In this case the user would need to create a new chat room and the attacker would be then able to substitute a user’s public key during the initial secret key exchange (see Figure 4.12). Most of the secure messaging systems that exist today operate a public-key directory server, e.g., WhatsApp, Signal, or iMessage<sup>78</sup>.

As discussed in Section 4.2.3, all IM communications are relayed through the LOCO messaging backend which decrypts packets, creates new ones, and forwards them re-encrypted to the destination. Due to the interception and transformation of messaging packets there is no integrity verification implemented on the client-side. Otherwise, clients would constantly throw an exception that messages have been tampered during network transmission. The only piece of information that is protected by a MAC is the actual chat message (see Section 4.2.3.3). However, KakaoTalk uses the concept of MAC-then-encrypt which means that the ciphertext integrity remains unprotected. Worse, all other data fields in KakaoTalk’s E2E encryption protocol do not have any integrity protection mechanisms (see Sections 4.2.3.1 and 4.2.3.3). The lack of integrity protection means that any local or global

---

<sup>78</sup><http://blog.cryptographyengineering.com/2013/06/can-apple-read-your-imessages.html>

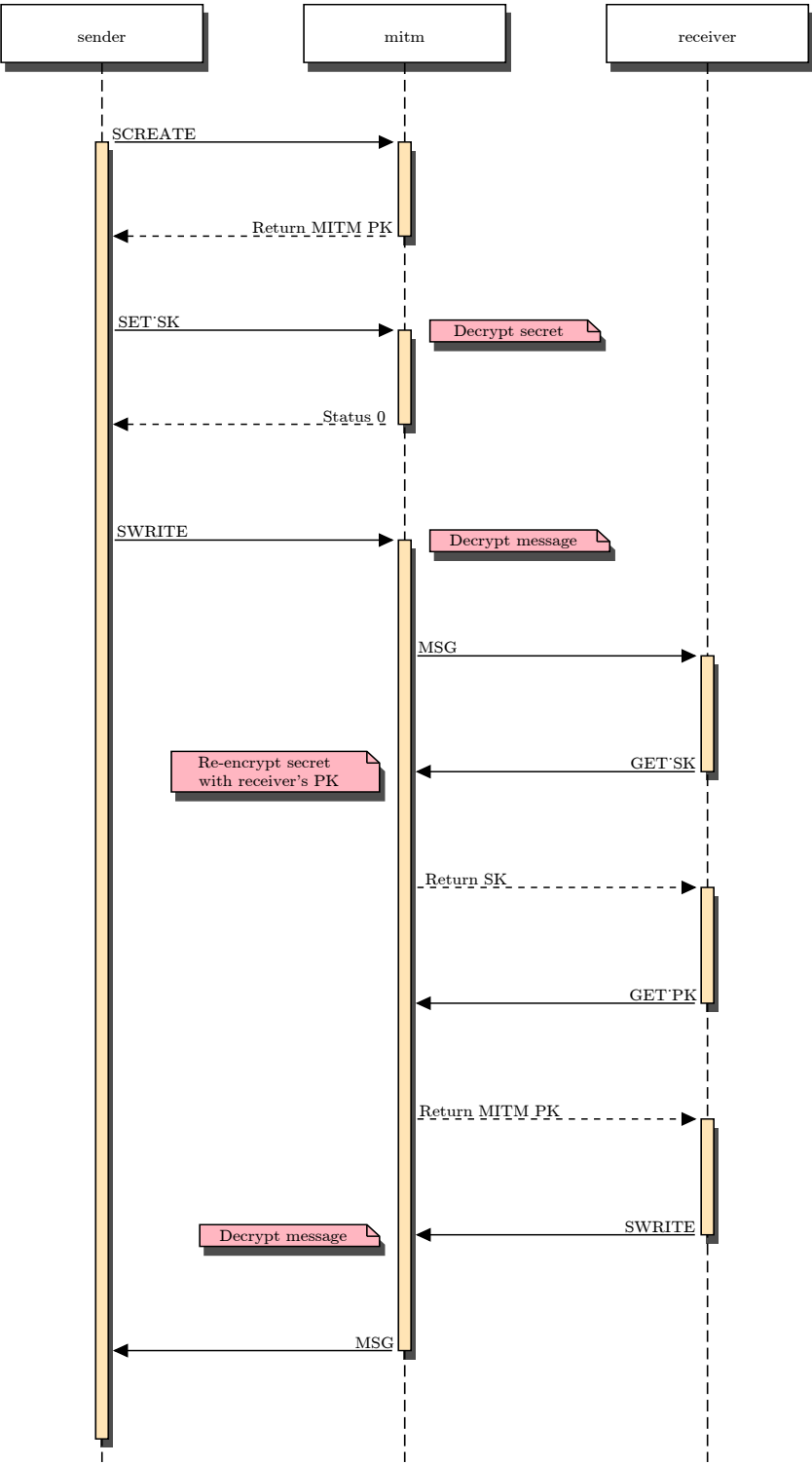


Figure 4.12: MITM attack on the operator-side.

network MITM attacker could manipulate the ciphertext without being detected. And since KakaoTalk’s LOCO “encrypted” packet uses malleable [40] block cipher modes (e.g., AES-CBC or AES-CFB), “bit-flipping” attacks<sup>79</sup> such as flipping a status code bit to change message flow behaviour, may be possible.

Another major design flaw of KakaoTalk’s messaging system is that there is no server authentication of the LOCO messaging backend. While Kakao’s Web API backend utilizes the HTTPS protocol to guarantee server authentication, the LOCO messaging backend uses the LOCO protocol which does not support such a security concept. Since there is no need for server authentication, the KakaoTalk client would blindly trust the malicious endpoint. Moreover, a malicious server could send legitimate LOCO commands to the KakaoTalk client application to possibly retrieve sensitive information or change the communication flow.

Furthermore, a number of other weaknesses in KakaoTalk’s messaging architecture exist. One flaw is that Kakao uses SMS to verify a user’s phone number to authenticate users even though SMS has been long known to be insecure. Another weakness is that Kakao’s Web API backend allows unauthenticated calls over HTTP. Even though we were not able to force KakaoTalk to use HTTP instead of HTTPS connections, it does not mean that KakaoTalk may not fallback to use HTTP under certain circumstances that we have not tested (e.g., older Android or KakaoTalk versions). In addition, we recognized that KakaoTalk sometimes transmits the LOCO CHECKIN packet (see Section 4.2.3.2) in plaintext. A MITM attacker may be able to modify the server reply of this packet in order to point the KakaoTalk client to a malicious IM server. However, we must admit that we were not able to reproduce this behaviour.

**Static values:** During our source code analysis we recognized a number of static values that are used for message encryption and user authentication. For instance, we found that KakaoTalk uses the hard-coded Initialization Vectors “locoforever” or “KaKAOtalkForever” if the application chooses to use AES-CBC for message encryption (see LOCO handshake packet in Section 4.2.3.1). This means that the ciphertext is not randomized and that an attacker may spot plaintexts that result in identical ciphertexts. Another issue that we found was that KakaoTalk uses the static String value “e2dc694aee2540c2de6b4a8be2d7718846a0dfb9” in case it cannot determine the mobile handset’s device UUID. This might be an issue since the device UUID is used in the HTTPS authorization header (alongside with an

---

<sup>79</sup>[https://en.wikipedia.org/wiki/Bit-flipping\\_attack](https://en.wikipedia.org/wiki/Bit-flipping_attack)

access token) as well as for mapping the user’s public keys to an user identity. Finally, we found that KakaoTalk uses the static hard-coded salt value “53656372657443686174526f6d4b6579” that is used together with a random shared secret value to compute the E2E encryption key. Since this value can be easily obtained by decompiling the application, an attacker may be able brute-force an E2E encryption key more easily.

**Known plaintexts:** The LOCO messaging protocol leaks various known plaintexts which makes it vulnerable to known plaintext attacks (KPA). As explained in Section 4.2.3.1, most of the data fields of the LOCO messaging protocol contain known and static plaintext entries (e.g., packet length, LOCO command, ID, and other entries). As a result, the LOCO messaging protocol may be vulnerable to a number of plaintext recovery attacks which do not require access to the encryption key [57].

Moreover, by observing encrypted messaging traffic we found that the first 15 bytes of the first ciphertext block of an encrypted LOCO packet are always static if KakaoTalk is using AES in Cipher Feedback mode. Only the 16th byte of the block is randomized each time a new packet is sent. The fact that the first plaintext block is static and the way how Cipher Feedback mode encryption works<sup>80</sup>, means that KakaoTalk most probably increments the Initialization Vector (IV) for each message (see Figure 4.13).

```
00 00 00 CE E8 F5 6F EF AE 55 A4 38 9 65 A8 10 77 48 F2 5E
00 00 00 CE E8 F5 6F EF AE 55 A4 38 9 65 A8 10 77 48 F2 54
```

Figure 4.13: The first 15 most significant bytes of an encrypted LOCO packet remain static (highlighted in red color). The packet structure starting from the most significant byte is as follows: Length (4 bytes) + ID (4 bytes) + Status (1 byte) + Command (11 bytes).

**Missing security goals:** As already mentioned, KakaoTalk is missing important security goals that are crucial for a secure instant messaging system. We found that KakaoTalk lacks the following secure IM properties:

*Forward Secrecy:* After a certain amount of time, KakaoTalk refreshes the “E2E encryption key” and presents a warning message to the user mentioning that the chatroom is no longer available as the user’s “encryption information has expired”. However, the RSA key-pair, which is

<sup>80</sup>[https://en.wikipedia.org/wiki/Block\\_cipher\\_mode\\_of\\_operation#Cipher\\_Feedback\\_.28CFB.29](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Feedback_.28CFB.29)

used to exchange a new secret value that in turn is used to generate a new message encryption key, is long-lived and is not replaced automatically. An attacker who has obtained access to the user's private RSA key and to the provider-shared AES key may be therefore able to decrypt previously collected end-to-end encrypted messages. For this reason, KakaoTalk does not support Forward Secrecy.

*Backward/Future Secrecy:* Similarly, KakaoTalk does not provide Backward/Future Secrecy since the user's RSA key-pair remains static.

*Offline Message Deniability:* Messages are signed with the user's long-lived private signature key. This means that each message can be clearly linked to the message's sender. The sender could *not* later deny having sent a particular message.

*Metadata protection:* KakaoTalk does not protect the communication metadata, but instead possibly makes use of it by creating complex social graphs<sup>81</sup>. However it is worth to note that all of the mobile secure messengers that exist today do not provide metadata protection.

*E2E messaging by default:* Instead of encrypting all messages end-to-end by default, KakaoTalk's "Secret Chat" feature is opt-in only and users need to explicitly choose to chat in an end-to-end encrypted manner.

*Open-source, open documentation, independant security audit:* KakaoTalk's end-to-end encryption protocol is closed-source and does not provide an open protocol specification. Also, to the best of our knowledge, the application and protocol have not been audited by an independent party.

**Missing freshness:** While we had not the time to test this in depth, we found evidence that the protocol may not provide freshness to prevent replay attacks (e.g., by adding a random nonce to each message). In a MITM scenario we were able to perform a simple replay attack against KakaoTalk's E2E encryption protocol. We were successful in replaying and reordering packets, as well as replacing ciphertext blocks. The latter chosen ciphertext attack (CCA) worked even though the LOCO messaging server or client auto-corrects this misbehaviour by resending the original message. However, we did not try to mirror or withhold particular messages. A more serious issue may be the possibility that an attacker who stole a KakaoTalk's user mobile

---

<sup>81</sup>[https://en.wikipedia.org/wiki/Social\\_graph](https://en.wikipedia.org/wiki/Social_graph)

handset, may be able to replay previously captured encrypted packets to the device in order to obtain the plaintext.

#### 4.4.2 Android Software Security Analysis

In this section we present our findings that we discovered in addition to examining the security of KakaoTalk’s E2E encryption protocol. KakaoTalk’s main software vulnerabilities are as follows:

***Insufficient transport layer protection:*** KakaoTalk utilizes the HyperText Transfer Protocol Secure (HTTPS) protocol for most network connections in order to prevent passive eavesdropping of the communication link. The application takes a number of important steps to validate the server’s X.509 certificate: First, it checks whether the certificate’s Common Name (CN) matches the domain’s name. If they do not, KakaoTalk throws a *CertificateException* and rejects establishing a TLS connection to the server. Second, KakaoTalk verifies whether the server’s certificate has been signed by a trusted Certificate Authority (CA). For this, the application checks if the signing CA certificate has been signed by a trusted Root CA certificate which is located in Android’s trusted CA store. At this point, an attacker may be still able to compromise a TLS session by luring the user to copy his own malicious CA certificate into the victim’s trusted CA store. For this reason, KakaoTalk also verifies the certificate chain by maintaining a whitelist of public keys that are trusted to sign certificates. This list is consulted when KakaoTalk validates the certificate chain, and if it does not include at least one of the *pinned* keys, certificate validation fails. This security concept is called “public key pinning” or “certificate pinning” and was proposed in RFC 7469<sup>82</sup> by Google in 2015.

Despite of implementing all of these certificate validation techniques, KakaoTalk mostly accepts self-signed MITM certificates and only shows a certificate validation error message that a user might not understand. The pop-up message says that “There are problems with the security certificate for this site” and allows the user to either accept the error or to go “Back”. After accepting the dialog, KakaoTalk continues working and silently accepts invalid TLS certificates in the background. Even though there are warning messages, it is well known that users often ignore security errors [11] mostly because of habituation [78].

We could reproduce the improper certificate validation for the domains `auth.kakao.com`, `auth.kakaocdn.net` and `katalak.kakao.com`. For other do-

---

<sup>82</sup><https://tools.ietf.org/html/rfc7469>

mains KakaoTalk rejects establishing a TLS connection. For instance, for `item.kakao.com` that is used by the Item Store, KakaoTalk does not present a warning dialog, but instead shows a blank WebView with a message saying “Unstable network connection”. In the following, we describe the particular certificate validation issues that we found during our assessment:

**Improper certificate validation during user registration:** As already mentioned in Section 4.2.3.2, KakaoTalk issues several HTTPS POST requests during the user registration process. In particular, the application establishes a TLS connection to a registration server (`ac-talk.kakao.com`) and to an authentication server (`auth.kakao.com`). Although KakaoTalk uses public key pinning for `ac-talk.kakao.com`, it does not do so for `auth.kakao.com`. Also, it does not verify the Common Name of the latter server since KakaoTalk accepts a MITM certificate that is self-signed.

**WebViews ignore certificate validation warning messages:** The app stops employing public key pinning when establishing a TLS connection to the Web API backend after the registration process. Instead, it validates the certificate chain of a server *without* consulting the list of whitelisted public keys. If KakaoTalk detects that the server’s certificate chain is invalid, it writes a *java.security.cert.CertPathValidator-Exception* warning message to the system log and notifies the user with a simple dialog. In addition, we found that some WebViews accept a certificate even though it contains a CN that does not match the server’s hostname. One example is KakaoTalk’s privacy policy and terms of service which are displayed in a WebView that loads the HTML from `https://1.201.0.61/android/account/privacy.html?locale=en_de` and `https://1.201.0.61/android/account/terms.html?locale=en_de`. Even though the server cannot prove that it is “1.201.0.61” as its certificate is from “\*.kakao.com”, the WebView does not notify the user about this misbehaviour.

In addition to the certificate validation issues described above, we found that some TLS endpoints of KakaoTalk’s Web API backend are misconfigured. For this analysis, we used Qualys TLS Server Test<sup>83</sup> in order to scan the backend for common TLS configuration flaws. We tested the domains `ac-talk.kakao.com`, `auth.kakao.com`, `katalink.kakao.com`, and `auth.kakao.net` and retrieved the following scanning results:

`auth.kakao.com`

---

<sup>83</sup><https://www.ssllabs.com/ssltest/>

- The server is vulnerable to the POODLE attack [95]
- The server is vulnerable to the DROWN attack [4]
- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107)
- Weak certificate signature algorithm (SHA1withRSA)
- The server supports weak cipher suites (e.g., `TLS_RSA_WITH_RC4_128_MD5`)
- The server does not support Forward Secrecy

ac-talk.kakao.com and auth.kakao.com

- The servers are vulnerable to the DROWN attack [4]
- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107)<sup>84</sup>
- Weak certificate signature algorithm (SHA1withRSA)
- The servers support weak cipher suites (e.g., `TLS_RSA_WITH_RC4_128_MD5`)
- The servers do not support Forward Secrecy

auth.kakaocdn.net

- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107)
- The server supports weak Diffie-Hellman (DH) key exchange parameters
- Weak certificate signature algorithm (SHA1withRSA)
- The server does not support Forward Secrecy

---

<sup>84</sup><https://www.openssl.org/news/secadv/20160503.txt>

Finally, we detected that KakaoTalk does not use TLS for all network connections. For instance, the application does not use HTTPS to secure the communication channel on older versions of Android. On a mobile handset that ran Android 4.1.2 we found that all communications to KakaoTalk's Web API backend are handled by the HTTP protocol only. Moreover, while running KakaoTalk on a more recent Android version, we recognized that some web content is still being served via HTTP. For instance, KakaoTalk downloads unauthenticated HTML ([http://1.201.0.61/android/help?lang=en&country\\_iso=DE&a=android%2F5.5.5%2Fen](http://1.201.0.61/android/help?lang=en&country_iso=DE&a=android%2F5.5.5%2Fen)) and JavaScript ([http://ad1-kant.kakao.com/track\\_jquery3.js](http://ad1-kant.kakao.com/track_jquery3.js)) when the user opens the `com.kakao.talk.activity.setting.HelpActivity` and `com.kakao.talk/.activity.setting.NoticeActivity`. We also noticed that several resources including images and zip archives are being downloaded unencrypted. This would allow a MITM attacker to inject arbitrary code that may be then executed on the user's mobile phone (see "Improper WebKit WebView implementation" below).

***Improper WebKit WebView implementation:*** As a hybrid mobile application, KakaoTalk makes heavily use of WebKit which is an open source web browser engine. KakaoTalk uses the framework's core view class `WebView`<sup>85</sup> to display web content that the application downloads from backend servers. We discovered a number of possible vulnerabilities due to the way how KakaoTalk has implemented its WebViews: On the one hand, the application uses various methods of the `android.webkit.WebSettings` class to configure its WebViews. We found that for almost all WebViews KakaoTalk is calling the following methods:

- `setJavaScriptEnabled()`: This method allows to execute JavaScript within a `WebView`.
- `setAllowFileAccess()`: This method allows the application to read cached web content from the file system. This feature is enabled by default if the developer has not explicitly disabled it.
- `setPluginState()`: In the `InAppBrowserActivity` this method is called to enable plugin support (e.g., to display Adobe Flash content).

On the other hand, KakaoTalk uses the `addJavascriptInterface()` public method of the `WebView` class. The method implements a Java to JavaScript bridge – which KakaoTalk declares either as “kakaotalk”, “kakao-Talk”, or “Kakao” – that can be used to allow JavaScript code to control

---

<sup>85</sup><https://developer.android.com/reference/android/webkit/WebView.html>

the native application. This feature makes KakaoTalk vulnerable to possible remote code execution attacks if an attacker uses the bridge to access the `getClass()` method which is inherited from the `Object` class<sup>86</sup> (see CVE-2012-6636<sup>87</sup> and CVE-2013-4710<sup>88</sup>). The attack works on devices running Android versions older than 4.2 while later versions may be also vulnerable since KakaoTalk adds the `@JavascriptInterface`<sup>89</sup> annotation to some public methods. This annotation is for example used in the `CommonWebLayout` and `PlusFriendListWebActivity` classes. Worryingly, not only a MITM attacker but also any third-party brand or pop artist that advertises its products through KakaoTalk’s “Plus Friend” marketing channel is able to access the JavaScript bridge. This means that any third-party may be able to execute arbitrary code if Kakao does not verify the integrity of web scripts prior publishing them on “Plus Friend”.

As a result of these WebView misconfigurations, an attacker may be able to inject arbitrary JavaScript (JS) code since KakaoTalk makes use of unauthenticated channels and also accepts invalid TLS certificates. Moreover, JS may be injected through various other channels such as Bluetooth, SMS messages or malformed multimedia files [73]. In a quick test, we injected simple JS statements such as `alert(document.cookie);` which were then executed in the context of the WebView. Also, we injected a false login page that may be used to phish user credentials and were able to retrieve a list of available HTML5 file objects (`window.File`, `window.FileReader`, and `window.FileList`). These objects may allow to read files from the file system even though we were not successful most probably due to same-origin policy (SOP) restrictions. That being said, exploitation might not always be possible as the level of compromise depends on which mobile platform KakaoTalk is running and on the context in which the particular WebView is used. However, concepts such as SOP were successfully bypassed in the past<sup>90</sup>.

***Possible Remote Code Execution using arbitrary file writes:*** First and foremost we have to admit that we were not able to turn this vulnerability into a remote code execution attack due to timing constraints. However, an attacker with more resources may manage to exploit this vulnerability successfully. In the following paragraphs we describe the vulnerability in

---

<sup>86</sup><http://d3adend.org/blog/?p=314>

<sup>87</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-6636>

<sup>88</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4710>

<sup>89</sup><http://stanford.edu/~pcm2d/blog/jsbridge.html>

<sup>90</sup><http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html>

more detail:

As discussed, KakaoTalk constantly downloads various resource files via unencrypted channels or through TLS connections that can be compromised by an attacker. The application typically first fetches a JSON file which includes file attributes such as file names and file sizes of resources that are further being downloaded from either `https://http://item.kakao.com/` or `http://item-kr.talk.kakao.co.kr`. Some of these resources include ZIP archives which package multiple files into one file. To the best of our knowledge, KakaoTalk does not perform any additional integrity checks of these ZIP archives. The lack of integrity verification means that a MITM attacker may inject arbitrary files into a ZIP archive. In addition, an attacker may control the path of where the injected file will be extracted on the device by naming a file `../../../../../../../../data/data/com.kakao.talk/malicious-file`. This way, an attacker may be able to achieve an arbitrarily file write into KakaoTalk's private data directory since KakaoTalk does not prevent from this kind of directory traversal attacks<sup>91</sup>. As a result, an attacker may be able to replace existing configuration files or executable binaries. Ryan Welton has been successful in turning this flaw into a remote code execution attack by replacing a DEX file that was located in the application's `/files/` directory<sup>92</sup>. However, in the case of KakaoTalk there is no additional DEX file that could be patched using this way. Instead, we propose two ideas on how a remote code execution attack may be successfully achieved:

**Native library patching:** Android stores KakaoTalk's native libraries in `/data/app-lib/com.kakao.talk-1/` and creates the symbolic link `/data/data/com.kakao.talk/lib` that points to this directory. KakaoTalk loads most native libraries implicitly by using the `System.load()` or `System.loadLibrary()` methods which load a native library from a path stored in the `LD_LIBRARY_PATH` environment variable. Since an attacker is not able to modify the `LD_LIBRARY_PATH` variable and does not have write permission to the `/data/app-lib/` directory, there is no chance to overwrite or patch a native library to achieve code execution. However, we found that KakaoTalk loads the `libNSaferJNI.so` and the `liba3030.so` libraries explicitly by providing a full path to the application's private library directory `/data/data/com.kakao.talk/lib/` which would be writable by KakaoTalk. The two shared libraries

---

<sup>91</sup>See <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2412> for an example directory traversal attack.

<sup>92</sup><https://www.nowsecure.com/blog/2015/06/15/a-pattern-for-remote-code-execution-using-arbitrary-file-writes-and-multidex-applications/>

are loaded by the `com.nshc.NSaferJNI` and `com.ahnlab.a3030.a3030` classes which are used by Kakao Pay. Now that we found a library that could possibly be patched and side-loaded<sup>93</sup>, an attacker may overwrite the symbolic “lib” link with a folder which has the same name and which an attacker may inject into the intercepted ZIP archive. This “lib” folder would contain a specially crafted version of either `libNSaferJNI.so` or `liba3030.so` which would be explicitly loaded if the user opens the Kakao Pay Activity. By performing some simple tests, we detected that Android does not verify the integrity of the loaded library, but only compares the file size of the original library shipped with the APK with the one placed in `/data/data/com.kakao.talk/lib/`. If the file sizes do not match, Android replaces our modified library with the original version. However, if the file sizes match, Android loads the malformed library successfully. In order to keep the original file size, we propose two options: First, replacing the entire library with a malicious one. However, this approach would possibly break the execution flow of Kakao Pay. Second, patching the ELF binary object of the library to load an additional dependency library which is under the attacker’s control. The advantage of patching the ELF object is that it does not change the file size of the native library. As explained in [92], an attacker may modify the `.dynamic` section of the ELF binary by modifying two parameters: First, changing the optional library dependency flag from “SONAME” to “NEEDED”. Second, changing the path of the optional dependency library from “optional-dependency.so” to `/data/data/com.kakao.talk/malicious-lib.so`. The “malicious-lib.so” library may contain a `JNI_Onload()` function which would execute malicious code when the library is loaded into the KakaoTalk process. However, despite that we were successful of loading our malicious dependency library, we were not able to execute the `JNI_Onload()` method because it is only called on the library that was loaded by the `System.load()` or `System.loadLibrary()` function call.

The drawback of the library side-loading attack would be that a user needs to explicitly open the Kakao Pay Activity to load the malicious library. Another disadvantage would be that this method does not survive a reboot of the mobile handset as KakaoTalk’s `/lib/` directory would be replaced by the original symlink upon system boot. Yet, there is another challenge when specifying the explicit directory path of our malicious library inside the ELF binary object: We only have

---

<sup>93</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0854>

a limited amount of space to control and specify the path of the library. We cannot solve this problem by using relative paths because the `System.load()` and `System.loadLibrary()` methods do not support them. Furthermore, we cannot place and load our malicious library to and from the SD card as it has the NX bit set.

**Patching the “lowell” binary:** KakaoTalk creates a world-readable and executable traceroute binary named “lowell” under the `/files/` directory when a user executes a network test located under “Settings” → “Network Test” → “Start Test”. We recognized that KakaoTalk does *not* overwrite the binary each time a user starts the network test from the beginning. This may mean that an attacker may be able to obtain remote code execution by reverse engineering and patching the binary.

**Results of our forensic analysis:** While examining the mobile handset’s file system and physical memory, we found numerous artifacts that may allow an attacker who is able to get a hold of the device, to obtain sensitive information or impersonate a KakaoTalk user. We found that KakaoTalk uses a shared preference file which stores various configuration settings. For instance, the file stores a session identifier which is used to authenticate the KakaoTalk application. In addition, KakaoTalk stores private and public RSA and Ed25519 keys in the `TalkKeyStore.preferences.xml` file in plain-text. With these key-pairs an attacker may be able to decrypt the shared secret value that is being used to compute the E2E encryption key. Moreover, an attacker may be able to forge valid Ed25519 signatures to impersonate a KakaoTalk user. If the mobile phone’s file system is not encrypted and has adb debugging enabled, an attacker may be able to obtain a copy of KakaoTalk’s files even from a non-rooted device. We also recognized that KakaoTalk uses the `file.delete()` method to delete files from the file system. However, it is well known that this may be an insecure way of deleting files [125]. Furthermore, we were able to extract the user’s email address, device UUID and several OAuth tokens from the handset’s physical memory. This is also possible on a non-rooted device by using the Memory Monitor which is part of Android Studio. With these artifacts found on the user’s mobile phone, an attacker may be able to perform the following actions:

- Web API and LOCO messaging server login using the user’s identity.
- Sending chat messages on a user’s behalf.
- Access encrypted chat history.
- Update and change server-side stored KakaoTalk configuration settings.

**Miscellaneous issues:** In addition to the previously discussed vulnerabilities, we found a number of additional issues:

- KakaoTalk exports 31 activities and four broadcast receivers with no permissions. These application components may be invoked by other applications to obtain sensitive data or to inject malicious code.
- KakaoTalk's OAuth access token may lack randomness (see Section 4.2.3.2).
- Other than using ProGuard for code obfuscation, KakaoTalk does not employ anti-reversing techniques. There is no root detection, emulator detection, or a mechanism for detecting an attached debugger.
- Even though KakaoTalk encrypts some database entries, it does not encrypt its SQLite database as a whole (e.g., by using SQLCipher).
- A four-digit pin that can be used to protect a KakaoTalk installation can be easily brute forced or guessed<sup>94</sup>.
- Prior Android 4.1 any application that declares the `READ_LOGS` permission is able to read the log files of any other application. Since KakaoTalk logs a number of events, this may be abused by an attacker to obtain sensitive information.
- KakaoTalk creates a world-readable and executable traceroute binary file within its private data directory if the user starts a network test.

### 4.4.3 Information Privacy Analysis

We are finishing our technical analysis chapter by providing a list of system weaknesses that may violate user's privacy. The particular privacy violations that we found are as follows:

**Information leakage:** We discovered that KakaoTalk leaks data through unauthenticated channels. First, KakaoTalk sends crash reports to a remote server in case there are issues such as a bad JNI memory access, a Java null pointer exception, or other errors. These reports are sent to the `http://group1.magpie.daum.net/magpie/put/` URL unencrypted over HTTP and include sensitive information such as the detailed software and hardware

---

<sup>94</sup><http://www.datagenetics.com/blog/september32012>

configuration of the user's device as well the error cause, user ID, environment variables and other data. Another source of information leakage is KakaoTalk's network test. This test also sends a report over an unauthenticated channel to a PHP script located at <http://nettest.kakao.com/up.php>. The network test report includes the user ID, mobile OS and KakaoTalk version as well as the mobile country code. Finally, KakaoTalk may unintentionally leak sensitive information through its usage of various permissions. For instance, the `ACCESS_WIFI_STATE` permission may leak data that may be abused to track individuals [1]. Regarding KakaoTalk's permission usage, we found that KakaoTalk constantly tries to access the user's private phone address book even if the user has disabled the application's feature to sync the address book with the Web API backend. We discovered that KakaoTalk accesses the address book in cases where it does not seem necessary, e.g., while browsing through the application's main UI tabs. Summing up, our found information disclosures may not seem critical, however we argue that even mild privacy leaks may uniquely identify a KakaoTalk user when they are combined together.

**User tracking:** First of all, KakaoTalk does not provide anonymous messaging. The application requires the user's phone number in order to sign-up for the service and a private e-mail address in case a user wishes to use additional services. Also, Kakao links the randomly generated device UUID against the user's phone number. In addition, we found evidence that KakaoTalk is tracking the way how a user is using the application. For instance, it tracks the time a user has spent on certain Activities in a database. Moreover, it runs a web cookie database which stores up to six different cookies that are linked to different domains including `tiara.daum.net` and `tiara.kakao.com`. To the detriment of the user's privacy, there is no mechanism to control these cookies as KakaoTalk does not provide a configuration setting to change cookie tracking policies (i.e., configuration options that would allow to accept all cookies, block third-party cookies, or disable all cookies). KakaoTalk also downloads external JavaScript that is most probably used for tracking. For example, scripts including <http://m2.daumcdn.net/tiara/js/td.min.js> and [http://ad1-kant.kakao.com/track\\_jquery3.js](http://ad1-kant.kakao.com/track_jquery3.js) are being downloaded onto the device to log actions such as which news entries the user has tapped on within the *NoticeActivity*.

**Social graph:** As listed in Section 4.2.2, Kakao has access to a large amount of user data (e.g., friends lists or chat metadata). This data set

allows Kakao to build a user's social graph<sup>95</sup> which means that the service knows about the user's social relationships. There is evidence that KakaoTalk is building a user's social graph as the application recommends potential friends that the user may know, e.g., by recommending a person who shares the same common friend.

**Spam:** In most messengers a new contact needs to be explicitly authorized by the user before the contact is allowed to add the user to her friend list. This feature prevents random persons from adding arbitrary contacts in order to send spam messages. KakaoTalk does not enable this feature by default which means that strangers may be able to add random contacts without requiring their explicit consent. This may allow a chat bot to automatically add contacts to spread spam messages.

## 4.5 Comparison Between Terms of Service Claims and Technical Findings

In this section we compare our technical findings from Section 4.4 with Kakao's public marketing claims from Section 3.3. For each question we raised in Section 3.3.3, we try to provide a detailed answer. However, we cannot answer all questions since we were facing timing constraints.

### End-to-end encrypted chat

- Despite Kakao's claims, would the company be able to decrypt end-to-end encrypted messages?

**Answer:** Our answer would be a simple "Yes" (see Section 4.4.1).

### Encryption and data protection

- Does KakaoTalk use encryption for all communication channels?

**Answer:** KakaoTalk does not always use encryption. Sensitive data leakage occurs due to unencrypted (HTTP) or improperly secured network channels (vulnerable HTTPS connections).

- Does KakaoTalk protect against user impersonation or identity theft?

**Answer:** If an attacker gets a hold of the user's mobile handset or otherwise is able to obtain authentication tokens from the device's

---

<sup>95</sup><https://developers.facebook.com/docs/graph-api>

physical memory/storage, an adversary may be able to impersonate a user successfully. Users who suspect a KakaoTalk account compromise may be able to deactivate their stolen account by contacting Kakao's IT support<sup>96</sup>.

### Collection of user information

**Data minimization:** Does KakaoTalk collect more information than it claims?

**Answer:** In our opinion, KakaoTalk stores more data than absolutely necessary (see Section 4.2.2).

**Data utilization:** Does KakaoTalk utilize user data for any other purposes?

**Answer:** No evidences found.

**User consent:** Does KakaoTalk require user consent? Are the policies the same as the ones published on Kakao's website?

**Answer:** KakaoTalk requires user's consent upon user registration. However, we did not find an option that KakaoTalk informs the user about privacy policy or terms of service changes. In addition, in some cases KakaoTalk transmits data without the user's explicit consent (e.g., when sending crash reports that contain detailed hardware and software information about the user's handset).

### Collection of user information from third-parties

- Does KakaoTalk collect any user information from third-parties (e.g., by using third-party advertising libraries)?

**Answer:** We did not find evidence that KakaoTalk is collecting user information from third-parties. KakaoTalk 5.5.5 does not make use of third-party advertising libraries.

### Sharing of user information

- Does KakaoTalk share more information with third-parties than officially claimed?

**Answer:** We did not find evidence that KakaoTalk is sharing more information with third-parties than officially claimed.

---

<sup>96</sup>[http://www.kakao.com/account\\_theft\\_requests?locale=en](http://www.kakao.com/account_theft_requests?locale=en).

- Does KakaoTalk share user information with any other third-party not mentioned in the privacy policy?

**Answer:** We did not find evidence that KakaoTalk is sharing user information with any third-party not mentioned in the privacy policy.

- Tracking: What about third-party cookies that are used by KakaoTalk? For what are they used?

**Answer:** KakaoTalk does not use third-party tracking cookies.

- Which data does KakaoTalk share with other Kakao services? According to [102], Kakao does not clearly disclose whether and how it shares user information between different Kakao services.

**Answer:** Kakao clearly discloses which third-parties are provided with the user's personal information<sup>97</sup>.

#### User control over information collection and sharing

- What online storage system does KakaoTalk use, Daum's or Kakao's? This is critical because the number of data requests for Daum and Kakao vary significantly [34].

**Answer:** KakaoTalk uses both online storage systems (Kakao and Daum).

#### User's access to their own information

- What kind of data is accessible by a KakaoTalk user?

**Answer:** The user has access to the number of "Connected Services" that are linked to a particular KakaoTalk user account.

#### Retention of user information

- What about any data that is left on the SD card after KakaoTalk was removed from an Android device?

**Answer:** After removal, KakaoTalk leaves an encrypted cookie as well as some non-critical web contents on the SD card. If the user has created a message backup, the backup file would remain on the SD card as well.

---

<sup>97</sup>[http://www.kakao.com/provide\\_layer?lang=en](http://www.kakao.com/provide_layer?lang=en)

- KakaoTalk’s privacy mode: Does Kakao store KakaoTalk messages longer than three days?

**Answer:** We did not have the time to verify if Kakao stores unread messages longer than three days.

### Spam protection

- Does KakaoTalk or any other associated third-party monitor users’ communications to enforce its operation policy? If yes, how do they block/filter/delete spam or other malicious content?

**Answer:** While we did not have the resources to test KakaoTalk’s spam protection feature, we found evidence that the application may block certain top-level domain (TLD) on the client-side. We found a Java class that contained a number of hard-coded TLDs. However, we did not verify if the code is used by the application to block domains.

- Juvenile protection: How does KakaoTalk enforce Articles 41-43 of the *Promotion of Information and Communications Network Utilization and Information Protection Act*<sup>98</sup>?

**Answer:** While we believe that there must be a form of server-side or client-side keyword blocking mechanism to filter malicious messages, we did have the time to investigate this further. Kakao has publicly said that the company scans for malicious keywords and URLs and filters them accordingly.

---

<sup>98</sup>[http://elaw.klri.re.kr/kor\\_service/lawView.do?hseq=32543&lang=ENG](http://elaw.klri.re.kr/kor_service/lawView.do?hseq=32543&lang=ENG)

## Chapter 5

# Evaluation

First and foremost, we must admit that we were not able to perform an in-depth evaluation of our own work. The main reason was that we did not have the time and resources to undertake the best possible evaluation. However, what we will do in this chapter is to describe how an ideal evaluation *could* look like. Also, we list various other reasons that explain why we were not able to evaluate our work and outline what we did instead.

As discussed in Section 1.2, we identified two main solutions (or outcomes) of this Master thesis and formulated a corresponding goal that each solution should accomplish. Furthermore, we outlined our own evaluation requirements and techniques that we would use to assess our two solutions. In the next sections, we will try to perform a fair evaluation with respect to our own defined requirements even though we were facing timing constraints.

### 5.1 Evaluation for Solution 1

The evaluation requirement of our security assessment was that it should identify the security goals of KakaoTalk and make a clear statement about how well KakaoTalk met a particular goal. Moreover, we stated that for any goal that is not met, our assessment should clearly explain why, and preferably include a Proof of Concept (PoC) exploit.

In Section 4.3.2 we claimed that KakaoTalk’s main security goal is to protect its valuable assets from potential threats such as data theft. In our security assessment we mainly focused on the most important asset, which we believe is the user’s Personally Identifiable Information (PII). In particular, we focused on user’s private end-to-end encrypted messages as well as on user’s private information (i.e., identifiers, contact or location information, and other data). Our security analysis showed that KakaoTalk may not

necessarily protect these assets to the full extend:

**End-to-end encrypted messages:** We showed that KakaoTalk’s end-to-end encryption feature is theoretically vulnerable to MITM attacks on the operator-side (Section 4.4.1). Unfortunately, we could not come up with a PoC exploit to demonstrate this weakness in practice due to timing constraints. However, one may be able to easily set up an experiment to simulate the operator-side MITM attack that we described in Section 4.4.1:

The controlled experiment would consist of two modified KakaoTalk 5.5.5 clients and a MITM proxy application. The two clients would need to be modified in such a way that they would successfully establish a LOCO session with the MITM proxy application (see Section 4.2.3.2). For this, one would need to modify the client’s source code in order to replace Kakao’s hard-coded RSA public key with a RSA key that belongs to the proxy. The main component of the experiment would be the MITM proxy application which would mimic a legitimate LOCO messaging server in order to intercept any LOCO messaging traffic. The proxy would be connected to a legitimate LOCO messaging server in order to relay any intercepted LOCO messaging traffic to a destination. After describing the components of the experiment, a MITM attack on end-to-end encrypted messaging could work as follows:

- 1 First, the modified clients would need to be forced to negotiate a LOCO handshake with the MITM proxy application. This could be easily achieved by applying well known MITM attacks such as DNS cache poisoning attacks. The clients would establish a LOCO session with the proxy as they do not verify the authenticity of the connection endpoint.
- 2 During the LOCO handshake, the modified clients would generate a symmetric AES key that would be encrypted with the proxy’s RSA public key. As a result, the proxy would be then able to decrypt the LOCO “handshake” packet (see Section 4.2.3.1) and all further LOCO packets sent by the clients. In a next step, the proxy would re-encrypt all LOCO packets with its own symmetric AES key in order to forward them to Kakao’s legitimate LOCO messaging backend.
- 3 Being the MITM, the proxy could then attack end-to-end encrypted chats by replacing each participants public keys (see Section 4.4.1).

In addition to an operator-side MITM attack on KakaoTalk’s “Secret Chat” feature, we also found a code injection vulnerability which may be exploited to compromise an end-to-end encrypted chat conversation (see Section 4.4.2). While we explained how a possible JavaScript injection attack

could work in theory, we did not have the resources to develop a PoC that would show the attack in practice. Nevertheless, one may build a JavaScript exploit that could be simply injected into a KakaoTalk WebView that has a JavaScript to Java bridge added to it. A potential PoC exploit may be able to access the bridge to start a remote shell on the user's mobile handset. After connecting to the shell, one may drop a root exploit onto the device to escalate privileges. In a final step, one may then be able to steal end-to-end encryption keys from the handset's data storage and/or physical device memory. The code injection attack may be performed by any local or remote network attacker as KakaoTalk downloads web content via improperly secured network channels.

One weakness of our security assessment is that our end-to-end (E2E) encryption protocol analysis suffers a major limitation. As stated in Section 4.1, we reverse-engineered the protocol by analysing an older version of KakaoTalk (4.7.0) as the source code of version 5.5.5 was obfuscated by ProGuard. Even though we were able to establish an E2E encrypted chat between KakaoTalk 4.7.0 and KakaoTalk 5.5.5, we cannot eliminate the fact that two clients running on version 5.5.5 may use a different version of the protocol. In our test, KakaoTalk 5.5.5 may just used an older protocol version in order to guarantee backwards compatibility with KakaoTalk 4.7.0. For this reason, it might be the case that our detected security flaws do not apply if only KakaoTalk 5.5.5 clients participate in an E2E encrypted chat. For a rigorous protocol security assessment one would need to test several KakaoTalk versions on different mobile operating systems (Android and iOS).

***User's private information:*** We argue that another major security goal of KakaoTalk is to protect user's private information from unauthorized access. While we described numerous application exit points that leak user's private information, we did not have the time to come up with a dedicated PoC program. Furthermore, we cannot eliminate the fact that KakaoTalk may leak user's private information through other channels that we have not investigated due to limitations of our study. For instance, while we completed a rigorous network traffic analysis, we did not perform a detailed analysis of KakaoTalk's inter-process communications (IPC). Although we showed that KakaoTalk does not secure a number of its IPC interfaces, we did not have the resources to identify further possible IPC weaknesses that may lead to information leakage. However, one may detect user's private information leakage through IPC by fuzzing. For example, one could fuzz KakaoTalk

IPC interfaces by using Drozer’s Android Intent Fuzzing module<sup>1</sup>.

**Other assets:** For all remaining assets including user’s message and communication metadata, user’s money, Kakao’s reputation and other assets, we did not have the time to investigate whether KakaoTalk is protecting them in an appropriate manner or not. There are a number of other reasons why we left out certain assets:

Our two assessment checklists (see Appendix A) do not include all possible vulnerabilities that may exist in Android chat applications. This is mostly due to the fact that there are no agreed best-practices or other sources that list all possible security weaknesses.

Our threat analysis faces limitations in that we intentionally excluded topics such as risk assessment, threat prioritization, or countermeasure specification. In addition, we made a number of security assumptions which further shrank the number of possible attack vectors (see Section 4.3.1).

Other than scanning Kakao’s servers for TLS weaknesses, we did not scan or fuzz the company’s backend for vulnerabilities or check for other weak server-side controls in order to not possibly disturb the company’s services.

Even though we identified KakaoTalk’s components and software libraries that deal with the user’s money (i.e., Kakao Pay), we did not have the time to search for any possible flaws.

Regarding user’s message and communication metadata we only listed a subset of data that is stored by KakaoTalk (see Section 4.2.2). A more profound analysis may detect more such metadata that the application generates and possibly stores.

After describing the main pros and cons of our security assessment, we suggest how a comparative study could be used to identify further strengths and weaknesses of our chosen approach. The main method of a comparative study would be a rigorous review of existing literature and to compare related work on Android application security assessments with our study. From what we know from our literature review concerning Chapter 2, we argue that existing related work may be divided into two categories. First, there is related work that explicitly focuses on end-to-end encryption capable mobile

---

<sup>1</sup><https://github.com/mwrlabs/drozer-modules/tree/master/intents>

chat applications. In this category, related work either concentrates on the assessment of messaging protocols (e.g., the iMessage protocol [55], the Signal protocol [53], the OTR protocol [38], or the MTPROTO protocol [72]) or on the digital forensics analysis of mobile chat applications (e.g., WhatsApp [2] or Telegram [108]). The other main related work category that we discovered is existing literature that focuses on the security assessment of general Android applications in order to identify a certain vulnerability type. For instance, [43] concentrates on TLS issues in Android applications, whereas [16] shows common OAuth weaknesses in Android applications. Reaves et al. [101] analyzed several mobile Android banking applications and used a similar methodology approach as our work.

Given these two related work categories, a profound comparative study would then compare our work to these different security assessments with respect to used methodologies and tools, chosen mobile risks, identified vulnerabilities, or published PoC exploits.

In summary, this was what we could and could not achieve for solution number 1:

1. We identified several security goals for KakaoTalk (Section 4.3).
2. We analyzed to what extent KakaoTalk met the security goals (Section 4.4), although with some caveats as we described above.
3. We performed a brief informal comparison of our analysis with other work (above in this section) but a more complete comparison is feasible, given enough time and resources.

## 5.2 Evaluation for Solution 2

For solution number two we planned to create a higher-level report that would represent our main findings in a clear format. The goal of this report would have been to provide an easy-understandable way for users to inform themselves about the strengths and weaknesses of KakaoTalk's end-to-end encryption system. With that knowledge users would be able to make an informed decision whether they want to trust the system or not. Again due to timing constraints, we were unfortunately not able to compile such a report. However, we will publish a press release of our main findings in a concise format on <https://citizenlab.org/> in August 2016. Since this will be after the submission of this Master thesis, we will not be able to verify whether the press release would make an impact on user's behaviour or not. However, in the following we propose a user study that may be used

to validate whether the press release would help users to make an informed decision about KakaoTalk’s end-to-end encryption feature or not.

The user study would be set up as a controlled experiment in which we would invite KakaoTalk users who regularly use its “Secret Chat” opt-in feature. We would then divide participants into two groups: One group would be given the press release prior the experiments whereas the other group would not receive the report in advance. After the experiments, which would consists of a number of specific user tasks (e.g., chatting, comparing public key fingerprints, reactions to security warnings, and other tasks), we would analyse and compare the different behaviour of these two groups. If we cannot get enough participants to set up a controlled study, we would use other forms of usability evaluation such as cognitive walkthrough with an usability expert or qualitative feedback surveys.

In summary, this was what we could and could not achieve for solution number 2:

1. We will publish a partial report on <https://citizenlab.org/> in August 2016 to make our technical findings available to a wider non-technical audience.
2. We did *not* perform a usability evaluation of the partial report, but again a user study is feasible, given enough time and resources (as described above).

## Chapter 6

# Discussion

In this chapter, we briefly discuss the impact of our results (Section 6.1) and reflect on our evaluation techniques (Section 6.2). We conclude by suggesting interesting projects for future work in Section 6.3.

### 6.1 Significance of Our Results

In this section we discuss the broader message of our technical findings. First, we speculate whether Kakao really intends to protect user’s privacy and security or if the company’s numerous public statements suit for marketing purposes only (Section 6.1.1). Second, we explain what our findings in respect to end-to-end encrypted messaging mean in general (Section 6.1.2). Lastly, we try to come up with a number of recommendations to give high-risk users a chance to use KakaoTalk’s “Secret Chat” feature most securely (Section 6.1.3).

#### 6.1.1 Terms of Service Claims Versus Real-World Technical Findings

What our comparison from Section 4.5 shows are two interesting facts: On the one hand, Kakao invested enormous efforts in improving user’s security and privacy. On the other hand, KakaoTalk suffers a number of software vulnerabilities that affect user’s security and privacy in a negative way. This may raise the question whether Kakao is really serious about user’s security and privacy despite making these claims. Especially, if one looks at KakaoTalk’s business model which is still mainly based on advertising: To show the best possible advertising to a user, KakaoTalk may track user behaviour by using up to six different cookies. What is more, since Kakao has a new

CEO, there may be concerns that the company may abandon some of its strict security and privacy guidelines (see Section 3.2.3.2). So yes, Kakao is concerned about user's security and privacy, but there is also proof that Kakao fails to keep some of its promises. What our comparison findings mean for the user is that a company's public security and privacy claims should not be trusted as long its software is closed-source and has not been verified independently.

### 6.1.2 End-to-end Encryption Alone is not Sufficient

KakaoTalk's "Secret Chat" feature serves as an example that violations of security and cryptography best practices result in practical attacks against end-to-end encryption in mobile chat applications. Since messaging applications have a tremendous attack surface, there is much potential of what can possibly go wrong. This means, that end-to-end encryption does not help much if the overall system security of the messaging application is vulnerable. What we have also seen in this work is how important the aspect of usable security really is. Users are often the weakest link in a software system. If KakaoTalk users do not understand security warning messages or do not know how to compare each other's public key fingerprint, end-to-end encryption does not provide security at all. Finally, sometimes the mere suspicion of sending or receiving an end-to-end encrypted message can be harmful to an user. Therefore, even though the message content may not be decryptable by an attacker, the metadata of a particular conversation can be enough to identify individuals. All of the popular secure instant messaging systems that exist today do not protect user's metadata. However, some of them can be used over anonymization networks such as Tor to provide some level of metadata protection.

Another observation is that KakaoTalk uses many different techniques for the same functionality (Section 4.2.3). For instance, the application uses different encryption modes (Cipher Feedback and Counter Mode) in different situations. However, KakaoTalk fails to use authenticated encryption modes such as Galois/Counter Mode (GCM) in places where it would have made perfect sense. KakaoTalk's design also seems to have other artifacts that suggest that the application design choices were made over a long period of time, possibly by multiple designers. While this is hard to avoid in large, long-lived software projects, it also suggest a lack of a coherent security architecture done by one or a small group of security architects.

### 6.1.3 Recommendations for High-risk Users

In this section we try to come up with a list of user recommendations on how to use KakaoTalk most securely. Given a high-risk user who has no chance to use a different more secure messenger such as Signal, these are general guidelines on how to have a rather secure end-to-end encrypted conversation (we use a language that is targeted to the user):

- Try to use KakaoTalk via the Tor network (e.g., via the Orbot Android application).
- Do not accept security warnings. If KakaoTalk shows a warning message – no matter which – stop using it. Try to switch your communications to another secure (out-of-band) channel. Verify whether you receive the same warning message when using a different WiFi or cellular network.
- Create a new “Secret Chat” chat room each time you start a new end-to-end encrypted conversation. Do not reuse previously created “Secret Chat” chat rooms.
- Always compare the other party’s public key fingerprint over a secure out-of-band channel.
- Do not use KakaoTalk for any other reason except for “Secret Chat”.
  - Do not use the regular or open chat rooms.
  - Do not use “Plus Friend” or the “Item Store”.
  - Do not tap on the “Help Center” and the “News feed”.

## 6.2 Reflection on Our System Evaluation

Evaluating our work proved to be difficult for several reasons. First, it was hard to come up with an evaluation technique for a subject that is not tangible. We were not proposing a real technical solution or implementation that could be easily evaluated. But instead, we “just” performed a security assessment and terms of service analysis of an end-to-end encryption protocol and chat application.

Second, evaluation was hard due to the strict timing constraints of our project. Reflecting on our time management, we spent too many hours on topics that may have required less detailed work and as such less time. For instance, we investigated enormous resources in researching South Korea’s

political and historical context. Also, we might spent too much time on reviewing Kakao’s terms of services, privacy policies, and various other sources in order to find public statements with regards to security and privacy. In retrospect, this led to the fact that we did not have enough timing resources for our technical analysis as we had planned initially. Regarding our technical analysis, the most time consuming factor was to identify common vulnerabilities in Android applications and to set up our development and tool chain. Since there are various existing tools on Android application security as well as numerous common weaknesses in Android applications, finding out what tools and vulnerabilities are the most important, took the most efforts. However, once we had agreed on a common set of Android weaknesses and had compiled our assessment checklists, the actual vulnerability analysis of KakaoTalk took less time as expected. The most difficult challenges during our reverse-engineering assessment of KakaoTalk’s end-to-end encryption protocol were code obfuscation and code recovery failures.

For these reasons, we were not able to evaluate our work to the full extent. However, what we did was to describe how an ideal evaluation of our work could look like (see Section 5).

## 6.3 Future Work

The fact that we had a limited time budget and that KakaoTalk provides a vast amount of possible attack vectors leads to a number of interesting future work projects:

**Server-side key generator:** In Section 4.2.3.2 we showed that Kakao may generate OAuth access tokens with weak randomness. If one receives Kakao’s permission, one could try to feed `https://ac-talk.kakao.com/android/account/signup.json` with different input to find out more about how the blackbox key-generator works.

**Group E2E encrypted chat:** In this thesis we only looked at KakaoTalk’s one-on-one E2E encryption feature. A future work project could examine how KakaoTalk uses the protocol for group E2E encrypted chat rooms.

**IPC analysis:** This work mainly focused on IP network communications rather than on KakaoTalk’s on-device inter-process communications (IPC). It would be interesting to see whether a third-party application with user-permissions is able to interact with KakaoTalk in such a way that it could sniff or spoof IPC messages. “On-device attacks”

are plausible since KakaoTalk allows to install customized themes as separate APK files. These APKs may be malicious and may exploit improperly secured IPC interfaces.

**Kakao Pay:** Kakao Pay would be an interesting component to take a look at since it uses a number of custom cryptography libraries with many of them not being obfuscated by ProGuard. We found that some of Kakao Pay’s third-party libraries may have an update service included and may have the permissions to install APKs. In addition, one could try to improve or continue to work on our suggested library injection attack (see Section 4.4.2). The native `libNSaferJNI.so` and `liba3030.so` would be suitable candidates as these libraries are loaded by Kakao Pay via an explicit path argument at runtime.

**Enumeration attacks:** Kim et al. proposed an enumeration attack in 2015 [80]. One could verify if the attack is still possible by developing a simple curl script that tries to crawl user data via sending POST requests to URLs such as `https://auth.kakao.com/kakao_accounts/find_account_by_phone.json` or `https://katalink.kakao.com/android/friends/add_by_phonenumber.json`.

**Replay attacks:** As explained in Section 4.4.2, we did not have the time to perform a detailed replay attack analysis on KakaoTalk’s chat protocol. A future work project may try to replay, drop, mangle, mirror, or reorder messaging packets in order to find out whether KakaoTalk’s client/server components handle tampered message flows correctly.

**Push notifications:** A future work project could work on the security of KakaoTalk’s push notifications. This may be another attack vector to possibly obtain the plaintext of an end-to-end encrypted chat message.

**JavaScript exploits:** We outlined in Section 4.4.2 that we did not come up with a PoC exploit that shows our identified code injection vulnerabilities in practice. There would be a need for a JavaScript exploit that tries to read a file from KakaoTalk’s private data directory by bypassing a WebView’s same origin policy. Another exploit could try to access the JavaScript to Java bridge to open up a remote accessible shell on older Android versions < 4.2.

**Testing:** In order to further verify the impact of our detected weaknesses, one would need to perform more testing. Profound testing would include testing our vulnerabilities on the latest version of KakaoTalk

(4.7.1 as of 04.07.2016), on different Android version, as well as on other mobile operating systems such as iOS.

## 6.4 Responsible Disclosure and Notification

In August 2016 we delivered a summary of the issues identified in this Master thesis to Kakao, and indicated that we would like to publish our findings no sooner than 45 days after having sent the notification. This process is in line with international standards on vulnerability disclosure<sup>1</sup>.

After the 45-day deadline we will announce our findings in a partial report on <https://citizenlab.org/>. The report will also contain any correspondence with Kakao related to the responsible disclosure process.

---

<sup>1</sup><https://www.cert.org/vulnerability-analysis/vul-disclosure.cfm>

## Chapter 7

# Conclusion

This Master thesis performed a detailed terms of service and technical analysis of KakaoTalk which is a mobile messaging application from South Korea. We explained how a company's (Kakao) terms of service policy has changed over time and showed how political and social changes in a country may affect a corporation's policies. To the best of our knowledge, this type of context analysis for a mobile end-to-end (E2E) encryption application has not been done before. However, this is crucial as political or social changes may affect the user's security and privacy in a positive or negative way. In terms of KakaoTalk, political events have improved user's security and privacy. Moreover, in our terms of service analysis we listed several statements that were made by Kakao with regard to security and privacy. We grouped Kakao's statements into security and privacy categories and derived a number of research questions from them. These questions were driving our technical analysis of the KakaoTalk application. The analysis consisted of an in-depth security assessment in which we first identified common Android and end-to-end encryption vulnerabilities by compiling two distinct security assessment checklists. Further, we described and reverse-engineered parts of KakaoTalk's messaging system including but not limited to KakaoTalk's end-to-end encryption protocol. After describing the system we performed a threat analysis in order to map out Kakao's security goals, possible threat agents, as well as the system's attack surface. We continued with the main part of our technical analysis, which was to identify vulnerabilities in KakaoTalk's end-to-end encryption protocol as well as to find weaknesses in KakaoTalk's Android version. Following we list the main findings of our end-to-end encryption protocol assessment:

- 1 Within two months, Kakao developed an E2E encryption protocol which is built on top of the company's existing non-E2E chat protocol. All

communications are relayed through a messaging server which unpacks the packets, creates new ones, and forwards them to the destination. The server also stores packets in case the recipient is offline.

- 2 Kakao uses a central public-key directory server which makes the end-to-end messaging system prone to man-in-the-middle (MITM) attacks on the operator-site. If users do not compare their public key fingerprints, this attack may remain undetected. Since Kakao has the ability to delete E2E encrypted chat rooms on the server-side, those MITM attacks can also be performed *after* two clients have agreed on a shared secret key. The user would just need to create a new chat room and the attacker would be then able to substitute a user's key during the initial secret key exchange. Most of the secure messaging systems that are used today run a public-key directory server, e.g., WhatsApp, Signal, or iMessage [55]. While this naive attack may seem obvious, KakaoTalk end users should know about the strength and weaknesses of the particular design that Kakao has chosen. With that knowledge they can make an informed decision whether they want to trust Kakao or not.
- 3 The actual chat message is the only data field in the protocol that is protected by a Message Authentication Code (MAC). However, KakaoTalk uses the concept of MAC-then-encrypt which means that the ciphertext remains unprotected. All other data fields in the packet do not have integrity protection.
- 4 There is no messaging-server authentication. A rogue or malicious server or a local/global network MITM attacker may tamper with the messaging packets at will. In addition, the KakaoTalk application responds to a number of commands sent by the messaging server. A MITM could do the same without being detected by the client.
- 5 Replay/reflection attacks are possible even though we have not tested this much. We were able to replay packets and change the order of packets, but in some cases, which we could not reproduce because of timing constraints, the client or server auto-corrects this misbehaviour.

In addition to our protocol analysis, we also found a number of application vulnerabilities in KakaoTalk's Android version:

- 1 For some domains KakaoTalk accepts self-signed MITM certificates. While throwing `CertPathValidatorException` messages in the background,

the user is presented with just a simple warning dialog mentioning “There are problems with the security certificate for this site”. If the user accepts the dialog the application continues working and a MITM can read the traffic. In addition, some of KakaoTalk’s TLS endpoints are vulnerable to POODLE, DROWN, and padding oracle attacks.

- 2 KakaoTalk is a hybrid application which means that it makes heavily usage of the `WebView` class to display web content that is being downloaded through HTTPS and sometimes even through HTTP connections. Most of the `WebViews` have JavaScript (JS) enabled and allow file system access as KakaoTalk is caching web content on disk. Since web content is downloaded through unauthenticated channels and TLS connections that can be compromised with a self-signed certificate, a MITM may be able to inject and executable arbitrary JS code. We did not have time to test this in depth, but we could inject a simple `alert(document.cookie);` statement that was executed on the client-side. Also, if an attacker manages to bypass a `WebView`’s same-origin policy she might be able to steal the application’s RSA and ED25519 key pairs that are stored in the `TalkKeyStore.preferences.xml`. file within the application’s private data directory.
- 3 For some `WebViews` KakaoTalk uses the `addJavascriptInterface()` public method which implements a Java to JavaScript bridge. This means that JavaScript code is able to interact with the native application by calling Java methods. This “feature” can possibly be turned into a remote code execution attack if an attacker uses the bridge to access the `getClass()` method that is inherited from the `Object` class. The attack works on Android versions older than 4.2 out of the box. Newer versions are typically not affected, but Java methods can be still called if they have the `@JavascriptInterface` annotation. Nevertheless, 10.8 percent of all Android devices still run versions older than 4.2<sup>1</sup>. Again, because of KakaoTalk’s TLS certificate validation issues this might be exploitable by a MITM. What might be worse is that KakaoTalk provides a marketing channel called “Plus Friend” which can be used by brands and artists for advertising their products. Basically, brands set up their own web site that the user can access through the “`PlusFriendListWebActivity`”. These websites also have access to KakaoTalk’s JavaScript bridge. If Kakao does not verify the integrity of these sites before they are published on “Plus Friend”, any third-party may be able to execute arbitrary code on Android versions ‘4.2.

---

<sup>1</sup><https://developer.android.com/about/dashboards/index.html>

- 4 Finally, there are a number of other minor security and privacy issues that we listed in this thesis, e.g., user tracking, information leakage, access control issues, and various other minor problems.

Summarizing, our findings indicate that KakaoTalk is violating some of Kakao’s statements made in the company’s terms of service and privacy policies. This way, we could accomplish the main goal of this Master thesis which was to verify whether some of Kakao’s marketing claims with respect to user’s security and privacy are true or false. We think that it is fair to say that one of Kakao’s most important claim “[...] Kakao’s server is unable to decrypt the encryption [of KakaoTalk’s Secret Chat mode] [...]” is – when looking at our findings – actually a false statement. So yes there is end-to-end encryption, but Kakao’s public key server infrastructure is not trustworthy. Kakao can decrypt end-to-end encrypted messages if they want or are ordered to by the local government. All in all, our findings show that various issues in a product can add up to the potential for serious attack vectors despite there being multiple layers of security.

# Bibliography

- [1] Jagdish Prasad Achara, Mathieu Cunche, Vincent Roca, and Aurélien Francillon. *WifiLeaks: Underestimated Privacy Implications of the ACCESS\_WIFI\_STATE Android Permission*. Research Report RR-8539. A short version has been accepted for publication in: 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC'14) Oxford, United Kingdom, July 23rd – 25th 2014. Inria, May 2014, p. 21. URL: <https://hal.inria.fr/hal-00994926> (cit. on p. 114).
- [2] Cosimo Anglano. “Forensic analysis of WhatsApp Messenger on Android smartphones”. In: *Digital Investigation* 11.3 (2014), pp. 201–213 (cit. on p. 123).
- [3] Nitay Artenstein and Idan Revivo. “Man in the binder: He who controls ipc, controls the droid”. In: *Europe BlackHat Conf.* 2014 (cit. on p. 26).
- [4] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. “DROWN: Breaking TLS using SSLv2”. In: () (cit. on p. 107).
- [5] Gayoon Baek and Youngsoug Chan. *NGO Oral Statement to the UN Human Rights Committee*. <http://www.peoplepower21.org/English/1369110>. Accessed: 2016-04-24. Oct. 2015 (cit. on p. 34).
- [6] Matthew Barakat. *Jeffrey Sterling, ex-CIA officer, convicted of leaking secrets to reporter*. <http://www.washingtontimes.com/news/2015/jan/26/deliberation-to-reach-third-day-in-cia-leak-case/>. Accessed: 2015-03-11. Jan. 2015 (cit. on p. 20).
- [7] Mihir Bellare and Phillip Rogaway. “Optimal asymmetric encryption”. In: *Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1994, pp. 92–111 (cit. on p. 156).

- [8] David Belson. “Akamai state of the Internet report, q3 2015”. In: 8.3 (Dec. 2015) (cit. on p. 33).
- [9] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. “High-speed high-security signatures”. In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89 (cit. on p. 83).
- [10] Nikita Borisov, Ian Goldberg, and Eric Brewer. “Off-the-record communication, or, why not to use PGP”. In: *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM. 2004, pp. 77–84 (cit. on pp. 18, 19).
- [11] Cristian Bravo-Lillo, Saranga Komanduri, Lorrie Faith Cranor, Robert W Reeder, Manya Sleeper, Julie Downs, and Stuart Schechter. “Your attention please: designing security-decision UIs to make genuine risks harder to ignore”. In: *Proceedings of the Ninth Symposium on Usable Privacy and Security*. ACM. 2013, p. 6 (cit. on p. 105).
- [12] BusinessKorea. *KakaoTalk Confirms Noncompliance with Wiretapping Warrants to Protect Personal Information*. <http://www.businesskorea.co.kr/english/news/politics/6846-business-law-kakaotalk-confirms-noncompliance-wiretapping-warrants-protect>. Accessed: 2016-04-24. Oct. 2014 (cit. on p. 39).
- [13] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. “Deniable encryption”. In: *Advances in Cryptology—CRYPTO’97*. Springer, 1997, pp. 90–104 (cit. on p. 19).
- [14] Jonathan Carter and Milan Singh Thakur. *OWASP Mobile Security Project*. [https://www.owasp.org/index.php/OWASP\\_Mobile\\_Security\\_Project](https://www.owasp.org/index.php/OWASP_Mobile_Security_Project). Accessed: 2016-06-25. May 2016 (cit. on pp. 57, 148).
- [15] PSPD Law Center and Open Net Korea. *Submission to the UN Human Rights Committee 115th Session, 19 October 2015 – 6 November 2015 Republic of Korea*. <http://opennetkorea.org/en/wp/wp-content/uploads/2016/03/warrantless-mass-surveillance.pdf>. Accessed: 2016-04-28. Oct. 2015 (cit. on p. 36).
- [16] Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. “Oauth demystified for mobile application developers”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 892–903 (cit. on p. 123).

- [17] Hewlett Packard Enterprise Development Company. *HPE Security Fortify Taxonomy: Software Security Errors*. <http://www.hpenterprisesecurity.com/vulncat/en/vulncat/index.html>. Accessed: 2016-06-25. 2016 (cit. on pp. 57, 148).
- [18] Jonathan Corbet. *Random numbers for embedded devices*. <https://lwn.net/Articles/507115/>. Accessed: 2016-07-02. July 2012 (cit. on p. 155).
- [19] Kakao Corp. *Daum Kakao Releases Transparency Report*. [http://www.kakaocorp.com/en/pr/pressRelease\\_view?page=3&group=1&idx=8145](http://www.kakaocorp.com/en/pr/pressRelease_view?page=3&group=1&idx=8145). Accessed: 2016-04-24. Jan. 2015 (cit. on p. 40).
- [20] Kakao Corp. *Frequently Asked Questions*. <http://privacy.kakaocorp.com/en/faq/protect/page1>. Accessed: 2016-04-28. 2016 (cit. on pp. 40, 47–50).
- [21] Kakao Corp. *Help*. <http://www.kakao.com/helps?locale=en&service=8>. Accessed: 2016-07-07. 2016 (cit. on p. 45).
- [22] Kakao Corp. *Kakao Comprehensive Terms and Conditions*. <http://www.kakao.com/policy/terms>. Accessed: 2016-06-22. Mar. 2016 (cit. on p. 53).
- [23] Kakao Corp. *Kakao Operation Policy*. <http://www.kakao.com/policy/oppolicy>. Accessed: 2016-07-07. 2016 (cit. on p. 52).
- [24] Kakao Corp. *KAKAO Privacy Policy[1]*. <http://www.kakao.com/policy/privacy>. Accessed: 2016-04-28. Mar. 2016 (cit. on pp. 47–51).
- [25] Kakao Corp. *Kakao Terms of Service*. <http://www.kakao.com/policy/terms>. Accessed: 2016-06-22. Mar. 2016 (cit. on pp. 47, 50–52).
- [26] Kakao Corp. *KakaoTalk Ranks No. 1 in App Sessions Worldwide!* [http://www.kakaocorp.com/en/pr/pressRelease\\_view?page=2&group=1&idx=8255](http://www.kakaocorp.com/en/pr/pressRelease_view?page=2&group=1&idx=8255). Accessed: 2016-06-27. May 2015 (cit. on pp. 12, 30).
- [27] Kakao Corp. *Notice*. <http://www.kakao.com/notices>. Accessed: 2016-06-22 (cit. on pp. 40, 41).
- [28] Kakao Corp. *Personal Information Lifecycle*. <http://privacy.kakaocorp.com/en/transparence/lifeCycle>. Accessed: 2016-07-07. 2016 (cit. on pp. 47, 49, 51).
- [29] Kakao Corp. *Privacy Philosophy*. <http://privacy.kakaocorp.com/en/philosophy/kakao>. Accessed: 2016-06-22 (cit. on pp. 42, 45).

- [30] Kakao Corp. *Protection of Users' Rights*. <http://privacy.kakaocorp.com/en/transparency/report/present>. Accessed: 2016-07-07. 2016 (cit. on p. 45).
- [31] Kakao Corp. *Secret Chat Mode and Decline Invites Feature Now Available on KakaoTalk*. [http://www.kakaocorp.com/en/pr/pressRelease\\_view?page=1&group=1&idx=8103](http://www.kakaocorp.com/en/pr/pressRelease_view?page=1&group=1&idx=8103). Accessed: 2016-04-28. Dec. 2014 (cit. on pp. 39, 44).
- [32] Kakao Corp. *Technical Measures*. <http://privacy.kakaocorp.com/en/protection/tech>. Accessed: 2016-07-07. 2016 (cit. on pp. 13, 45, 47).
- [33] Kakao Corp. *Terms and Conditions of Location-Based Services*. <http://www.kakao.com/policy/location>. Accessed: 2016-07-07. Mar. 2016 (cit. on p. 46).
- [34] Kakao Corp. *Transparency Report*. Tech. rep. <http://privacy.kakaocorp.com/en/transparency/report/request>. Kakao Corp., Aug. 2015 (cit. on pp. 12, 36, 37, 39, 40, 50, 54, 117).
- [35] NAVER Corp. *NAVER Privacy Report*. Tech. rep. [https://nid.naver.com/inc/user/pdf/NAVER\\_PrivacyReport\\_2015.pdf](https://nid.naver.com/inc/user/pdf/NAVER_PrivacyReport_2015.pdf). NAVER Corp., 2015 (cit. on p. 36).
- [36] Scott E Coull and Kevin P Dyer. “Traffic Analysis of Encrypted Messaging Services: Apple iMessage and Beyond”. In: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), pp. 5–11 (cit. on p. 17).
- [37] Jedidiah Crandall. Personal Communication. July 6, 2016 (cit. on pp. 57, 154).
- [38] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. “Secure off-the-record messaging”. In: *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM. 2005, pp. 81–89 (cit. on p. 123).
- [39] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. “Composability and on-line deniability of authentication”. In: *Theory of Cryptography Conference*. Springer. 2009, pp. 146–162 (cit. on p. 19).
- [40] Danny Dolev, Cynthia Dwork, and Moni Naor. “Nonmalleable cryptography”. In: *SIAM review* 45.4 (2003), pp. 727–784 (cit. on p. 102).

- [41] The Korea Economic. *Six Out of Ten Korean Internet Users Use IM Services Everyday*. <http://english.hankyung.com/news/apps/news.view?popup=0&nid=0&c1=&newscate=1&nkey=201510071123391>. Accessed: 2016-06-27. Oct. 2015 (cit. on p. 30).
- [42] Nikolay Elenkov. *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. No Starch Press, 2014 (cit. on p. 21).
- [43] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. “Why Eve and Mallory love Android: An analysis of Android SSL (in) security”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 50–61 (cit. on pp. 62, 123).
- [44] Geoffrey Fattig. *South Korea's Anti-Terror Law Part of a Worrisome Trend*. <http://thediplomat.com/2016/03/south-koreas-anti-terror-law-part-of-a-worrisome-trend/>. Accessed: 2016-06-22. Mar. 2016 (cit. on p. 42).
- [45] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011 (cit. on pp. 155–158).
- [46] Electronic Frontier Foundation. *20131014-Wapo-Content Acquisition Optimization2*. <https://www.eff.org/document/2013-10-14-wapo-content-acquisition-optimization2>. Accessed: 2016-04-06. 2013 (cit. on p. 16).
- [47] Electronic Frontier Foundation. *In South Korea, the Only Thing Worse Than Online Censorship is Secret Online Censorship*. <https://www.eff.org/deeplinks/2011/08/south-korea-only-thing-worse-online-censorship>. Accessed: 2016-06-22. Sept. 2011 (cit. on p. 35).
- [48] Electronic Frontier Foundation. *Secure Messaging Scorecard*. <https://www.eff.org/secure-messaging-scorecard>. Accessed: 2016-04-07. Apr. 2016 (cit. on p. 17).
- [49] Free Software Foundation. *GNU/consensus/Secure Messaging Scoreboard*. [https://libreplanet.org/wiki/GNU/consensus/Secure\\_Messaging\\_Scoreboard](https://libreplanet.org/wiki/GNU/consensus/Secure_Messaging_Scoreboard). Accessed: 2016-04-07. Oct. 2015 (cit. on p. 17).
- [50] Jay Freeman. *Android Bug Superior to Master Key*. <http://www.saurik.com/id/18>. Accessed: 2016-04-11 (cit. on p. 26).
- [51] Jay Freeman. *Exploit (& Fix) Android “Master Key”*. <http://www.saurik.com/id/17>. Accessed: 2016-04-11 (cit. on p. 26).

- [52] Jay Freeman. *Yet Another Android Master Key Bug*. <http://www.saurik.com/id/19>. Accessed: 2016-04-11 (cit. on p. 26).
- [53] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Thorsten Holz, et al. “How Secure is TextSecure?” In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 457–472 (cit. on pp. 11, 123).
- [54] Amit Fulay and Yariv Adan. *Saying hello to Allo and Duo: new apps for smart messaging and video calling*. <https://googleblog.blogspot.com/2016/05/allo-duo-apps-messaging-video.html>. Accessed: 2016-07-07. May 2016 (cit. on p. 11).
- [55] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. “Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage”. In: () (cit. on pp. 123, 132).
- [56] Graham Greenleaf and Whon-il Park. “Korea’s new Act: Asia’s toughest data privacy law”. In: *Privacy Laws & Business International Report* 117 (2012), pp. 1–6 (cit. on pp. 13, 33).
- [57] Oscar M Guillen, Dawin Schmidt, and Georg Sigl. “Practical evaluation of code injection in encrypted firmware updates”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 325–330 (cit. on p. 103).
- [58] Christoph G Günther. “An identity-based key-exchange protocol”. In: *Advances in Cryptology–Eurocrypt’89*. Springer. 1990, pp. 29–37 (cit. on p. 18).
- [59] Junghee Han and Okjoo Cho. “Platform business Eco-model evolution: case study on KakaoTalk in Korea”. In: *Journal of Open Innovation: Technology, Market, and Complexity* 1.1 (2015), pp. 1–14 (cit. on p. 30).
- [60] Chico Harlan. *In S. Korea, a shrinking space for speech*. [https://www.washingtonpost.com/world/asia\\_pacific/in-s-korea-a-shrinking-space-for-speech/2011/12/21/gIQAmAHgBP\\_story.html](https://www.washingtonpost.com/world/asia_pacific/in-s-korea-a-shrinking-space-for-speech/2011/12/21/gIQAmAHgBP_story.html). Accessed: 2016-04-24. Dec. 2011 (cit. on p. 35).
- [61] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. “Threat modeling-uncover security design flaws using the stride approach”. In: *MSDN Magazine-Louisville* (2006), pp. 68–75 (cit. on p. 90).
- [62] Neal Hindocha and Eric Chien. “Malicious threats and vulnerabilities in instant messaging”. In: *Virus Bulletin Conference, vb2003*. 2003 (cit. on p. 95).

- [63] Taylor Hornby. *Telegram's Cryptanalysis Contest*. <http://www.cryptofails.com/post/70546720222/telegrams-cryptanalysis-contest>. Accessed: 2016-07-02. Dec. 2013 (cit. on p. 157).
- [64] Freedom House. *South Korea — Country report — Freedom in the World — 2015*. <https://freedomhouse.org/report/freedom-world/2015/south-korea>. Accessed: 2016-04-24. 2016 (cit. on p. 39).
- [65] Freedom House. *South Korea — Country report — Freedom on the Net — 2015*. <https://freedomhouse.org/report/freedom-net/2015/south-korea>. Accessed: 2016-04-24. 2016 (cit. on pp. 34, 39).
- [66] Peng Hwa Ang. *How Countries Are Regulating Internet Content*. [http://www.isoc.org/INET97/proceedings/B1/B1\\_3.HTM#s9](http://www.isoc.org/INET97/proceedings/B1/B1_3.HTM#s9). Accessed: 2016-04-24 (cit. on p. 34).
- [67] Philip Iglauer. *Kakao data handover ends year-long privacy dispute with South Korean government*. <http://www.zdnet.com/article/kakao-data-handover-ends-year-long-privacy-dispute-with-south-korean-government/>. Accessed: 2016-04-28. Oct. 2015 (cit. on p. 41).
- [68] Google Inc. *Transparency Report*. <https://www.google.com/transparencyreport/userdatarequests/>. Accessed: 2016-04-06. 2014 (cit. on p. 16).
- [69] Amnesty International. “South Korea: Amnesty International’s Submission to the UN Human Rights Committee, 115th Session (19 October – 6 November 2015)”. In: (Oct. 2015) (cit. on p. 34).
- [70] Kim Jae-seob. “KakaoTalk managers present at prosecutors’ meeting on countering defamation of the president” (in Korean). <http://www.hani.co.kr/arti/economy/it/657974.html>. Accessed: 2016-04-24. Oct. 2014 (cit. on p. 38).
- [71] Kim Jae-seok. *Kakao worried about government agencies response after info release*. [http://english.hani.co.kr/arti/english\\_edition/e\\_national/659194.html](http://english.hani.co.kr/arti/english_edition/e_national/659194.html). Accessed: 2016-06-22. Oct. 2014 (cit. on p. 40).
- [72] Jakob Jakobsen and Claudio Orlandi. “A practical cryptanalysis of the Telegram messaging protocol”. PhD thesis. Master Thesis, Aarhus University (Available on request), 2015 (cit. on p. 123).
- [73] Xing Jin, Tongbo Luo, Derek G Tsui, and Wenliang Du. “Code injection attacks on HTML5-based mobile apps”. In: *arXiv preprint arXiv:1410.7756* (2014) (cit. on p. 109).

- [74] Kim Jin-cheol. *To avoid government snooping, companies going into “server exile”*. [http://english.hani.co.kr/arti/english\\_edition/e\\_national/719205.html](http://english.hani.co.kr/arti/english_edition/e_national/719205.html). Accessed: 2016-06-22. Nov. 2015 (cit. on p. 41).
- [75] Sam Judah and Thom Poole. *Why South Koreans are fleeing the country’s biggest social network*. <http://www.bbc.com/news/blogs-trending-29555331>. Accessed: 2016-04-24. Oct. 2014 (cit. on p. 39).
- [76] KakaoTalk. *We recently hit 200 million users! Thanks to all our awesome users, we love you!* <https://twitter.com/kakaotalk/status/672327098971742208>. Accessed: 2016-06-27. Dec. 2015 (cit. on pp. 11, 30).
- [77] *KakaoTalk chat app boss quits after child porn row*. <http://www.bbc.co.uk/news/technology-34791020>. Accessed: 2016-04-28. Nov. 2015 (cit. on pp. 42, 52).
- [78] Michael J Kalsher and Kevin J Williams. “Behavioral compliance: Theory, methodology, and results”. In: *Handbook of warnings* (2006), pp. 313–329 (cit. on p. 105).
- [79] Yun-ji Kang. “*Hide your RRN away! Ban on online collection of user RRNs*” (in Korean). <http://reporter.korea.kr/newsView.do?nid=148755878>. Accessed: 2016-04-24. Feb. 2013 (cit. on p. 38).
- [80] Eunhyun Kim, Kyungwon Park, Hyoungshick Kim, and Jaeseung Song. “Design and analysis of enumeration attacks on finding friends with phone numbers: A case study with KakaoTalk”. In: *computers & security* (2015) (cit. on p. 129).
- [81] Alex Klyubin. *Some SecureRandom Thoughts*. <http://android-developers.blogspot.ca/2013/08/some-securerandom-thoughts.html>. Accessed: 2016-07-02. Aug. 2013 (cit. on p. 155).
- [82] Jeffrey Knockel. Personal Communication. July 7, 2016 (cit. on pp. 57, 148).
- [83] Se-Woong Koo. *South Korea’s Invasion of Privacy*. <http://www.nytimes.com/2015/04/03/opinion/south-koreas-invasion-of-privacy.html>. Accessed: 2016-04-24. Apr. 2015 (cit. on p. 36).
- [84] Frank La Rue. *Full text of the press statement delivered by the UN Special Rapporteur on the promotion and protection of the right to freedom of opinion and expression, Mr. Frank La Rue, after the conclusion of his visit to the Republic of Korea*. Tech. rep. <http://www2.ohchr.org/english/issues/opinion/docs/ROK-Presstatement17052010>.

- pdf. UN Special Rapporteur on the promotion, protection of the right to freedom of opinion, and expression, May 2010 (cit. on pp. 34, 43).
- [85] Adam Langley. *Enhancing digital certificate security*. <https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html>. Accessed: 2016-04-08. Jan. 2013 (cit. on p. 16).
- [86] Se Young Lee and Sohee Kim. *South Korea tries to ease cyber surveillance fears*. <http://www.reuters.com/article/us-southkorea-cybersecurity-idUSKCN0I514A20141016>. Accessed: 2016-07-07. Oct. 2014 (cit. on p. 52).
- [87] Matthew Keys Live. *Former NSA boss: "We kill people based on meta-data"*. <https://www.youtube.com/watch?v=UdQiz0Vavmc>. Accessed: 2016-07-24. YouTube, May 2014 (cit. on p. 20).
- [88] Mohammad Mannan and Paul C van Oorschot. "On instant messaging worms, analysis and countermeasures". In: *Proceedings of the 2005 ACM workshop on Rapid malware*. ACM. 2005, pp. 2–11 (cit. on p. 95).
- [89] Mohammad Mannan and Paul C van Oorschot. "Secure public instant messaging: A survey". In: *Proceedings of Privacy, Security and Trust* (2004) (cit. on p. 95).
- [90] Moxie Marlinspike. *Facebook Messenger deploys Signal Protocol for end to end encryption*. <https://whispersystems.org/blog/facebook-messenger/>. Accessed: 2016-07-24. July 2016 (cit. on p. 17).
- [91] Moxie Marlinspike. *WhatsApp's Signal Protocol integration is now complete*. <https://whispersystems.org/blog/whatsapp-complete/>. Accessed: 2016-07-07. Apr. 2016 (cit. on p. 11).
- [92] mayhem. *The Cerberus ELF Interface*. <http://phrack.org/issues/61/8.html>. Accessed: 2016-06-25. Aug. 2013 (cit. on p. 111).
- [93] Daniel Medianero. *Open Android Security Assessment Methodology*. <http://oasam.org/en>. Accessed: 2016-06-25. 2016 (cit. on pp. 57, 148).
- [94] Peter Micek. *South Korean IM app takes bold stand against police abuses*. <https://www.accessnow.org/south-korean-im-app-takes-bold-stand-against-police-abuses/>. Accessed: 2016-04-24. Oct. 2014 (cit. on p. 39).
- [95] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. "This POODLE Bites: Exploiting The SSL 3.0 Fallback". In: (2014) (cit. on p. 107).

- [96] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009 (cit. on p. 16).
- [97] Brian Pak. *[KakaoTalk+] LOCO UNICODE (1) (in Korean)*. <https://www.bpak.org/blog/2012/12/kakaotalk-loco-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C-%EB%B6%84%EC%84%9D-1/>. Accessed: 2016-06-16. Dec. 2012 (cit. on p. 77).
- [98] Kyungwon Park and Hyounghick Kim. “Encryption Is Not Enough: Inferring user activities on KakaoTalk with traffic analysis”. In: () (cit. on p. 17).
- [99] Siegfried Rasthofer, Steven Arzt, Robert Hahn, Max Kolhagen, and Eric Bodden. “(In)Security of Backend-as-a-Service”. In: *Black Hat Europe*. 2015 (cit. on p. 158).
- [100] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. “Haystack: A Multi-Purpose Mobile Vantage Point in User Space”. In: () (cit. on p. 62).
- [101] Bradley Reaves, Nolen Scaife, Adam Bates, Patrick Traynor, and Kevin RB Butler. “Mo (bile) money, mo (bile) problems: analysis of branchless banking applications in the developing world”. In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 17–32 (cit. on pp. 62, 123).
- [102] Ranking Digital Rights. *Corporate Accountability Index 2015*. Tech. rep. <https://rankingdigitalrights.org/index2015/download/>. New America Foundation, Nov. 2015 (cit. on pp. 40, 43, 44, 46, 50, 54, 117).
- [103] James Risen and Nick Wingfield. *Web’s Reach Binds N.S.A. and Silicon Valley Leaders*. <http://www.nytimes.com/2013/06/20/technology/silicon-valley-and-spy-agency-bound-by-strengthening-web.html>. Accessed: 2016-04-06. June 2013 (cit. on p. 16).
- [104] Choe Sang-Hun. *An Artist Is Rebuked for Casting South Korea’s Leader in an Unflattering Light*. [http://www.nytimes.com/2014/08/31/world/asia/an-artist-is-rebuked-for-casting-south-koreas-leader-in-an-unflattering-light.html?\\_r=0](http://www.nytimes.com/2014/08/31/world/asia/an-artist-is-rebuked-for-casting-south-koreas-leader-in-an-unflattering-light.html?_r=0). Accessed: 2016-04-24. Aug. 2014 (cit. on p. 35).

- [105] Choe Sang-Hun. *Korea Policing the Net. Twist? It's South Korea*. <http://www.nytimes.com/2012/08/13/world/asia/critics-see-south-korea-internet-curbs-as-censorship.html>. Accessed: 2016-06-22. Aug. 2012 (cit. on p. 34).
- [106] Choe Sang-Hun and Shreeya Sinha. *How to Get Censored in South Korea*. <https://storify.com/nytimesworld/censorship-in-south-korea>. Accessed: 2016-04-24. 2013 (cit. on p. 35).
- [107] Brahima Sanou. "The World in 2015: ICT facts and figures". In: *International Telecommunications Union* (May 2015) (cit. on pp. 11, 33).
- [108] Gandeva Bayu Satrya, Philip Tobianto Daely, and Muhammad Arief Nugroho. "Digital Forensic Analysis of Telegram Messenger on Android Devices". In: () (cit. on p. 123).
- [109] Jeremy Scahill and Glenn Greenwald. *The NSA's Secret Role in the U.S. Assassination Program*. <https://firstlook.org/theintercept/2014/02/10/the-nsas-secret-role/>. Accessed: 2016-07-06. Feb. 2014 (cit. on p. 20).
- [110] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons, 2007 (cit. on p. 16).
- [111] Bruce Schneier. *New Details on Skype Eavesdropping*. [https://www.schneier.com/blog/archives/2013/06/new\\_details\\_on.html](https://www.schneier.com/blog/archives/2013/06/new_details_on.html). Accessed: 2016-04-06. 2013 (cit. on p. 16).
- [112] S.C.S. *Why South Korea is really an internet dinosaur*. <http://www.economist.com/blogs/economist-explains/2014/02/economist-explains-3>. Accessed: 2016-04-24. Feb. 2014 (cit. on p. 34).
- [113] Lim Ji-seon. *In S. Korea, the NIS has a virtual monopoly on wire-tapping*. [http://english.hani.co.kr/arti/english\\_edition/e\\_national/719031.html](http://english.hani.co.kr/arti/english_edition/e_national/719031.html). Accessed: 2016-06-22. Nov. 2015 (cit. on p. 42).
- [114] Kadhim Shubber. *BlackBerry gives Indian government ability to intercept messages*. <http://www.wired.co.uk/news/archive/2013-07/11/blackberry-india>. Accessed: 2016-04-06. 2013 (cit. on p. 16).
- [115] Statista. *Countries with the highest average internet connection speed as of 4th quarter 2015 (in Mbps)*. <http://www.statista.com/statistics/204952/average-internet-connection-speed-by-country>. Accessed: 2016-04-28. Mar. 2016 (cit. on p. 33).

- [116] Statista. *Number of monthly active Kakaotalk users from 1st quarter 2013 to 1st quarter 2016 (in millions)*. <http://www.statista.com/statistics/278846/kakaotalk-monthly-active-users-mau/>. Accessed: 2016-06-27. 2016 (cit. on pp. 11, 30).
- [117] Yoon Sung-won. *Kakao to offer ‘privacy mode’ amid gov’t monitoring fears*. <http://www.koreatimesus.com/kakao-to-offer-privacy-mode-amid-govt-monitoring-fears/>. Accessed: 2016-04-28. Oct. 2014 (cit. on p. 45).
- [118] Hong Eun Taek. *Chief Privacy Officer*. <http://privacy.kakaocorp.com/en/philosophy/responsible>. Accessed: 2016-07-07 (cit. on p. 46).
- [119] Daniel Tay. *KakaoTalk submitted chat logs to government, pledges to stop after users protest*. <https://www.techinasia.com/daum-kakao-snooped-stopped-after-protest>. Accessed: 2016-06-22. Oct. 2014 (cit. on p. 39).
- [120] Kakao Team. *New Security Features and Improvements Coming to KakaoTalk*. <https://blog.kakaocorp.com/?p=643>. Accessed: 2016-06-22. Oct. 2014 (cit. on pp. 39, 51, 83).
- [121] Kakao Team. *Secret Chat and Decline Invites Now Available on KakaoTalk*. <https://blog.kakaocorp.com/?p=943>. Accessed: 2016-07-07. Dec. 2014 (cit. on p. 51).
- [122] Korea Internet Transparency Reporting Team. *Korea Internet Transparency Report 2015*. Tech. rep. <http://transparency.kr/wp-content/uploads/2015/10/Korea-Internet-Transparency-Report-2015.pdf>. Korea University Law School, Clinical Legal Education Center, Oct. 2015 (cit. on pp. 11, 35, 36, 39, 43).
- [123] David Thiel. *IOS Application Security: The Definitive Guide for Hackers and Developers*. No Starch Press, 2016 (cit. on p. 27).
- [124] Steve Thomas. *DecryptoCat*. <https://tobtu.com/decryptocat.php>. Accessed: 2016-07-02. 2015 (cit. on p. 155).
- [125] *UNDERSTAND SECURE DELETION OF DATA*. <https://www.nowsecure.com/resources/secure-mobile-development/coding-practices/understand-secure-deletion-of-data/>. Accessed: 2016-07-30. NowSecure, Nov. 2014 (cit. on p. 112).
- [126] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. “SoK: Secure Messaging”. In: *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE. 2015, pp. 232–249 (cit. on pp. 18, 20, 57, 94, 154, 156).

- [127] International Telecommunications Union. “Mobile-cellular telephone subscriptions 2000–2014”. In: (2015) (cit. on p. 33).
- [128] International Telecommunications Union. “Percentage of Individuals using the Internet 2000–2014”. In: (2015) (cit. on pp. 11, 33).
- [129] Raja Vallee-Rai and Laurie J Hendren. “Jimple: Simplifying Java bytecode for analyses and transformations”. In: (1998) (cit. on p. 64).
- [130] VASCO. *DigiNotar reports security incident*. [https://www.vasco.com/about-vasco/press/2011/news\\_diginotar\\_reports\\_security\\_incident.html](https://www.vasco.com/about-vasco/press/2011/news_diginotar_reports_security_incident.html). Accessed: 2016-04-08. Aug. 2011 (cit. on p. 16).
- [131] Barbara White. *Android Secure Coding Standard*. <https://www.securecoding.cert.org/confluence/display/android/Android+Secure+Coding+Standard>. Accessed: 2016-06-25. May 2015 (cit. on pp. 57, 148).
- [132] Wikipedia. *Hector Monsegur*. [https://en.wikipedia.org/wiki/Hector\\_Monsegur](https://en.wikipedia.org/wiki/Hector_Monsegur). Accessed: 2016-04-07. Apr. 2016 (cit. on p. 19).
- [133] Derek Williams. “The Tiny Encryption Algorithm (TEA)”. In: *Network Security* (2008), pp. 1–14 (cit. on p. 157).
- [134] Claud Xiao and Jin Chen. *New OS X Ransomware KeRanger Infected Transmission BitTorrent Client Installer*. <http://researchcenter.paloaltonetworks.com/2016/03/new-os-x-ransomware-keranger-infected-transmission-bittorrent-client-installer/>. Accessed: 2016-04-11. Mar. 2016 (cit. on p. 26).
- [135] Karim Yaghmour. *Embedded Android: Porting, Extending, and Customizing*. ” O’Reilly Media, Inc.”, 2013 (cit. on p. 21).
- [136] Kim Yoo-chul. *Anti-terrorism bill threatens IT ecosystem in Korea*. [http://www.koreatimes.co.kr/www/news/opinon/2016/03/264\\_199811.html](http://www.koreatimes.co.kr/www/news/opinon/2016/03/264_199811.html). Accessed: 2016-04-28. Mar. 2016 (cit. on p. 42).
- [137] Lee Youkyung. *S. Korea rumor crackdown jolts social media users*. <http://bigstory.ap.org/article/97c92b056482488abd990db9a4acb388/s-korea-rumor-crackdown-jolts-social-media-users>. Accessed: 2016-04-28. Oct. 2014 (cit. on pp. 38, 41, 45).
- [138] Chang-hoon Yun, Byung-woo Kang, Hyun-seung Lee, Da-woon Chung, Hyun-cheol Jung, and Derik Cho. *PORTFOLIO SECURITY ANALYSIS In-depth Analysis of KAKAO*. <http://advancedmanagement.net/sites/default/files/SNUCombined.pdf>. Accessed: 2016-06-27. Oct. 2015 (cit. on p. 30).

## Appendix A

# Security Assessment Checklists

In Section A.1 and Section A.2 we present two security assessment checklists that we used during our security and privacy audit of KakaoTalk.

### A.1 Checklist 1: Questions to Answer About Software Security in Any Given Android Application

This checklist outlines the most common “low-hanging fruits” or security flaws in Android apps. It was compiled from various different sources [17, 131, 93, 14] as well as from comments from Jeffrey Knockel [82].

#### A.1.1 Questions Related to Insufficient Transport Layer Protection

1. What connections are and are not protected by encryption (e.g., user authentication to the server, client-to-client communications, connections to third-parties, etc.)?
  - (a) If the app’s backend API also allows connections over HTTP: Is there a way to force the app to use HTTP instead of HTTPS?
  - (b) Is sensitive information inside a TLS connection secured by an extra layer of encryption?
2. How is the server’s certificate checked (e.g., certificate pinning, managed by the OS, etc.)?
  - (a) Are all TLS certificates in date?

- (b) Does the app verify the TLS server's hostname?
  - (c) Does the app include any code that was used for testing (e.g., `org.apache.http.conn.ssl.AllowAllHostnameVerifier` or `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER`)?
  - (d) Does the app accept self-signed certificates?
  - (e) Does the app accept third-party certificates as authorities?
  - (f) Does the app use classes that extend from `SSLSocketFactory`? Do such classes properly implement the `checkServerTrusted()` method?
  - (g) Does the app's UI detect invalid certificates and show an error to the user?
  - (h) Does the app validate the certificate chain correctly?
3. Are there any certificate pinning issues (e.g., CVE-2016-2402)?
    - (a) Does the app use any ineffective certificate pinning techniques (e.g., by using `checkServerTrusted()` incorrectly)?
  4. Does the app negotiate a weak TLS cipher suite?
  5. Is the TLS endpoint misconfigured (e.g., offering weak cipher suites, vulnerable to POODLE, DROWN attack, etc.)?
  6. Is the app vulnerable to SSL stripping attacks?

### A.1.2 Questions Related to Access Control and Information Leakage

1. Does the app leak sensitive information through files that have world readable or writable permission?
  - (a) Does the app open files with world readable file permissions?
  - (b) Does the app create world readable databases?
  - (c) Does the app make use of world writable sharedPreferences?
2. Does the app leak sensitive information through files that are included in the app's APK?
  - (a) Are there files that leak metadata (e.g., EXIF data in images)?

3. Does the app store information outside the app's sandbox (e.g., on the SD card by using the `getExternalFilesDir()` method)?
4. Does the app store data on improper protected "cloud" services?
5. Does the app write sensitive information to the system log?
6. Is there any data leakage outside the app's scope (e.g., URL caching, HTML5 data storage, analytics data sent to third parties, etc.)?
7. Does the app leak any data through error messages or crash dumps?
8. Does the app properly assign permissions to its components (Content Providers, Activities, Services, and Broadcast Receivers)?
9. Is the app vulnerable to unauthorized implicit intent sniffing and interception?
  - (a) Does the app protect its implicit intents with a permission (e.g., type `Signature` or `SignatureOrSystem`)?
  - (b) Is the app vulnerable to broadcast theft?
  - (c) Is the app vulnerable to activity hijacking?
  - (d) Is the app vulnerable to service hijacking?
  - (e) Is the app vulnerable to the interception of pending intents?

### A.1.3 Questions Related to Cryptography

1. Does the app make use of non-standard cryptography such as custom encryption protocols?
2. Does the app make use of deprecated cryptographic algorithms (e.g., RC2, RC4, MD4, MD5, SHA1, DES, etc.)?
3. How does the app manage keys? Are their any hard-coded or easy-accessible encryption keys?
4. Does the app use any insecure default settings of cryptographic security provider APIs (e.g., `Cipher.getInstance()`)?
5. How are keys and initialization vectors generated?
6. Does the app use flawed random number generators (e.g., `Random.random`)?

7. Does the app seed its random number generators correctly?
8. Does the app use insecure block cipher modes (e.g., ECB)?
9. Does the app use insecure padding schemes (e.g., RSA/ECB/NoPadding)?
10. Does the app verify the integrity of ciphertexts?
11. Does the app make use of any outdated third-party cryptographic libraries?
12. Does the app use different cryptography depending on the Android OS version, language or country?
13. Does the app make use of weak hash functions?

#### **A.1.4 Questions Related to Improper Authentication Procedures**

Which authentication methods does the app use?

1. Does the app locally authenticate the user on the client-side?
2. Does the app always authenticate the user via an online backend?
3. Does the app use device-specific tokens for authentication?
  - (a) Does the app securely generate, maintain, and destroy tokens over the life-cycle of a session?
  - (b) Does the app use easily guessable or spoofable authentication tokens?
  - (c) Does the app store tokens in URLs?
4. Does the app load data from the backend onto the mobile device already before user authentication?
5. Does the app lockout the user after a number of failed login attempts?
6. Does the app use Single Sign-On or authentication APIs (e.g., Google Apps, Facebook, OAuth, etc.)?
7. Does the app consume information from push notifications?

8. Does the app implement functionality that permits inbound connections from other devices (i.e., WiFi Direct, Android Beam, etc.)?
9. Does the app support two-factor authentication?
10. What is the app's password policy? Are 4-digit PIN numbers allowed?
11. How does the session management work?
  - (a) Is the app vulnerable to session fixation attacks?
  - (b) Are all session invalidation events executed both on the client and the server side? Are sessions properly destroyed?
  - (c) Are any session cookies properly reset after authentication state changes (e.g., switching from an anonymous user to a logged in user)?
  - (d) How does the app authenticate the user when the user's mobile phone is offline?
  - (e) How does the app implement persistent authentication?
  - (f) How big is the session timeout window?
  - (g) Is there any sensitive information in memory after session expiration?
  - (h) Does the app support a logout functionality?
12. Does the app remember login credentials? Where does it store them?
13. Does the app provide a functionality to revoke any authentication tokens remotely in case the user's mobile phone was stolen?
14. Does the app allow for intent spoofing against exported app components?
  - (a) Is the app vulnerable to intent spoofing attacks on broadcast receivers?
  - (b) Is the app vulnerable to arbitrary launch of activities?
  - (c) Is the app vulnerable to arbitrary launch of services?
  - (d) Does the app insecurely control pending intents?

### A.1.5 Questions Related to Improper Data Validation

1. Does the app properly validate input from IPC, the UI, the network, and the local filesystem?
  - (a) Is the app vulnerable XSS/HTML/XML/CSS injection attacks?
  - (b) Is the app vulnerable to local filesystem injection attacks (e.g., web content that is cached on the SD card)?
  - (c) Is the app vulnerable to SQL injection attacks?
    - i. Does the app make use of raw SQL queries?
2. Is the app vulnerable to intent injection attacks through the UI or IPC?
  - (a) Does the app properly check for NULL values in intent fields?
  - (b) Does the app generate files using improperly validated input?
  - (c) Is the app vulnerable to log injection attacks?
3. Does the app properly secure its WebViews?
  - (a) Is browser plugin support disabled?
  - (b) Is local file access disabled?
  - (c) Is Javascript disabled?
  - (d) Is content URL access disabled?
  - (e) Does any WebView load unauthenticated content?
4. Does the app have memory corruption bugs in native code?

### A.1.6 Questions Related to Software Updates

1. Does the app check for updates itself, independently of an app store?
2. Does the app update itself using the `ACTION_VIEW` intent and downloaded APK's?
3. Does the app verify the digital signature of a downloaded APK before prompting the user? Is the APK also verified to be a newer version of the app being updated?
4. Does the app silently update itself (e.g., via a `DexClassLoader` or native library)?

### A.1.7 Miscellaneous Questions

1. Does the app load external code at runtime (e.g., via DexClassLoader’s or native libraries)?
2. Does the app interact with any other apps, third-party services or data?
3. Does the app make use of any APIs (e.g., payment gateways, SMS messaging, social networks, cloud file storage, advertising networks, etc.)?
4. Does the app make use of more permissions than it actually requires?
5. Does the app make use of any outdated (third-party) libraries?
6. Can parts of the source-code be found online (e.g., on GitHub)?
7. Is the “debuggable” flag in the AndroidManifest set?
8. Does the app detect a rooted mobile handset?
9. Does the app detect an attached debugger?
10. Does the app detect a virtual machine environment?
11. Does the app perform runtime code integrity checks?
12. Does the app encrypt its SQLite databases?

## A.2 Checklist 2: Questions to Answer About End-to-end Encryption in any Given Chat Application

This checklist tries to provide a more specialized set of weaknesses that are common for secure mobile chat apps. It was mainly authored by Jedidiah Crandall [37] and was additionally influenced by Unger’s evaluation framework [126], and Green’s “Cryptographic Engineering” blog<sup>1</sup>.

---

<sup>1</sup><http://blog.cryptographyengineering.com/>

### **A.2.1 Questions Related to Marketing Claims, ToS, and EULA**

1. What marketing claims (e.g., in blog posts) does the company make that may be relevant to encryption or security properties?
2. If they claim end-to-end encryption, is it truly end-to-end or just client-to-server?
3. Is there anything in the End User License Agreement (EULA) or Terms of Service (TOS) that prohibits reverse engineering?
4. What is the contact information for reporting vulnerabilities?
5. Does the local government enforce crypto bans or mandated designs?

### **A.2.2 Questions Related to Random Number Generation**

1. Where do random numbers come from (e.g., the operating system through `/dev/urandom`, or somewhere else) [45]?
2. What are random numbers used for (e.g., key material, initialization vectors, nonces, session keys)?
3. Is entropy destroyed because of bad type casting (e.g., as in Cryptocat) [124]?
4. Are there potential device or source issues with entropy (e.g., the entropy pool is not properly seeded with high entropy events) [18, 81]?

### **A.2.3 Questions Related to Symmetric Cryptography**

What symmetric cryptography algorithm(s) is/are used? For each instance of symmetric cryptography:

1. Where does the Initialization Vector come from [45]?
2. Where does the shared secret key come from?
3. What is the key size?
4. Is it a block or stream cipher?

5. If block encryption, is it done in ECB mode, CBC mode, or something else [45]?
6. If stream cipher, is key material ever reused?

#### **A.2.4 Questions Related to Asymmetric Cryptography**

1. What algorithm is used for key exchange?
2. What parameters are used for key exchange (e.g., which elliptic curve)?
3. If not a well-known set of parameters, does it have issues with, e.g., group sizes [45]?
4. What is the key size?
5. Where does the public key come from?
6. How does public key discovery work, e.g., does the service provider use a directory service for storing and exchanging the keys?
7. Where does the private key come from?
8. Is the private key encrypted with the user's password (which is also used to authenticate with the service provider)?
9. Does the service provider have cleartext access to the user's password?
10. How is authentication performed (e.g., not at all, Trust on First Use – or TOFU, visual or auditory verification of keys by users, etc.) [126]?
11. What is the security of the discovery transport?
12. Where is the discovered public key stored?
13. Are there any possible birthday or meet-in-the-middle attacks [45]?
14. Are there any possible replay attacks [45]?
15. Is there a way for users to know when a MiTM attack has occurred, such as transparency logs [126]?
16. Is the algorithm semantically secure (i.e., does it use OAEP or something like that if they are using an algorithm that is not already proven to have semantic security [7])?

17. Is there any form of key management? I.e., are asymmetric keys ever revoked or expired? What happens when the user loses his/her phone (and his/her private key) and he/she needs to issue a new public key?
18. Does the algorithm for key exchange preserve forward secrecy [45]?
19. Is large integer arithmetic performed using a standard library or custom code? Is wooping used? [45]
20. Public key availability: Once you validated a public key the validation needs to be available on all possible end devices as well. How is that handled?

### **A.2.5 Questions Related to Hash Functions or MACs**

1. What hash function and Message Authentication Code (MAC) algorithms are used?
2. What exactly is each used for?
3. Are any of the above algorithms known to be susceptible to preimage attacks (e.g., CRC32 or something based on TEA) or collision attacks (e.g., MD5 or SHA1) [133]?
4. Do they MAC and then encrypt (insecure) or encrypt and then MAC (secure)? E.g., do they include a hash of the plaintext in the ciphertext as does Telegram [63]?

### **A.2.6 Questions Related to Traffic Analysis**

1. Are communications direct from user to user or do they go through a central server (or something else)?
2. If a central server is involved, can the plaintext or ciphertext between two users be pattern matched or is it re-encrypted by the server?
3. Is there anything that is sent plaintext, like stickers?
4. What metadata is transmitted alongside encrypted messages? Anything unusual or potentially compromising?
5. If a central server is not involved, can a passive eavesdropper learn about user's actions based on the characteristics of the encrypted traffic?

### A.2.7 Questions Related to Forensics

1. What compromising information can be found with physical access to the phone, especially the filesystem?
2. If user passwords are stored, is hashing, salting, and/or stretching applied [45]?
3. Is it possible to dump any session keys from memory? Can an attacker intercept and read messages before they get encrypted? Where and how are the private and session key(s) stored on the device?

### A.2.8 Miscellaneous Questions

1. Are the crypto algorithms used self-rolled or standard, well-known algorithms?
2. Are there any custom modifications to any crypto?
3. Is there anything else that increases or decreases the trust of the crypto implementation? E.g.:
  - (a) automatic background updates that replace your implementation with a malicious one
  - (b) use of subliminal channels to exfiltrate keys
  - (c) a maintained list of minor vulnerabilities in the communication protocol (e.g., uninitialized memory to scan key material)
4. Is it possible to register a shadow device to an existing user account?
5. Does the company implement some sort of key escrow (e.g., encrypt the session key with a third-party public key and send it in an extra field along the encrypted message)?
6. Does E2E work between different mobile operating systems or versions (e.g., between iOS and Android)?
7. Does encryption work on all network interfaces? Do they use E2E over 3G, for example?
8. Does the crypto implementation use any insecure BaaS solutions [99]?
9. Data availability: Is the secret chat history stored in the cloud so that it is available on all end devices? Or does the app implement some other sort of secure synchronization protocol instead?

10. Does the app support the following privacy-related configuration settings?
  - (a) Adding a new contact requires permission from that contact
  - (b) Automatic file downloads/transfers can be disabled
  - (c) The app detects malicious file extensions
  - (d) Application and notifications can be locked with a passphrase
  - (e) Application allows to auto-lock itself after a specified time interval
  - (f) Messages and the chat history can be deleted
  - (g) Old messages can be automatically deleted
  - (h) The app allows to create encrypted backups of messages
  - (i) Presence information can be restricted
  - (j) Notifications can be disabled or muted
  - (k) Contacts can be blocked
  - (l) Contacts can be verified
  - (m) The app blocks other apps to take screenshots

## Appendix B

# Application Information

In this appendix we list KakaoTalk’s permissions (Section B.1) as well as several application details (Section B.2).

### B.1 Permissions

1. `android.permission.CALL_PHONE`
  - (a) Directly call phone numbers (this may cost money)
  - (b) Allows the app to call phone numbers without the user’s intervention. This may result in unexpected charges or calls. Note that this does not allow the app to call emergency numbers. Malicious apps may cost money by making calls without the user’s confirmation.
2. `android.permission.READ_PHONE_STATE`
  - (a) Read phone status and identity
  - (b) Allows the app to access the phone features of the device. This permission allows the app to determine the phone number and device IDs, whether a call is active and the remote number connected by a call.
3. `android.permission.SEND_SMS`
  - (a) Send SMS messages
  - (b) Allows the app to send SMS messages. This may result in unexpected charges. Malicious apps may cost you money by sending messages without your confirmation.

4. `android.permission.RECEIVE_SMS`

- (a) Receive text messages (SMS)
- (b) Allows the app to receive and process SMS messages. This means that the app could monitor or delete messages sent to the user's device without showing them to him/her.

5. `android.permission.CAMERA`

- (a) Take pictures and videos
- (b) This permission allows the app to use the camera at any time without the user's confirmation.

6. `android.permission.RECORD_AUDIO`

- (a) Record audio
- (b) This permission allows the app to record audio at any time without the user's permission.

7. `android.permission.ACCESS_COARSE_LOCATION` and `android.permission.ACCESS_FINE_LOCATION`

- (a) Approximate location (network-based), precise location (GPS and network-based)
- (b) Apps may use this (GPS, mobile towers, and WiFi) to determine where the user is, and may consume additional battery power.

8. `android.permission.READ_CONTACTS`

- (a) Read user's contacts
- (b) Allows the app to read data about the user's contacts stored on his/her phone, including the frequency with which the user has been called, emailed or communicated in other ways with specific individuals. This permission allows apps to save user's contact data, and malicious apps may share contact data without user's knowledge.

9. `android.permission.READ_EXTERNAL_STORAGE` and `android.permission.WRITE_EXTERNAL_STORAGE`

- (a) Modify or delete the contents of the user's SD card
- (b) Read the contents of the user's SD card

10. `android.permission.GET_ACCOUNTS`

- (a) Find accounts on the device
- (b) Allows the app to get the list of accounts known by the phone. This may include any accounts created by applications that the user has installed.

11. `android.permission.CHANGE_NETWORK_STATE`

- (a) Change network connectivity
- (b) Allows the app to change the state of network connectivity

12. `android.permission.CHANGE_WIFI_STATE`

- (a) Connect and disconnect from WiFi
- (b) Allows the app to connect to and disconnect from WiFi access points and to make changes to device configuration for WiFi networks.

13. `android.permission.INTERNET`

- (a) Full network access
- (b) Allows the app to create network sockets and use custom network protocols. The browser and other applications provide means to send data to the Internet, so this permission is not required to send data to the Internet.

14. `com.android.vending.BILLING`

- (a) Google Play billing service
- (b) Allows the user to buy items through Google Play from within the app.

15. `com.google.android.c2dm.permission.RECEIVE`

- (a) Receive data from Internet
- (b) Allows the app to accept cloud-to-device messages sent by the app's service. Using this service will incur data usage. Malicious apps could cause excess data usage.

16. `android.permission.ACCESS_NETWORK_STATE`

- (a) View network connections

- (b) Allows the app to view information about network connections such as which networks exists and are connected.

17. `android.permission.ACCESS_WIFI_STATE`

- (a) View WiFi connections
- (b) Allows the app to view information about WiFi networking, such as whether WiFi is enabled and name of connected WiFi devices.

18. `android.permission.BLUETOOTH`

- (a) Pair with Bluetooth devices
- (b) Allows the app to view the configuration of the Bluetooth on the phone and to make and accept connections with paired devices.

19. `android.permission.RESTART_PACKAGES`

- (a) Close other apps
- (b) Allows the app to end background processes of other apps. This may cause other apps to stop running.

20. `android.permission.GET_TASKS`

- (a) Retrieve running apps
- (b) Allows the app to retrieve information about currently and recently running tasks. This may allow the app to discover information about which apps are used on the device.

21. `android.permission.RECEIVE_BOOT_COMPLETED`

- (a) Run at startup
- (b) Allows the app to have itself started as soon as the system has finished booting. This can make it take longer to start the phone and allow the app to slow down the overall phone by always running.

22. `android.permission.SYSTEM_ALERT_WINDOW`

- (a) Draw over other apps
- (b) Allows the app to draw on top of other apps or parts of the user interface. They may interfere with the user's use of the interface in any apps, or change what the user thinks he/she is seeing in other apps.

- 23. `android.permission.VIBRATE`
  - (a) Control vibration (affects battery)
- 24. `android.permission.WAKE_LOCK`
  - (a) Prevent phone from sleeping
- 25. `android.permission.MODIFY_AUDIO_SETTINGS`
  - (a) Change your audio settings
  - (b) Allows the app to modify global audio settings such as volume and which speaker is used for output.
- 26. `com.android.launcher.permission.INSTALL_SHORTCUT`
  - (a) Install shortcuts
  - (b) Allows the app to add homescreen shortcuts without user intervention.
- 27. `android.permission.BROADCAST_STICKY`
  - (a) Send sticky broadcast
  - (b) Allows the app to send sticky broadcasts, which remain after the broadcast ends. Excessive use may make the phone slow or unstable by causing it to use too much memory.

In addition, KakaoTalk defines the following custom permissions:

- `com.kakao.talk.permission.C2D_MESSAGE`
- `com.kakao.talk.permission.RECEIVE_NOTIFICATION`
- `com.kakao.talk.permission.FRIENDS_PICKER`
- `com.kakao.talk.permission.START_CHAT`
- `com.kakao.talk.permission.ADD_FRIEND_AND_START_CHAT`
- `com.kakao.home.permission.SNOOZE`
- `com.kakao.talk.permission.INTERNAL`

## B.2 Application Details

Description	Value
SHA256	ca5e56b8ea1e53bad462e2561028685fdc0abc839f678af538a409a047e477ec
SHA512	ebea2a5f361505cef6ab81d4150f23300ff36ce21766dbd8597c867a0031c4fe0da450372ac70fa53bdf9c0d6a86428e634a668336306b3104c39c3feb4750f5
Application label	KakaoTalk
Package name	com.kakao.talk
Package version name	5.5.5
Package version code	1400223
Minimum SDK version	14
Target SDK version	22
Launcher Activity	com.kakao.talk/.activity.SplashActivity
Main Activities	com.kakao.talk/.activity.main.MainTabFragmentActivity com.kakao.talk/.activity.TaskRootActivity
APK path	/data/app/com.kakao.talk-1.apk
Private data directory	/data/data/com.kakao.talk/
External data directories	/sdcard/KakaoTalk/ /sdcard/Android/data/com.kakao.talk/

Table B.1: KakaoTalk Application Details.

TRITA TRITA-EE 2016:131