


Technische Universität München
Department of Informatics

Interdisciplinary Project in Electrical Engineering

Investigating Mobile Messaging Security

Elias Hazboun


Technische Universität München

Department of Informatics

Interdisciplinary Project in Electrical Engineering

Investigating Mobile Messaging Security

Untersuchung von Mobile Messaging Sicherheit

Author	Elias Hazboun
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Matthias Wachs, Quirin Scheitle
Date	April 27, 2016

Informatik VIII
Chair for Network Architectures and Services

Abstract

In this report we document our work in analyzing the security of a selection of mobile messaging apps. Focusing on network based security, we studied traffic generated by the apps to gain an understanding of the current state of applying encryption and authentication protocols. Our findings show a positive trend in security with developers steadily improving security with newer app updates partly due to the increased scrutiny from the community and academia. Although not all apps analyzed had perfect state of the art security properties, none have shown any major vulnerabilities exploited by normal adversaries. It was also evident that only one app - namely TextSecure - is using the industry standard TLS for server-client authentication and security while others have opted for custom made protocols and algorithms.

Contents

Chapter 1

Introduction

The proliferation of smartphones and fast cheap internet connection over mobile networks meant that users opted to communicate using mobile apps that use the Internet instead of sending messages over regular cellular networks. This is a predictable phenomenon, given that for the past few decades society has been increasingly reliant on digital communication and trying to leverage platforms such as the Internet for inter-personal communication. For example, E-mail has been in widespread use since the 1980s [1] and XMPP (originally jabber) since the early 2000s [2].

However, one big difference between protocols like the ones behind E-mail and the protocols powering mobile messaging apps today is that the former are open standardized protocols which were built to allow different providers to interoperate, analogous to the interoperability between snail mail service providers [3]. The majority of the most popular mobile messaging apps relies on underlying proprietary communication protocols that do not support compatibility between them [4]. On the one hand, the closed platform architecture is very appealing from a business point of view; app makers get to incorporate added value features more easily and encourage users to stay loyal to their platforms or users might risk getting cut-off from their social circle. On the other hand, this black-box approach and closed environment hinders compatibility and competition and most importantly means that when using these apps, users are not able to scrutinize the method in which their data is being handled in order to be delivered to the intended recipient but must place their blind trust in the app makers discretion [4].

Moreover, security in private digital communication such as email or social media has been the focus of research ever since the scienitific community realized its power. For example, standard protocols like OpenPGP [5] were invented to secure E-mail communication. Nevertheless, adoption of these secure protocols

has been lacking. The vast majority of messages today are still sent unsecure and without complete proofing for privacy, which means that as a society, we have concluded that the risk of non-secure digital communication does not match the effort needed to secure it using currently offered secure solutions [6]. This problem adds another layer of complexity to assessing the security of mobile messaging apps and to the ongoing effort of securing them.

1.1 Research Question

The goal of this project is to investigate the security properties of a selection of today's most popular mobile messaging apps in terms of encryption, authentication and standardized security protocol usage in communication.

1.2 Outline

The report is structured as follows. In Chapter 2, we outline the methodology we took and give an introduction into the apps and some necessary security terms. While in Chapter 3, we describe related work and the analysis findings on a per app basis. Finally, we conclude our report in Chapter 4 where we summarize our findings.

Chapter 2

Background and Related Work

In the following chapter, we outline our methodology, define some key security terms necessary for the topic at hand, and give a short overview of the related work that has been done on mobile messaging security.

2.1 Methodology

2.1.1 App Selection

This project continues on a previous research effort conducted at the Chair of Network Architectures and Services [7], where some of today's most popular mobile messaging apps were tested in terms of security in relation to users' geo-location using an automated test framework. The selection of apps to be tested was determined based on popularity, security measures, architecture and geographical server distribution. The apps selected were WhatsApp [8], Threema [9], WeChat [10] and TextSecure [11] (now known as Signal). The testing was performed on the latest version available on the Android marketplace for each app. More information on the previous work and its resutls can be found in [7].

2.1.2 Approach

As a first step towards answering our research questions, we investigated the official app documentation published by the developers. However, since only one app out of the four is actually open source, we did not envision finding a thorough description of the inner workings of the app protocols especially in terms of security. Another source of valuable information for our work was the past research on this topic done by the scientific community and interested hackers. While this information could be potentially outdated by later app updates, it

gave us a glimpse of the basic app architecture and its approach in securing users' data.

Finally, the main method to investigate these apps was by analyzing their network traffic, that is the network packets being sent and received by the app as seen by an eavesdropper on the network. The packet capture process itself has been carried out by the aforementioned research effort at our chair. The process was done by using two different mobile devices to communicate through the apps while capturing the resultant traffic in parallel. In total, there were 3256 capture files split equally among the two mobile phones and the four apps for a total of 407 captures per phone per app. 406 of which were of normal traffic generated by the app when sending and receiving messages, and one was of the setup process performed when the phone number is registered for the first time with the app server.

Analysis of these capture files was done in two steps. Firstly, we manually inspected them by hand using Wireshark packet capture tool [12]. Points of interest we focused on were protocol usage, the existence of encryption and its type and the existence of certificates for authentication. Secondly, after having a broad idea of the type of traffic sent by the apps, we automated the analysis process into scripts that could parse the multitude of capture files.

2.2 Definition of Security Concepts

In the following we present some key concepts that are important for the understanding of the discussion that follows in Chapter 3.

2.2.1 Layers of Encryption

Securing messages against eavesdroppers can be done on two different layers of communication as shown in Figure 2.1. The first layer, end-to-end encryption, is performed on the plaintext of the message using keys that are only known to the sender and the intended recipient. The server or any other party handling the routing of such message cannot decrypt it to reveal the plaintext [13]. The second layer, transport layer encryption, is performed on messages between the server and the client using keys shared between them. Transport layer encryption can be done on plaintext or on messages already secured by end-to-end encryption [14]. For secure communication, it is necessary to use both layers of encryption. End-to-end encryption is important when the server cannot be trusted or when governments can force companies into revealing users' data, while transport layer encryption is important against network level attackers and eavesdroppers.

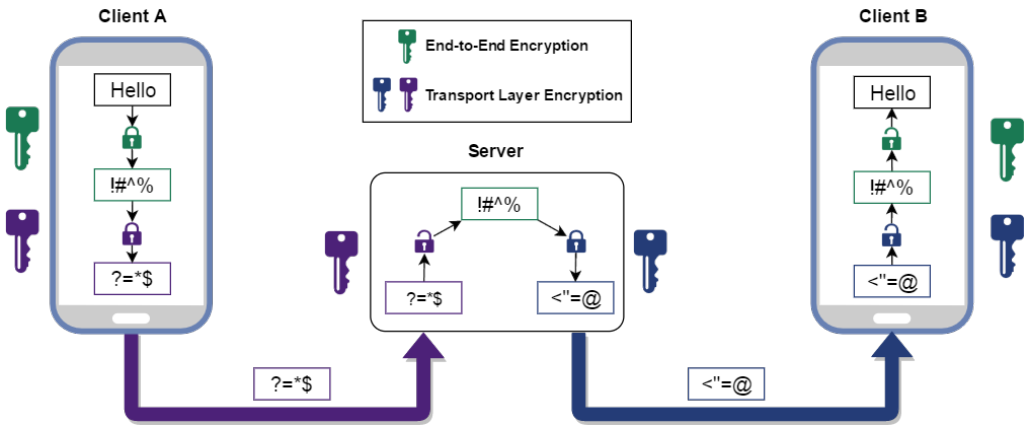

Figure 2.1: Transport Layer and End-to-End Encryption

2.2.2 Transport Layer Security Protocol (TLS)

TLS [15] and its predecessor Secure Sockets Layer (SSL) are the current industry standard for establishing secure connections between a client and a server over the Internet. They form the backbone of secure HTTP (HTTPS) for millions of websites ranging from banking to online shopping. They support this by providing multiple services including authentication, integrity and confidentiality [15].

2.2.3 Perfect Forward Secrecy (PFS)

The term ”Perfect Forward Secrecy” refers to a property possessed by a communication protocol which guarantees that the compromise of long-term keys, used to negotiate session keys, does not compromise the secrecy of past session keys [16]. The most common way to achieve PFS is by Diffie-Hellman (DH) key exchange using ephemeral keys for sessions and restricting the use of long term private keys to authentication only [17]. This property is important against a global passive adversary model, where an attacker can monitor and store traffic for an extended period of time until they gain access to long term keys with which (without PFS) they can decrypt all stored traffic.

2.2.4 Asynchronous Messaging Security

In asynchronous messaging such as E-mail or instant messaging, security can be a challenge compared to synchronous forms of communication, mainly because communicating parties are not guaranteed to be online at the same time to perform key negotiation and other security procedures [18]. In E-mail, protocols

like OpenPGP and S/MIME [19] have been proposed and used, though they lack features such as PFS for example. Other solutions include the use of trusted third parties for distribution of short term keys while others like Authenticated Diffie-Hellman do not.

Protocols for instant messaging have been created like Off-the-Record Messaging (OTR) [20] where a Diffie-Hellman Ratchet is used to create ephemeral keys for each communication session, this guarantees backward secrecy but session keys might not be renewed for each message, so PFS is only partially provided [18]. Another approach - which was used in Silent Circle Instant Messaging Protocol (SCIMP) [21] - is by using key derivation functions (KDF) to compute future message keys from past keys. This provides PFS, but does not provide backward secrecy within conversations; if a key is compromised, all future keys can be derived using the KDF [22].

2.2.5 Certificate Pinning

Digital certificates are electronic documents used for security purposes to verify the identity of a party in communication. The certificate is said to bind a public key to an entity; that is, it guarantees that a person or a machine is truly the owner of the public key which it claims to own [23]. This is crucial in TLS protocol where a rogue website could otherwise claim to be a legitimate one and perform a Man-in-The-Middle (MiTM) attack. However, some level of trust that an entity is providing their own certificate and not a fake one is still needed, that's why certificates are usually signed by a trusted third party called a certificate authority in order to attest for the entity's claims [24].

To remove this need of trusted third parties or simply to add another security measure to it, "certificate pinning" can be used in software where communication should take place with a specific server or entity. It means that developers insert the certificate or sometimes the public key in the software itself, and if a communicating party fails to present a matching certificate or public key, the software concludes that this party is not legitimate [25].

2.3 Related Work

Interest in analyzing the inner-working of mobile messaging apps especially in terms of security and privacy has been growing in the past few years, not only in the scientific community anymore, but also in the general public due in part to the Snowden leaks [26] in 2013 which exposed governmental schemes of mass Internet communication surveillance [27]. Today, almost all major apps have been

the subject of at least one or more research studies and almost all have since been updated to incorporate fixes of major flaws found.

Different research projects have focused on different aspects of security and privacy. Research like Mueller et al [28] focused on vulnerabilities in the authentication process when registering a phone number and the ability to send unsolicited messages to victims. Other studies like [29] and [30] focused on the local security of apps; that is the privacy of messages stored on the mobile phone or the ability to retrieve sensitive information such as encryption keys from the app's local database. Finally, research such as [31] and [32] focused on analyzing network traffic generated by said apps, whether when sending messages or using VoIP features or simply authenticating with the server.

In general, early results of research efforts on this topic have shown that developers have been very negligent in securing users' data, to the point of sending messages in clear text, or using very weak encryption schemes. Nevertheless, later research has pointed out that there is a positive trend in enforcing security measures. In fact some developers push increased privacy and security as features of their apps; citing increased demand for privacy in communication by customers. For example, Threema and TextSecure were developed and marketed on the premise of being more secure than their competitors.

Chapter 3

Analysis

In this chapter, we discuss the work that we have done. Each section is dedicated to a single app where we give a short description of it, followed by the related work surrounding it and finally analyzing the app security and outlining our results.

3.1 TextSecure

TextSecure is an app developed by Open Whisper Systems for Android OS in 2010. It promises end-to-end encryption and is developed with security measures as its selling point to customers. It is the only one out of the four apps to be completely open source. It has been recently discontinued and replaced with a more capable app called Signal, though the base of its text-based messaging protocol remained the same namely "Signal Protocol".

3.1.1 Related work

TextSecure security analysis has been focused on direct inspection of its open source code and the implementation of their security algorithms. For example, Frosch et al [33] concluded that apart from some peculiarities in the security design, TextSecure is successful in providing stateful authenticated encryption. Their most notable peculiarity was the protocol's susceptibility to an Unknown Key-Share attack (UKS) [34] of which they have proposed a fix and informed the developers who acknowledged their findings.

3.1.2 Analysis and Results

In the following section, we describe the major aspects of TextSecure security. As it is open source, its developers openly publish that TextSecure uses both end-to-end and transport layer encryption. For end-to-end encryption, it uses "Axolotl ratchet" also called "double ratchet", because it combines DH ratchet and key derivation based ratchet to provide both PFS and backward secrecy. Messages are encrypted using keys produced by key derivation ratchets which are in turn produced by seeding session keys from DH ratchet [22]. As for transport layer, it uses standard TLS protocol to secure the communication with the server. In all of our capture files of TextSecure, the server used a self-signed certificate and chose cipher suite ECDHE_RSA_WITH_AES_128_GCM_SHA256 for communication. TextSecure also uses certificate pinning to guard against fake certificates [35].

To cope with asynchronous nature of instant messaging while still providing PFS and easy user experience, the Textsecure app preemptively calculates 100 prekeys (signed key exchange messages) and sends them to the server. A client wishing to communicate with someone would ask the server for the next valid prekey and calculates a shared secret using its own and the destination prekeys, and encrypts the message using it [36]. After this first step, the aforementioned Axolotl ratchet enters full force to regularly update the encryption and MAC keys for every outgoing message.

For symmetric encryption, TextSecure uses AES in counter mode without padding, but when the server receives the encrypted message, it encrypts it in another layer before handing it off to Google Cloud Messaging server to deliver it to the intended recipient. This second layer of encryption also uses AES but in CBC mode with PKCS5 padding, and is needed to guarantee that Google's server cannot know the sender of the message but only the recipient [33].

Table 3.1 gives a listing of the security primitives used by TextSecure. An in-depth analysis of these choices is not in the scope of this research effort.

Table 3.1: TextSecure security primitives

Process	Algorithm	Key Length
Key derivation	Axolotl Ratchet	128
Symmetric encryption	AES	256
Integrity protection	HMAC-SHA2	256

3.2 Threema

Threema is a popular mobile messaging app in German speaking countries, it is developed by Threema GmbH with a promise of end-to-end encryption and hosting data only on servers in Switzerland. Part of its code base is open source while the majority is proprietary. Like TextSecure, it is focused on the security and privacy of user's data as its selling point.

3.2.1 Related work

In his work, Jan Ahrens [37] provided an analysis on Threema protocol especially in terms of encryption and key exchange. Most of his findings about general encryption behavior are aligned with what is officially published by Threema [38]. His work also shed light on the binary protocol that Threema uses in terms of packet format and the message code denotation. Part of Dimitrov, Laan and Pineda [39] analysis of Threema has also focused on network traffic approach like the one we are conducting, and their results have demonstrated with high probability that certificate pinning for TLS communication was present in the app code because when presented with fake or edited certificates, the app dropped all communication and in some cases even prevented future communication with the IP address providing fake certificates.

3.2.2 Analysis and Results

Our analysis of Threema's network capture files have demonstrated that their handshake process for user authentication is similar to what is described in [37]. During the authentication process, a three way handshake is performed between client and server. The client and server exchange freshly generated short term keys and nonces to be used in this session. As for the messaging protocol itself, it does not use standardized protocols like TLS for transport layer security over TCP, but rely on proprietary binary protocol with security properties implemented using the NaCl Library [40]. However, the directory services protocol that is the protocol used to discover users among other things as well as the media protocol, both employ standard HTTP over TLS with certificate pinning. The cipher suite observed in our analysis was ECDHE_RSA_WITH_AES_256_GCM_SHA384.

In the following we will give a brief explanation of the security scheme currently used in Threema, this information is the aggregate of our work and the various sources previously mentioned.

When a user runs Threema for the first time on a device, a long term public-

private key pair is generated which will be stored locally on the device, of which only the public key will be sent to the server as identification. The server responds back by creating a unique 8 character ID and binds it to that public key in its directory server for future user queries. When a user wants to communicate with another user whom she knows the public key of, she generates a shared key using Diffie–Hellman (DH) as shown in Figure 3.1. This shared key will be used to encrypt the messages and will be equal to the one created by the recipient using the same method.

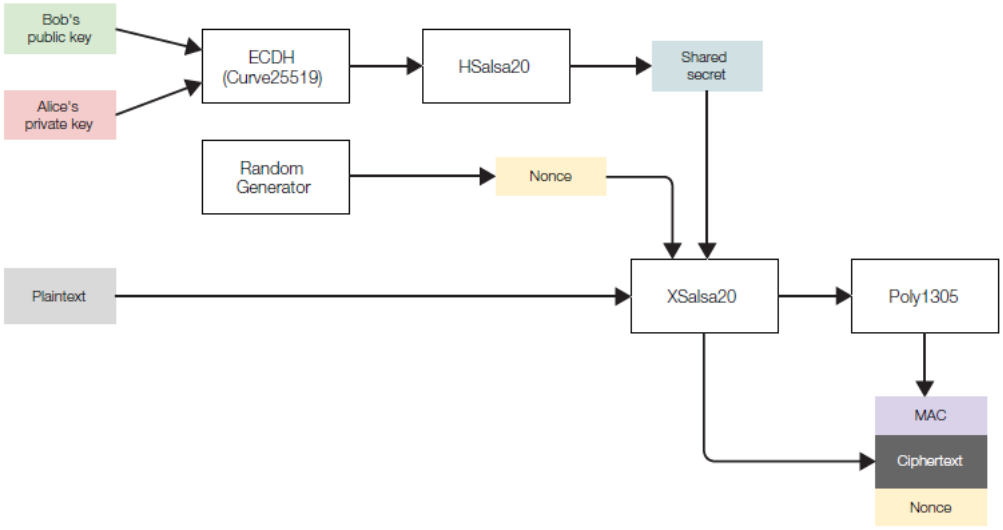

Figure 3.1: Threema End-to-End Encryption [38]

As for transport security, whenever Threema restarts (or up to a maximum of seven days), it creates a fresh temporary key pair for the communication with the server. The server also creates its own short term key pair and sends its public half to the client encrypted with the server's long term private key to prove its identity. These short term keys will be used to encrypt data between the client and the server.

Therefore, Threema provides PFS on the transport layer, but not on the on end-to-end level. Threema developers' justification for it is that the implementation of end-to-end PFS is not suitable for the nature of mobile messaging due to users going offline, the increased complexity of clients interactions and the overhead of key establishment.

Table 3.2 gives a listing of the security primitives used by Threema. An in depth analysis of them and their implementation is again not in the scope of this research effort. However, the use of NaCl library is a good indication since it is a crypto library designed to minimize operator errors and past studies have proven its security.

Table 3.2: Threema security primitives

Process	Algorithm	Key Length
Key derivation	ECDH over Curve25519	256
Symmetric encryption	XSalsa20	256
Integrity protection	Poly1305-AES	128

Threema is a closed source app, but that does not bode well for its adoption by privacy minded people it aims to gain their trust, therefore it has allowed for an external audit of its security aspects. The audit report [41] has found no deviations between what is published by Threema in its white paper and the actual implementation.

Finally, after inspecting the size of the messages in the capture files, we suspected random padding is being done to conceal the true size of plain text messages since identical messages resulted in different packet sizes in the capture files we observed. This was confirmed by [37] and aligns with official Threema documentation.

3.3 WeChat

One of the largest messaging apps by monthly active users, WeChat is developed by Tenecent in China and has over 650 million active users, mostly in Asia [42]. It is a feature rich app with many additions to the usual text and voice messages. It is completely closed source and apart from a public API to send and receive messages through their services, its developers do not communicate publicly about its inner working.

3.3.1 Related work

Work analyzing WeChat security in the English language academia is rather limited. Work such as [31] has focused on VoIP traffic security, while [43] has focused on forensic analysis of the app artifacts on iPhone. Finally, the most similar work to our focus was done by Roberto Paleari [44], where he analyzed network traffic and concluded that WeChat is using a custom protocol somewhat similar to HTTP/S.

3.3.2 Analysis and Results

We have found that WeChat indeed uses a custom communication protocol to deliver messages, and also even uses custom HTTP queries to perform DNS queries

as discussed in more detail in [7]. WeChat symmetric encryption algorithm is AES with 256 bit key length, although not clearly visible in the capture files themselves, it is alluded to as the encryption method for "official" WeChat accounts (accounts for verified individuals, i.e. celebrities), and we speculate that they are using the same length for normal accounts as well but unfortunately without concrete evidence. Finally, in [44] it is suggested that key derivation is done using RSA method but we could not verify this ourselves. As for end-to-end encryption and PFS, we could not find any evidence in the literature or in the packet capture files of their existence, and we conclude that they are not available in WeChat. All in all, after analyzing the relevant files, we can only say that WeChat developers were not very wise in creating their own custom communication protocol with many oddities and also in not being very open about their app security details. However, we realize that this could be due to Chinese restrictions on Internet or due to cultural differences in software development.

3.4 WhatsApp

Developed by WhatsApp Inc., WhatsApp is the second most popular messaging app in the world with over 1 billion users as of February 2016 [45]. Its popularity prompted Facebook Inc. to buy it in 2014, a move which prompted some users to abandon it citing privacy concerns [46]. The app is closed source like WeChat but has announced in late 2014 a partnership with Open Whisper Systems to incorporate the encryption mechanisms of the open source TextSecure protocol within WhatsApp itself [47]. Our work affects the state of WhatsApp before the completion of the partnership on the 5^{th} of April 2016[1].

3.4.1 Related work

Due to its massive popularity, WhatsApp has been the target of numerous studies on its security whether from hackers or researchers. Multiple research papers such as [28] and [48] have analyzed among other apps the authentication process of WhatsApp and have demonstrated that a MiTM attack can be carried out against the authentication process of WhatsApp to hijack user accounts. Furthermore, due to address book upload feature, it is possible to enumerate random phone numbers to test whether or not their respective users were using WhatsApp. Fortunately, the authentication process has been updated as evident in our capture files where the verification PIN is now generated on the server instead of

[1] WhatsApp announced on 5 April 2016 that it has completed the transition process to end-to-end encryption. A quick look at the announcement reveals that the security architecture is revamped and even user authentication is now possible.

the client, which renders the attack in [28] to be irrelevant. However, the address book enumeration exploit is still existent and has proven to be an issue in the balance between privacy and ease of use of instant messaging apps in general. Work from enthusiasts have spawned WhatsApp API clones such as [49] and [50] where they rely on reverse engineering of traffic and code to understand the inner workings of WhatsApp. While their work is not focused on finding vulnerabilities or assessing security properties of WhatsApp per se, it is still valuable since it is almost identical to WhatsApp and is updated regularly to the point where one can fully use it instead of an official WhatsApp client. Finally, Karpisek, Baggili and Breitinger [32] have focused on decrypting network traffic of text messages and VoIP calls using WhatsApp which has proven beneficial in our work.

3.4.2 Analysis and Results

WhatsApp security has proven to be the most changing in our work. Reading through the past literature and various web articles indicated that WhatsApp transitioned through multiple states of security provisions and that the general public and users are not completely aware of it but in fact confused and misinformed.

WhatsApp started out as plain text messaging [51], then moved to a very weak transport layer encryption where the password used for encryption key derivation was generated from the MAC address or the IMEI of a mobile phone both of which are not secure enough [52].

Fortunately, the current version of the WhatsApp protocol which we tested has been since then improved and in the following we will try to summarize our findings. When a WhatsApp client registers with the server using a new phone number, the server generates a 20-byte-long password (pw) and sends it over secure TLS connection to the client which will save it and will be bound to that phone number for the lifetime of the registration. When a client subsequently communicates with the server for the sake of sending messages it performs a three way handshake where the first step is a client hello, followed by a server hello piggybacked on it a 20 byte challenge data. The client performs the last step of the handshake by using pw and the challenge data it received as nonce for input to PBDKF2 function to derive cryptographically secure keys which in turn will be used to encrypt the challenge data, a timestamp and the user phone number among other data as a response to the server [32] . By this step, the server guarantees that it is receiving a fresh handshake request from the intended client which is the only entity that has knowledge of pw.

After this handshake, the server sends in encrypted form, new challenge data to be used for the next time a client wants to authenticate itself with the server;

thus the client for the next session can simply derive the keys using this challenge data without a full handshake process, which expedites the process considerably and makes it even harder for eavesdroppers to decrypt data since the nonce was not sent in clear text like the normal handshake.

Table 3.3 shows the current security primitives used in WhatsApp and their key lengths.

Table 3.3: WhatsApp security primitives

Process	Algorithm	Key Length
Key derivation	PBKDF2	160
Symmetric encryption	RC4	160
Integrity protection	HMAC-SHA1	160

The reliance on pw which is at some point sent over the wire and is also known to the server, makes WhatsApp considerably less secure compared to Threema and TextSecure since the latter two rely on private keys known only to the user and are never transmitted over the wire at any point. WhatsApp also by the virtue of the previously mentioned key derivation scheme does not guarantee PFS, that is if an adversary gains access to pw by extracting it from a phone, they can decrypt in some cases future or even past traffic of the victim. Moreover, an adversary who can compromise the TLS connection on which pw is transmitted can eavesdrop on traffic of any future user who registers with the server.

We stated that it is possible only in some cases to decrypt past/future traffic because of the aforementioned method of authentication where a server sends the next session challenge data in encrypted form. Therefore, to decrypt traffic, an attacker must collect all WhatsApp traffic between a client and the server starting by the three way handshake; this way a chain can be constructed with the first session decrypted using the first challenge data sent in clear text and all subsequent sessions can be decrypted using the encrypted challenge data sent in the now decrypted previous sessions. Unfortunately, in our capture files we had a gap between the three way handshake and later capture files, so we could not predict the challenge data used in those sessions (one could theoretically brute force the challenge data). We could only decrypt the first session (setup traffic capture) with the server, and in Appendix A we demonstrate steps to repeat our process.

Originally WhatsApp did not provide end-to-end encryption but in late 2014 it announced that it is partnering with Open Whisper Systems to implement the Axolotl ratchet of TextSecure in WhatsApp [47]. As of the writing of this report[2], not all WhatsApp client versions support end-to-end encryption and the

[2]See note 1 on page 14.

development team as well as the user interface of the client do not convey to the users any information whether or not end-to-end encryption is in fact enabled on messages. Moreover, the current version of WhatsApp does not have any form of user authentication, so a user cannot be sure they are communicating with the intended recipient but must place trust in the server to authenticate the recipient which defeats the purpose of end-to-end encryption. Ultimately, we are pleased to see that WhatsApp is trying to improve its security, but we must also admit that their current effort for enabling end-to-end encryption is still lacking.

Chapter 4

Conclusion

The findings of our analysis can be summarized in Table 4.1.

Table 4.1: Analysis findings

	TextSecure	Threema	WeChat	WhatsApp
Transport Layer Encryption	Yes	Yes	Yes	Yes
End-to-End Encryption	Yes	Yes	No	Partial
PFS	Yes	Partial	No	No
Open Source	Yes	Partial	No	No

These findings illustrate the wide spectrum of the state of security in today's mobile messaging apps. We focus here on two interesting contrasts.

First, the approach to applying security in development. Both WhatsApp and WeChat developers deployed security as an after thought; they developed their apps and communication protocols and then little by little decided to add security measures to their solutions. This approach frees developers (in early stages of development) to use the simplest and most user-friendly features in their apps, but sacrifices security and is prone to errors in implementation. On the other hand, developers of Threema and TextSecure started with a security level in mind and tried their best to come up with a communication protocol that guarantees this level without making the app too difficult for an average user to use.

Second, the approach to embracing open source and public or standard protocols vs secret or proprietary protocols. TextSecure is the only app in this study that excels in this aspect. The next in line is Threema, although it is closed source, it uses open source implementations for security related algorithms and publicly publishes its approach to security. Third follows WhatsApp, it is closed source and does not proactively communicate its security philosophy, but it has recently made strides to get better by publishing some aspects of its provided security level and is collaborating with the makers of TextSecure to incorporate their

security algorithms in WhatsApp. Finally, WeChat is not only closed source but also doesn't publish its security approach or even acknowledge its need for better security like end-to-end encryption.

Our findings have also demonstrated that because app developers have complete control over the protocols including backend and frontend technologies, they tend to push updates to their architectures quite frequently. This means that research findings about the security of one app might be completely invalidated by the its newest update a very short period later. While this translates into better, more secure apps for the end users, researchers are spread thin to cover all the updates from the plethora of apps available in the market. For example, a large portion of the flaws we found in the literature has already been fixed in the current iteration of the apps we tested.

4.1 Future work

After we analyzed WhatsApp security, its developers completed the revamp of the security architecture and moved to end-to-end encryption, so it will be interesting to analyze their work, and see if they are successful in their effort and compare it to the offering of TextSecure app. Moreover, one aspect which we could explore even further is the analysis of WeChat protocol; future work might focus solely on the reverse engineering of the protocol using deep packet inspection.

4.2 Acknowledgments

We would like to thank Karpisek, Baggili and Breitinger for their valuable input in helping us decrypt WhatsApp traffic.

Appendix A

Decrypting WhatsApp Traffic

WhatsApp messages can be decrypted if the original traffic is captured and the WhatsApp password file is extracted from the device. In the following we explain how to do this in Wireshark:

install wirehsark if needed (tested version 1.12):

```
$ sudo apt-get install wireshark
```

install cmake if needed:

```
$ sudo apt-get install cmake
```

install required libraries:

```
$ sudo apt-get install libgcrypt20-dev
$ sudo apt-get install wireshark-dev
```

Download and compile the plugin:

```
$ wget https://github.com/davidgfnet/wireshark-whatsapp/archive/
072ce9b12cb85891370fb0e3f365e70869d60d97.zip
```

unzip the archive and cd to the folder. Type:

```
$ mkdir build
$ cmake ..
$ make
$ make install
```

To test it out, open Wireshark:

edit→preferences→protocols→whatsapp.

check enable packet decoding.

edit user/pass list with one or both of these details:

Mobile 1:

phone number:+4915731538084

password:yqE226nFjlnXLbA55Y7SfBj6IQw=

Mobile 2:

phone number:+4915162498970

password:aDCaW4et3QSQDEuX9fLtKSM+4Bw=

After this, WhatsApp messages should be denoted as WhatsApp protocol in Wireshark and the payload should be decrypted.

Note that this plugin can only work on a login procedure which includes the original challenge response between the server and WhatsApp client. In cases where the login uses previously transmitted challenge data to hasten the procedure, the plugin cannot predict the previous challenge data (used as salt) and therefore cannot decrypt it.

List of Figures

List of Tables

Bibliography

[1] I. Peter, "The History of Email," 2004. [Online]. Available: http://www.nethistory.info/History%20of%20the%20Internet/email.html

[2] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," Internet Requests for Comments, RFC Editor, RFC 3920, October 2004, http://www.rfc-editor.org/rfc/rfc3920.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3920.txt

[3] H. Tschofenig, *Improving Security on the Internet*. World Wide Web Consortium, 2014. [Online]. Available: https://www.w3.org/2014/strint/papers/62.pdf

[4] P. Dashtinejad, "Security System for Mobile Messaging Applications," Ph.D. dissertation, KTH Royal Institute of Technology, 2016.

[5] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," Internet Requests for Comments, RFC Editor, RFC 4880, November 2007, http://www.rfc-editor.org/rfc/rfc4880.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4880.txt

[6] B. Adida, S. Hohenberger, and R. L. Rivest, *Lightweight Encryption for Email*. USENIX Association, 2016. [Online]. Available: https://www.usenix.org/legacy/event/sruti05/tech/full_papers/adida/adida.pdf

[7] Q. Scheitle, M. Wachs, J. Zirngibl, and G. Carle, "Analyzing Locality of Mobile Messaging Traffic using the MATAdOR Framework," in *PAM*, 2016.

[8] "Whatsapp," 2016. [Online]. Available: https://www.whatsapp.com/

[9] "Threema - Seriously Secure Messaging," 2016. [Online]. Available: https://threema.ch/en

[10] "WeChat - Free Messaging and Calling App," 2016. [Online]. Available: http://www.wechat.com/en

[11] "Open Whisper Systems," 2016. [Online]. Available: https://whispersystems.org/

[12] "About Wireshark," 2016. [Online]. Available: https://www.wireshark.org/about.html

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://doi.acm.org/10.1145/357401.357402

[14] R. Nelson and J. Heimann, "SDNS Architecture and End-to-End Encryption," in *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '89. London, UK, UK: Springer-Verlag, 1990, pp. 356–366. [Online]. Available: http://dl.acm.org/citation.cfm?id=646754.704917

[15] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008, http://www.rfc-editor.org/rfc/rfc5246.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[16] H. Krawczyk, *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2005, ch. Perfect Forward Secrecy, pp. 457–458. [Online]. Available: http://dx.doi.org/10.1007/0-387-23483-7_298

[17] E. Rescorla, "Diffie-Hellman Key Agreement Method," Internet Requests for Comments, RFC Editor, RFC 2631, June 1999, http://www.rfc-editor.org/rfc/rfc2631.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2631.txt

[18] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 232–249.

[19] B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification," Internet Requests for Comments, RFC Editor, RFC 3851, July 2004, http://www.rfc-editor.org/rfc/rfc3851.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3851.txt

[20] N. Borisov, I. Goldberg, and E. 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.

[21] V. Moscaritolo, G. Belvin, and P. Zimmermann, *Silent Circle Instant Messaging Protocol Protocol Specification*, 1st ed. Silent Circle, 2012. [Online]. Available: https://web.archive.org/web/20150402122917/https://silentcircle.com/sites/default/themes/silentcircle/assets/downloads/SCIMP_paper.pdf

[22] M. Marlinspike, “Advanced Cryptographic Ratcheting,” 2013. [Online]. Available: https://whispersystems.org/blog/advanced-ratcheting/

[23] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” Internet Requests for Comments, RFC Editor, RFC 5280, May 2008, http://www.rfc-editor.org/rfc/rfc5280.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5280.txt

[24] “Certificate Authorities & Trust Hierarchies,” 2016. [Online]. Available: https://www.globalsign.com/en/ssl-information-center/what-are-certification-authorities-trust-hierarchies/

[25] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, “Rethinking SSL Development in an Appified World,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS ’13. New York, NY, USA: ACM, 2013, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516655

[26] S. Landau, “Making Sense from Snowden: What’s Significant in the NSA Surveillance Revelations,” *IEEE Security & Privacy*, vol. 11, no. 4, pp. 54–63, 2013.

[27] N. Unger, “Deniable Key Exchanges for Secure Messaging,” Ph.D. dissertation, University of Waterloo, Canada, 2015.

[28] R. Mueller, S. Schrittwieser, P. Fruehwirt, P. Kieseberg, and E. Weippl, “What’s new with WhatsApp & Co.? Revisiting the Security of Smartphone Messaging Applications,” in *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2014, pp. 142–151.

[29] C. Anglano, “Forensic Analysis of WhatsApp Messenger on Android Smartphones,” *Digital Investigation*, vol. 11, no. 3, pp. 201–213, 2014.

[30] A. Mahajan, M. S. Dahiya, and H. P. Sanghvi, “Forensic Analysis of Instant Messenger Applications on Android Devices,” *CoRR*, vol. abs/1304.4915, 2013. [Online]. Available: http://arxiv.org/abs/1304.4915

[31] A. Azfar, K.-K. R. Choo, and L. Liu, “Android Mobile VoIP apps: a Survey and Examination of Their Security and Privacy,” *Electronic Commerce Research*, vol. 16, no. 1, pp. 73–111, 2015.

[32] F. Karpisek, I. Baggili, and F. Breitinger, “WhatsApp Network Forensics: Decrypting and Understanding the WhatsApp Call Signaling Messages,” *Digital Investigation*, vol. 15, pp. 110–118, 2015.

[33] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How Secure is TextSecure?" *IACR Cryptology ePrint Archive*, vol. 2014, p. 904, 2014.

[34] S. Blake-Wilson and A. Menezes, *Public Key Cryptography: Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99 Kamakura, Japan, March 1–3, 1999 Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, ch. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol, pp. 154–170. [Online]. Available: http://dx.doi.org/10.1007/3-540-49162-7_12

[35] L. Onwuzurike and E. De Cristofaro, "Danger is my middle name: Experimenting with ssl vulnerabilities in android apps," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '15. New York, NY, USA: ACM, 2015, pp. 15:1–15:6. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/2766498.2766522

[36] M. Marlinspike, "Forward Secrecy for Asynchronous Messages," 2016. [Online]. Available: https://whispersystems.org/blog/asynchronous-security/

[37] J. Ahrens, "Threema Protocol Analysis," 2014, http://blog.jan-ahrens.eu/files/threema-protocol-analysis.pdf.

[38] T. GmbH, "Threema Cryptography Whitepaper," Threema GmbH, Tech. Rep., 2015. [Online]. Available: https://threema.ch/press-files/cryptography_whitepaper.pdf

[39] H. Dimitrov, G. Pineda, and J. Laan, "Threema Security Assessment," 2013, https://www.os3.nl/_media/2013-2014/courses/ssn/projects/threema_report.pdf.

[40] D. J. Bernstein, T. Lange, and P. Schwabe, *Progress in Cryptology – LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. The Security Impact of a New Cryptographic Library, pp. 159–176. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33481-8_9

[41] cnlab security AG, "Security Review Threema: Security Statement," 2015, https://threema.ch/press-files/2_documentation/external_audit_security_statement.pdf.

[42] S. Millward, "WeChat Continues Growth, Hits 650 Million Users," 2015. [Online]. Available: https://www.techinasia.com/wechat-650-million-monthly-active-users

[43] F. Gao and Y. Zhang, "Analysis of wechat on iphone," in *2nd International Symposium on Computer, Communication, Control and Automation, Atlantis Press.*, vol. 69, 2013.

[44] R. Paleari, "A Look at WeChat Security," 2013. [Online]. Available: http://blog.emaze.net/2013/09/a-look-at-wechat-security.html

[45] "One Billion," 2016. [Online]. Available: https://blog.whatsapp.com/616/One-billion

[46] R. Dillet, "Bye Bye, WhatsApp: Germans Switch To Threema For Privacy Reasons," 2016. [Online]. Available: http://techcrunch.com/2014/02/21/bye-bye-whatsapp-germans-switch-to-threema-for-privacy-reasons/

[47] M. Marlinspike, "Open Whisper Systems Partners With WhatsApp to Provide End-to-End Encryption," 2016. [Online]. Available: https://whispersystems.org/blog/whatsapp/

[48] C. Rottermanner, P. Kieseberg, M. Huber, M. Schmiedecker, and S. Schrittwieser, "Privacy and data protection in smartphone messengers," 2015, https://www.sba-research.org/wp-content/uploads/publications/paper_drafthp.pdf.

[49] "The PHP WhatsApp Library," 2016. [Online]. Available: https://github.com/mgp25/Chat-API

[50] "The Python WhatsApp Library," 2016. [Online]. Available: https://github.com/tgalal/yowsup

[51] R. Eikenberg, "WhatsApp versendet keinen Klartext mehr," 2012. [Online]. Available: http://www.heise.de/security/meldung/WhatsApp-versendet-keinen-Klartext-mehr-1673054.html

[52] F. Balducci, "WhatsApp is Broken, Really Broken," 2012. [Online]. Available: https://balau82.wordpress.com/2012/09/19/whatsapp-is-broken-really-broken-fileperms/