Threema security assessment

Research project for Security of Systems and Networks
Master System and Network Engineering

Hristo Dimitrov, Jan Laan, Guido Pineda

December 22, 2013

Abstract

The research described in this paper aims to find out if there are any security flaws in the mobile messaging application Threema. In order to do this the memory of the application was analyzed, as well as the files stored on the mobile device. Lastly the network traffic generated by Threema was inspected, and attempts were made to perform a man-in-the-middle attack. No serious security flaws were found in this research, as long as a master key is used for added security, this is a feature of Threema itself. The conclusion is that Threema looks safe and well built.

Contents

1 Introduction

Messaging applications are constantly used nowadays, and one big concern is whether they are secure or not. Big security flaws have been discovered in widely used messaging applications like Whatsapp or BlackBerry Messenger [9] [8], where communications can be eavesdropped by an attacker. The Threema application claims to be secure by using true end-to-end encryption, where even the server operator has no access to read the user's messages. Widely used applications such as Whatsapp, BlackBerry Messenger, have claimed that they are secure, but seriously security flaws have been found, such as unencrypted channels. In this research, we want to test the current state of the security of the application, and test if it is as secure as they claim it to be.

1.1 Research questions

The main research question for this project is: What is the current observable state of the Threema application?
The main question can be split into the following subquestions:

- Is there any readable data left on the mobile's device flash drive?
- Is Threema susceptible to Man in the Middle Attacks?
- Does Threema employ Perfect Forward Secrecy?
- Is there any reachable data left in the mobile's RAM?
- Are there no obvious security flaws in the application source code? e.g.: Wrong use of (cryptographic) libraries.

1.2 Threema

The Threema application is available both for Android and iOS. In this research project we focused on Android mainly because we have some experience developing Android applications, and specifically decompiling the application would be easier for us, as well as reading the decompiled code.

The Threema application is a classical mobile messaging app, that can be downloaded at Google's Play Store for Android devices, or App Store for iOS devices. To start using it, the user has to create his own private key, randomness is created by moving the finger on the display. The app can be linked to the user's email account and his phone number.

Every contact that uses the Threema app in the contact list will appear as a contact in the user's app. The verification level of each contact is represented by three dots, that indicate the degree of confidence that a stored public key really belongs to the contact. This are the levels of confidence that the application uses:

- One dot colored (red): The ID and the public key were delivered by the server, there is no match with the address book and the user can't be sure that the person is who it claims to be.

- Two dots colored (orange): The verified email or the phone number of the contact was found in the user's address book, the user can have some confidence that the person is who he claims to be.
- Three dots colored (green): The ID and the public key were checked by scanning the contact's QR code. Unless the device has been hacked, the user can be sure that the person is who it claims to be.

Contacts can exchange messages with eachother. Messages can contain text, images or locations. Threema also allows for messages to be acknowledged, a way of showing the other user that the message has been read and understood without having to send a reply. The iOS version of Threema has a group chat function, the Android version which is the focus of this research does not.

1.2.1 Security used

Threema uses asymmetric cryptography to protect the communication between two users and the communication between the application and the servers.

The specific type of encryption that is used is Elliptic Curve Cryptography, which has a strength of 255 bits. A 128 bit message authentication code (MAC) is also added to each message to detect manipulation/forgeries.

The app also adds a random amount of padding to each message to thwart attempts to guess the content of a message by looking the amount of data.

It has two layers of encryption, the end-to-end layer between two users, and an additional layer to protect against eavesdropping of the connection between the app and the servers.

Message encryption and decryption is made on the device, and the user has control of the key exchange. This guarantees that not a third party, not even the servers, can decrypt the captured messages.

2 Approach

The approach to this project is mainly trying to answer the research questions proposed in the Introduction. In order to do this, we are going to do the following:

- Decompile the downloaded app for an Android device
- Analyze the code of the application to see how it works
- Try to read any data left in the memory of the device
- Set up a test environment to capture network traffic and analyze any vulnerability

3 Implementation

3.1 Decompiling

To be able to research the application, it was decompiled, using the tools dex2jar [3] and jd-gui [4]. This enabled us to read the source code of the application,

albeit without any symbolic information. All package, class and variable names were lost, making the source code harder to read.

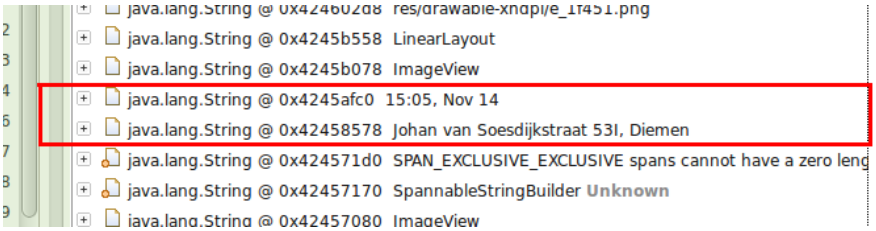
Then, the application was recompiled with debugging symbols, using apktool [1]. The source code was loaded into the Netbeans IDE so breakpoints could be set in the code, in order to further analyze and understand it.

3.2 Memory analysis

The Android development environment [2] for Eclipse and the Eclipse plugin MAT [5] were used to analyze the memory of the Threema application. The memory of the application was dumped while it was running, and then loaded into MAT. Using MAT, the application memory could be browsed.

Inspecting the memory quickly showed received and sent messages. An example can be seen in figure 1. This shows the date of the message, directly followed by the contents of the message. This specific format and layout makes it easy to identify messages in the memory dump.

Figure 1: Threema memory snippet



Because it is hard tfor an attacker to access a user's phone memory with the Threema application active, not much time was spent analyzing the memory. Though, if an attacker does get access, it will be easy to read the messages.

3.3 Filesystem

The Threema application has many files in its database directory. Some are of particular interest:

- databases/threema.db
- files/key.dat
- shared_prefs/ch.threema.app_preferences.xml

Our first assumption was that the file `threema.db` should hold some interesting data. Maybe the messages, contacts, or even the private key. It was encrypted, so not readable immediately. `key.dat` was a small binary file, some sort of key, to access what? `ch.threema.app_preferences.xml` was a plaintext xml file.

3.3.1 Preferences

The preferences file showed some interesting, but not neccesarily sensitive information. It holds amongst others:

- The user's Threema ID
- The user's public key
- The user's linked email address (when available, linking is optional)
- The user's linked phone number (when available, linking is optional)
- The user's public nickname

3.3.2 Database

The decompiled source of Threema showed use of a library called SQLCipher [10]. SQLCipher is an open source library, which protects SQLite databases with AES-256 encryption. SQLCipher is a popular solution for securing databases on iOS, Android and other platforms, used by many commercial applications.

Threema uses this as well. This is what Threema has to say about it on their website [11]:

> Threema includes its own app-specific encryption based on AES-256 to protect stored messages, media and your ID's private key. The key used for this encryption is generated randomly the first time you start Threema, and can optionally be protected by setting a Master Key Passphrase in the settings, which we highly recommend. Without a passphrase, the encryption will only add obscurity due to the way hardware encryption is handled on Android.

We wanted to verify this claim. After inspecting the source code we were able to determine that the file `key.dat` held the key used to encrypt the database. However, the file could not be used directly. Threema added some obfuscation to make it harder to open the database. We created a small java program, that given the keyfile would output a key in the format that SQLCipher would accept. The program can be found in appendix A. SQLCipher can accept two types of keys:

1. A passphrase, from which a key is derived
2. a 265-bit raw key input, formatted as hex string

Threema uses the latter of the two. For example, the key of our test database, which is randomly generated on database creation, was, once converted,

```
x"96e7d71689c8c79a7a546c69b8c338a165d9c9c61f10d70d6d45dc7748eead31"
```

We then opened the database in the command-line verison of SQLCipher, entered the key, and the database was decrypted.

Table: contacts

	identity	publicKey	firstName	lastName	verificationLevel	androidContactId	threemaAndroidCont
1	P38E74DM	❓!❓ll9❓F❓❓	Jan	Laan	1	2005i37eb285a0c	2005i37eb285a0c3c19
2	VBRF7YDV	❓C❓_❓❓❓			2		3096r283-5129493120

(a) Contacts

Table: message

New Record | Delete Record

	id	apiMessageId	identity	outbox	type	body	isRead	isSaved	state	postedAt	createdAt	modifiedAt
1	1	0ff9d0501d86	VBRF7YDV	1	0	Hi Guido	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
2	2	20b6e7ffb305	VBRF7YDV	1	0	Dbdhfhfhfhf	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
3	3	e22c451af1de	VBRF7YDV	1	0	Dgdgs	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
4	4	056f2ce8dec6	VBRF7YDV	1	0	Emmen	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
5	5	60a9d5935d4a	VBRF7YDV	1	0	@	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
6	6	065ecef6dc3d	VBRF7YDV	0	0	Test	1	1		2013-11-25 15	2013-11-25 15	
7	7	5b7ac864b720	VBRF7YDV	1	0	hahaha	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15
8	8	e02ff30ab7bc	VBRF7YDV	1	0	Cgj	0	1	READ	2013-11-25 15	2013-11-25 15	2013-11-25 15

(b) Messages

Figure 2: Threema database

The database has a few tables. **contacts** holds a list of all contacts, and **message** contains a list of all sent and received messages. This shows the validity of the statement on the Threema site. With just the Threema data files, it is possible to retrieve all messages and contacts, which is certainly not secure. When a master key passphrase was added to the application, the above process was not possible anymore.

3.4 Network traffic

3.4.1 Environment Set Up

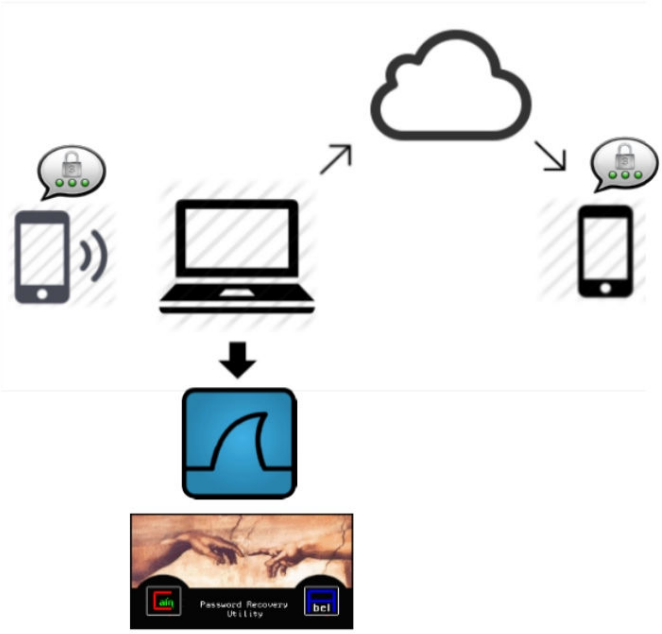
For the purpose of investigating the network traffic that the Threema app generates and performing attacks on it, a special set up needed to be implemented. The set up should allow for:

- Internet connectivity for the Threema app
- Traffic sniffing and inspection
- Traffic manipulation for the purpose of communication attacks

Since there are different tools for performing attacks on a network traffic and some of them work on Windows only, others on Linux only, we came up with two set ups, which will allow us to use both kinds of tools.

The first set up is a Windows set up. It uses a rooted Android Phone which runs the Threema app and DroidWall which was used to block all other traffic from the phone. The phone connects to a Windows laptop which acts as a WiFi Hotspot. The laptop runs the MyPublicWiFi software for setting up the WiFi Hotspot [6]. It also runs Wireshark for traffic sniffing and Cain and Abel for attacks on the traffic [7]. The Ethernet network card of the laptop is used for connecting to the Internet.

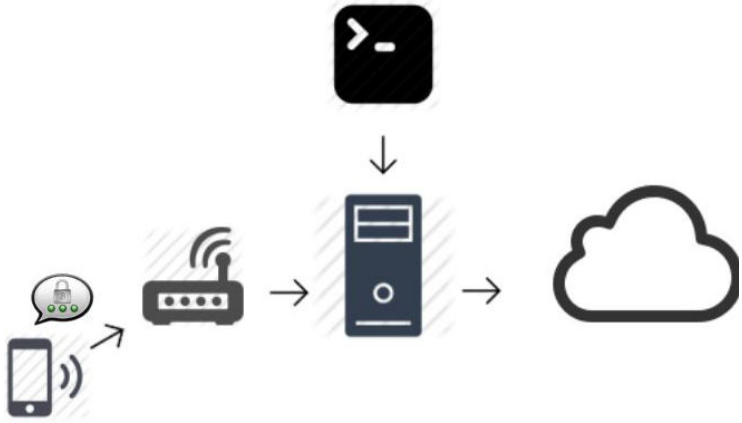
Figure 3: This is the Windows set up used for packet capturing and MitM attacks



Since we did not have a Linux laptop in our possession, the Linux set up is quite different. Again as in the Windows set up a rooted Android Phone which runs the Threema app and DroidWall was used. The phone connects to an Access Point which is connected to a Ubuntu server via Ethernet cable. The Access Point is configured to act as a repeater only, with as default gateway the IP address of the server. The server runs SSLSplit for performing a Man-in-the-Middle attack. It has two Ethernet network cards and the second one is used to connect to the Internet. Also IP forwarding is enabled on the server and the following iptables rules were added so the traffic can be successfully routed from the one network interface to the other:

```
iptables -t nat -A POSTROUTING -o em1 -j MASQUERADE
iptables -A FORWARD -i em2 -o em1 -j ACCEPT
iptables -A FORWARD -i em1 -o em2 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Figure 4: This is the Ubuntu Set Up used for MitM attacks



3.4.2 Network Communication

The Windows set up was used for observing the network traffic. The network packets from the following scenarios were captured using Whireshark:

- Starting the app
- Unlocking the master key
- Adding contacts and verifying the status of the added contacts
- Sending messages to a contact which is online
- Sending messages to a contact which is offline and becomes online after some period of time
- Changing profile details

All of the scenarios were performed multiple times. The following observations were made from the captured traffic.

1. The Threema app communicates to two different servers. The first one is used to communicate Meta data and status updates. For example when the user changes his name, this is the server that the change will be communicated to.
2. For the communication it uses the Hypertext Transfer Protocol Secure (HTTPS) which implements Secure Sockets Layer (SSL) connection.
3. In order to connect to that server the Threema app issues a DNS query for "api.threema.ch", then fetches the IP address of the server from the response and starts an SSL session with that server. A Man-in-the-Middle attack seams feasible for that connection.
4. The second server is only used for forwarding messages to other users.

5. The communication protocol used for this communication is Extensible Messaging and Presence Protocol (XMPP). XMPP implements Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS) for its security. This means that MitM attack might be feasible for this communication.

6. In order to connect to the second server the Threema app issues a DNS query for "g-¡2 digit number¿.0.threema.nl" (where the 2 digit number is filled dynamically in the Threema code and no matter what that number is, the response IP address is always the same), then fetches the IP address of the server from the response and starts an XMPP session with that server.

7. Every message send in this session is amended with random number bytes, which results in variable and non-predictive packet sizes. Therefore, sending the same message resulted in very different traffic captured by Wireshark.

There are three interesting characteristics that Threema claims to provide when users are chatting. The first one is that if a recipient is offline, the messed will be sent from the sender to the server and the server will store it until the recipient comes online. The second one is that the messages are being delivered with an end-to-end encryption. This means that there is a layer of encryption which happens at the sender and is being decrypted only after the message reaches the receiver. And the third one is that Threema ensures perfect forward secrecy. Which means that session keys are being established for sending the messages over the wire, which are being forgotten after the session finishes, so that if packets are captured from an attacker he will not be able to decrypt them in the future by getting his hands on the private keys of the users.

The first one was easily verified, simply by sending a message to a user that is offline, checking for network traffic from then sender to the server, then bringing the receiver online and verifying that he receives the message.

However for the other two to be fulfilled, there should be at least two layers of encryption. One layer of encryption: from the sender to the recipient, which will provide the end-to-end encryption. This layer should use public key cryptography since there is no way to negotiate a session key with the recipient if he is offline. The second layer of encryption will ensure the perfect forward secrecy. This layer should use session keys for establishing 2 sessions: one from the sender to the server and one from the server to the recipient. If this is the way this communication is implemented, an attack against it will be harder to perform.

However Threema does establish session keys, which stay valid for up to one week. If those are used for the user to user communication, there will be no longer need for two-layer encryption, however the implementation of perfect forward secrecy becomes questionable, since one week is a long period for a session key to be valid. Because the part of the code, which deals with this part of the communication is too obfuscated after the decompilation, we couldn't figure out what is the exact implementation for this part of the communication. We also did not find a good tool for MitM attacks against XMPP sessions, so no attack against the communication with the second server was performed in this research.

3.4.3 Man-in-the-Middle

The SSL connection with the first server will be the target of the Man-in-the-Middle attack. There are a couple of variants of how this attack can be performed, based on the certificate that the attacker uses for impersonating the server.

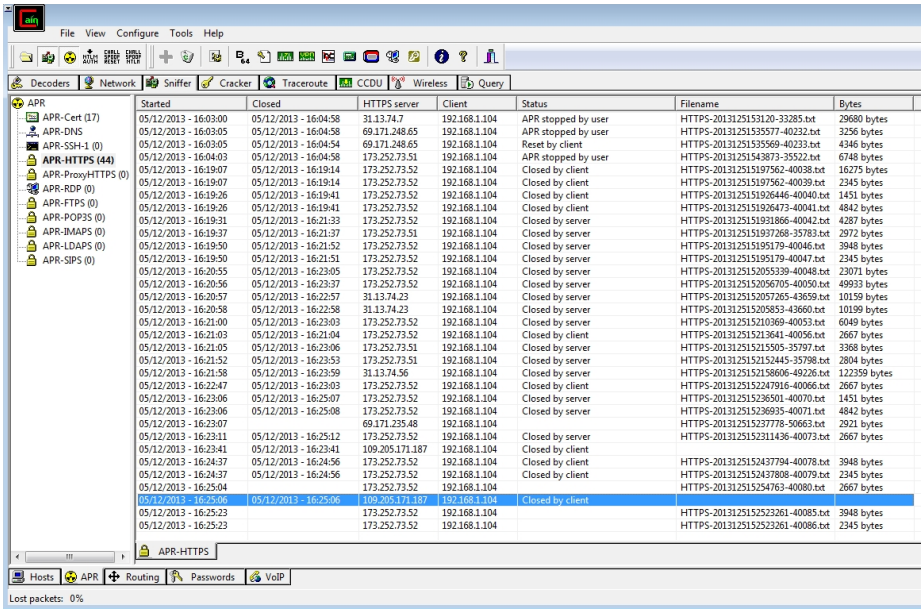
1. Using a fake certificate which is not signed by a Certificate Authority (CA), but has the same parameters as the real one
2. Using a fake certificate which is signed by a fake Certificate Authority (CA) which is not trusted by the user, but both have the same parameters as the real one
3. Using a fake certificate which is signed by a fake Certificate Authority (CA) which is trusted by the user and both have the same parameters as the real one
4. Using a real certificate which is signed by a real Certificate Authority (CA), but has different parameters

In this research only the first three variants were performed, because we did not have the private key for any real certificate. The first and second variant of MitM attack was performed on the Windows set up using the Cain and Abel software tool. There were no results from the attack on the targeted link. Then the created fake CA certificate was installed to the user's phone in order to test the third variant of the attack. However this variant is not very feasible, because one has to have access to the user's device in order to install a CA on it.

The second and the third variant of the attack were then performed on the Ubuntu set up using SSLSplit. Again there were no results from the attack. Actually the device did not even try to establish an SSL session for a second time, which is probably a security feature of the app, to not attempt to connect to the server for a second time after detecting a MitM attack.

Finally the third variant of the attack was tested on the Windows set up. This time there was a visible result in the Cain and Abel GUI. The attempt for establishing the session was displayed, however the client dropped the connection immediately, so no data was captured from that connection (see Figure 5). It was not exactly clear why the client dropped the connection, but we assumed that it is most likely, because the certificate used by Threema was pinned. It can be seen that the same set up performed a successful MitM attack on almost all of the rest of the HTTPS traffic. In fact the only other SSL session that got closed from the client in the same way was the one from the banking app from ABN Ambro. In conclusion, there must have been some additional check, like a hardcoded certificate, that only the most secure apps perform, which makes the MitM attack infeasible, or at least harder.

Figure 5: Man-in-the-Middle Attack - The client closes the connection before any data is transmitted



4 Conclusion

We could not find any serious security flaws in the Threema application. However, we do encourage the use of a Master password to encrypt the Threema database, as this enhances security. The application uses established crypto libraries, such as SQLCipher and NaCl (Salt), which are well known and open source libraries to do encryption. The application has some levels of verification for users by using email addresses, phone numbers and public key exchange by QR codes, which prevents impersonation. In order to get access to the users messages, it is only possible by having physical accesses to the phone, which in a practical scenario is not very feasible. Even with access to the device, access to Threema messages and contacts is very limited. Threema looks secure and well implemented.

5 Further research

In this research, different angles of the Threema application have been explored. However, there are more things left unexplored. The focus was on the Android app, while there is also an iOS version, which is similar, but not the same. For example the database encryption in iOS is different from Android [11], and iOS supports a group-chat feature with the Android version does not have.

The decompiled code has not been extensively looked at, only for the extent needed for the topics discussed above. This can be looked at to verifiy if there are any implementation errors.

The strength of the used encryption was not validated, however due to the use of standard cryptographic libraries, this is an unlikely vector of attack.

References

[1] android-apktool. https://code.google.com/p/android-apktool/.

[2] Android sdk. https://developer.android.com/sdk/index.html.

[3] dex2jar. http://code.google.com/p/dex2jar/.

[4] jd-gui. http://jd.benow.ca/.

[5] Eclipse memory analyzer. http://www.vogella.com/articles/EclipseMemoryAnalyzer/article.html.

[6] Mypublicwifi. http://www.mypublicwifi.com/publicwifi/en/index.html.

[7] Apr-https in cain and abel. http://www.oxid.it/ca_um/topics/apr-https.htm.

[8] Neha s. Thakur. Forensic analysis of whatsapp on android smartphones. *University of New Orleans Theses and Dissertations. Paper 1706.*, June 2013.

[9] Peter Kieseberg Sebastian Schrittwieser, Peter Frhwirt. Guess who's texting you? evaluating the security of smartphone messaging applications. *19th Annual Network & Distributed System Security Symposium*, February 2012.

[10] Sqlcipher. http://sqlcipher.net.

[11] Threema. http://www.threema.ch.

A Threema database key retrieval

Usage: `java ThreemaDecrypter key.dat`

Class "p.a" is extracted from the Threema source code. It is the class that deals with reading the keyfile.

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.util.Locale;
4  import p.a;
5
6  class ThreemaDecrypter
7  {
8    public static void main(String args[])
9    {
10     try
11     {
12       File f = new File(args[0]);
13       a decrypter = new p.a(f);
14       byte[] databasekey = decrypter.c();
15
16       String realdbkey = "x\"" + aaa(databasekey) + "\"";
17       System.out.println(realdbkey);
18     }
19     catch(Exception e)
20     {
21       System.out.println("Exception: "+ e.getMessage());
22     }
23   }
24
25  public static String aaa(byte[] paramArrayOfByte)
26   {
27     char[] arrayOfChar1 = { 48, 49, 50, 51, 52, 53, 54,
         55, 56, 57, 97, 98, 99, 100, 101, 102 };
28     char[] arrayOfChar2 = new char[2 * paramArrayOfByte.
         length];
29     for (int i = 0; i < paramArrayOfByte.length; i++)
30     {
31       int j = 0xFF & paramArrayOfByte[i];
32       arrayOfChar2[(i * 2)] = arrayOfChar1[(j >>> 4)];
33       arrayOfChar2[(1 + i * 2)] = arrayOfChar1[(j & 0xF)
           ];
34     }
35     return new String(arrayOfChar2);
36   }
37 }
```

B Contribution

Hristo Dimitrov	Network environment setup, network/traffic analysis, report
Jan Laan	Decompilation, code analysis, memory analysis, application data analysis, report, presentation
Guido Pineda	Network environment setup, network/traffic analysis, report