Fakultät für
Informatik und
Mathematik

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

TÉCNICO
LISBOA

Implementation of a secure TODO–App on Android

Presentation in class Mobile Application

class Mobile Application in the study course computer science

submitted on 12.06.2020

Submitted from: João Pedro Moreira Sousa, moreira.joao@hm.edu
Michael Fuchs, michael.fuchs@hm.edu

Examiner: Prof. Dr. Gudrun Socher, gudrun.socher@hm.edu

Contents

1 Motivation

The number of clients on the Internet is constantly increasing. Mobile devices such as smartphones and tablets are a major driver of these trends. The sheer number of these devices also arouses the interest of potential attackers. The average user is usually not very concerned about the security of mobile apps and therefore they are a popular target. So it is up to the developer to be aware of potential attacks and their effects. Starting with a simple TODO–App with backend, security measures are iteratively implemented to store and to backup data in an untrusted environment.

2 Preliminary considerations

2.1 Mobility aspects

In the following, the security aspects of mobile devices are considered in order to provide a basis for planning security mechanisms. The devices are often taken along by their users, which means that they are rarely left unattended. However, mobility also causes problems, which will be discussed further below.

2.1.1 Need for communication

In order to achieve this mobility, a compromise in terms of performance is required. This means that all resources (CPU, memory, battery) are clearly limited. For this reason, computationally intensive operations, such as cryptographic operations, cannot be carried out for long because the battery would be drained very quickly. To reduce the computing effort on the mobile device, a remote server is usually used. On this server the operations can be performed quickly. The mobile device can display the calculated data in a resource–saving way. Therefore the application requires a permanent connection to the server.

2.1.2 Need for stored data

However, the mobility of the devices also means that a connection to the server is not permanently guaranteed. To be able to provide the functions of the application even without a connection, the application needs the necessary data locally. This means that any safety–relevant data can be found on the device. The question therefore arises as to how this data is to be secured.

2.2 Safety aspects

As already mentioned, mobile devices have become an integral part of modern life. They collect extensive data from users and enable various applications to handle this data. From mobile banking to classic password storage, all data is concentrated in one device. In the following, the question of how communication from server to client can be secured is clarified.

2.2.1 Communication

The basic goal of an attacker is to read or manipulate the communication between two devices. Mobile devices are connected to a WiFi and the user has no way to verify the routing or router without technical knowledge. This situation is susceptible to Man–in–the–middle attacks as shown in figure 1. Man–in–the–middle attacks are, as the name implies, situation in which the attacker intercepts a message in the middle of its transportation and changes its content. In the example below Eve can therefore provide a network, impersonate a known router and read the messages.

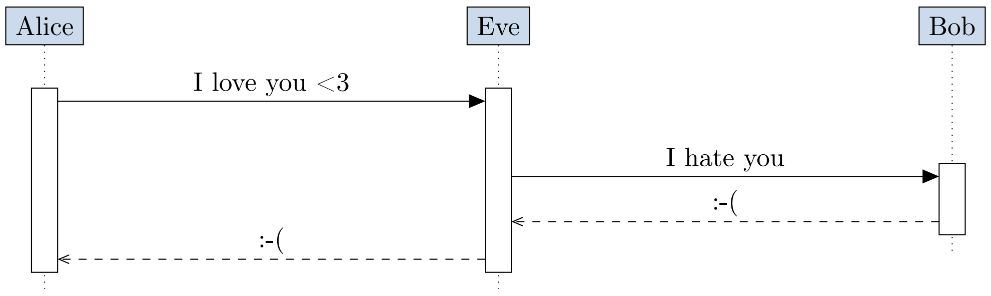

Figure 1 | Modification of messages by a Man–in–the–Middle attack.

In the internet, most data is transferred via HTTP [12], no matter if REST, GraphQL or SOAP interface. In addition, transport encryption (TLS) offers the possibility to transfer messages encrypted between the communication partners [16]. Here the Diffie–Hellman key exchange is performed according to the well–known TCP handshake (figure 2) to derive the symmetric session key from the asymmetric cryptosystem. A basic assumption of this procedure of this connection setup is that the initiator of the communication knows and can authenticate his partner. For this reason, the certificate for authenticating the client to the server is a central point of attack [15].

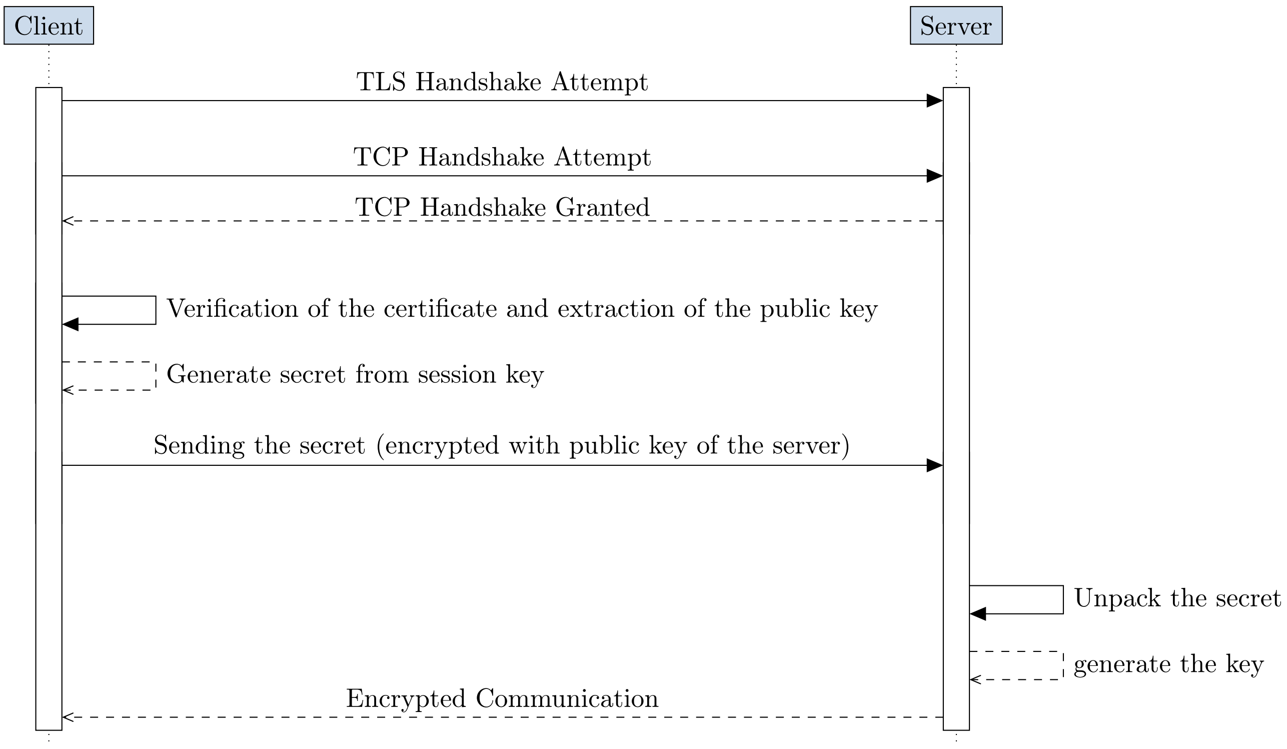
Client
Server
TLS Handshake Attempt
TCP Handshake Attempt
TCP Handshake Granted
Verification of the certificate and extraction of the public key
Generate secret from session key
Sending the secret (encrypted with public key of the server)
Unpack the secret
generate the key
Encrypted Communication

Figure 2 | Connection setup with TLS.

Communication partners cannot uniquely authenticate themselves **Solution**: Certificate pinning by comparing the stored certificate with the server's certificate [1] and is illustrated in figure 3.

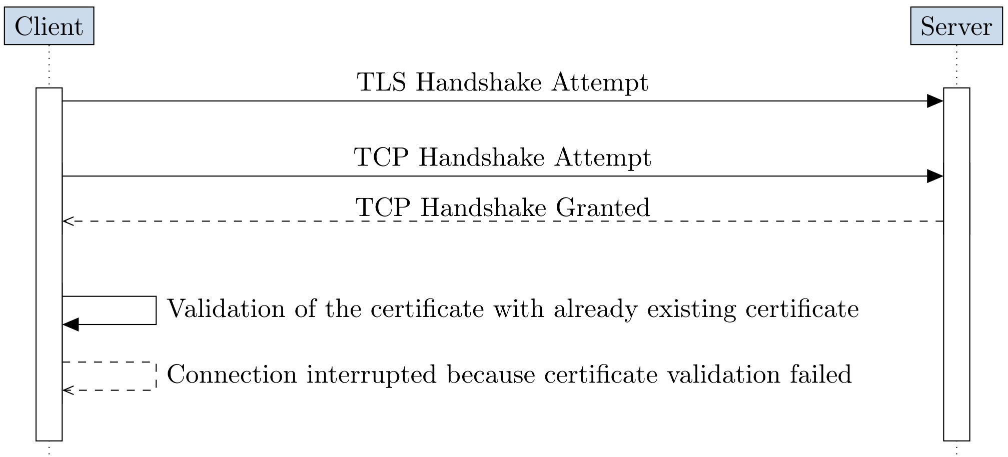
Client
Server
TLS Handshake Attempt
TCP Handshake Attempt
TCP Handshake Granted
Validation of the certificate with already existing certificate
Connection interrupted because certificate validation failed

Figure 3 | Procedure of a Man–in–the–Middle attack with Certificate Pinning.

2.2.2 Data Storage

As has already been noted, a permanent connection to the Internet cannot be assumed for a mobile device. In order not to lose all functions during this time, data must be kept on the device. The persistence of the data on the mobile device opens up further attack vectors for an attacker. This can only be prevented by effective encryption of the data. Basically Android supports and forces that the devices are delivered with an encrypted hard disk. Three different scenarios are to be considered as attackers:

- the device is switched off
- the device is switched on but in locked state
- the device is switched on and enabled by the user

When the device is turned off, the data is decrypted only at startup and there is no data in volatile memory. Only the attack can be started via brute force, but this is made even more difficult because the password is a combination of the user's password and a non-extractable hardware secret. The other two scenarios behave differently because the data is already decrypted, i. e. it is present in the file system without being encrypted. The system is not aware at this point that it is not being used by a legitimate user. The lock screen is just another hurdle, but not sufficient protection. However, secure software consists of layers of security mechanisms. One way to further increase the security of data is to have the data stored encrypted by the application.

3 Scenario

The goal of the scenario is to provide the largest possible cross section of all security mechanisms of Android, while still being able to evaluate the individual points. The application focuses on storing sensitive notes. The stored data should be protected against unauthorized access both locally and during transmission. The encrypted connection is secured and certificate pinning is built in. Finally, TODO–App is signed in order to be able to distribute it via the Google PlayStore. The security of the application is given the highest priority and therefore it should always be noted that the user group is not necessarily informed about the security of the software. For this reason, the control mechanisms should be planned so that the user cannot influence them.

3.1 Architecture

This presentation focuses on the secure development of the mobile application. Here, the notes are first stored locally in a database and, if necessary, a backup is made using serverless computing. The backend[1] is only a mock backend in order not to lose the focus on the actual implementation of the mobile application[2]. Here vercel is used as a service provider. Each note is encrypted by the same symmetric key, which is derived from the user's biometric fingerprint. Here, AES is used in Galios/Counter mode without padding as an encryption function to enable high data throughput with the option of parallel data streams [11].

3.2 Threat modelling

Threat modelling is a technique that lists and analyses external threats. A major problem with software projects is that control measures may be implemented in the wrong places, wasting development time and money without adding any security value. In contrast, a threat analysis should show what is worth protecting in the software, what threats there are, against which threats one wants to protect oneself and what control mechanisms are implemented for the individual threats [14, pp. 10–18]. In the following, threat modeling is applied to the application in order to analyze possible attacks: The first step of the threat analysis is to identify the assets and for the TODO-App the following points should be mentioned.

- the stored data of the users
- the certificate for signing the application
- the certificate of the server (not subject of this presentation)

The next step is to define the attack surface more precisely. The user data is generated by a smartphone and stored there or synchronized with the backend via the Internet. Finally, the possible attack vectors are mentioned in the illustration figure 3.

[1]can be found at `https://github.com/theexiile1305/ma-sp-security-backend`
[2]can be found at `https://github.com/theexiile1305/ma_sp_security`

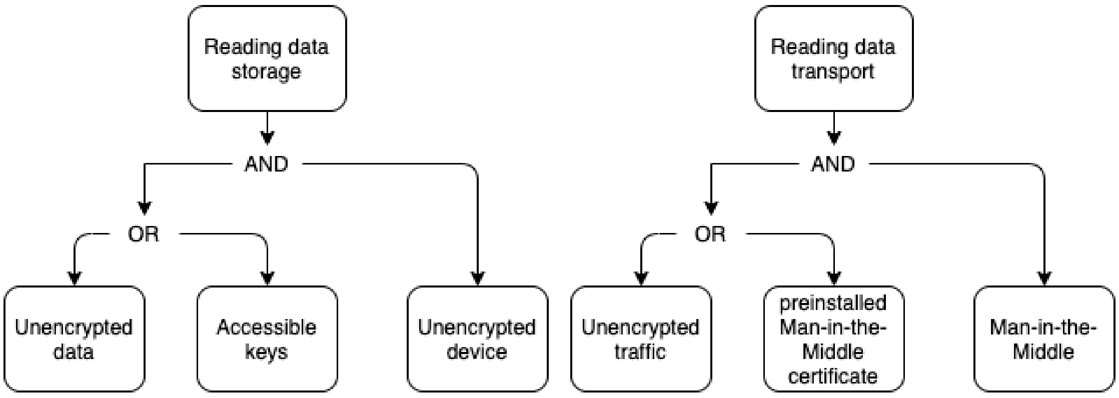

Figure 3 | Attack tree for communication and data storage.

3.3 | Implementation

3.3.1 | Data storage

In this section, mechanisms to increase the IT security of the TODO–App are implemented iteratively. First, all data is encrypted at the application level in addition to the device encryption, and a distinction must be made between different procedures: Either the keys are derived from a password or the keys are generated initially with high entropy. Furthermore, the generation of a key requires the encrypted storage of the key on a back end, so that access is not restricted to the source device. In principle, the key may only be stored in a secure hardware module, so that the contents of the key cannot be extracted even if the entire external system would be compromised. In the example implementation, the symmetric key used for encrypting and decrypting the data is derived from the biometric fingerprint of the user. This approach requires the KeyStore (Android API level 18), the AndroidKeyStore (Android API level 23), which provides symmetric encryption methods, and the StrongBox (Android API level 28), which is the name of secure hardware modules in Android [7, 3, 2].

3.3.2 | Communication

Subsequently, the transport encryption of the data should be ensured when backing up the TODOs. In Android, forcing HTTPS connections from API level 24 is very easy. For this purpose, a Network Security Configuration must be entered in the manifest of the application [8].

3.3.3 Certificate Pinning

Although the connection is secured after the first contact, it is, as already shown, quite possible to compromise the connection as soon as communication is established. Therefore, a technique called Certificate Pinning is suitable. In this case, the certificate delivered by the server is not verified by its signatures, but it is compared with a certificate delivered with the application. The simplest variant is to check the certificate for equality. This test is easy to implement, but on the other hand not very flexible. If, for example, the server's certificate changes because it has expired, the application must be updated. The second variant is not to check the entire certificate, but only its public key for equality. If the server's certificate expires, a new certificate with the same key pair can simply be created and the application can continue running unchanged. A disadvantage of this variant, however, is that the creation of new certificates becomes more complex, since the key pair must remain fixed [13, 8, 9]. By pinning certificates in any form, the security of the communication is only dependent on the server and the security of its certificate. It must therefore be ensured that these do not leave the server. However, this is not the subject of this presentation [8, 9].

3.3.4 Signing

In the further course of the app's development it will be released in the official stores. In the case of Android this is Google Play and requires a signature of the application. Furthermore, the app is scanned before release, which further increases security on the user side. For example, Google states that devices that get their applications only from Google Play are much less vulnerable to potentially malicious applications [10].

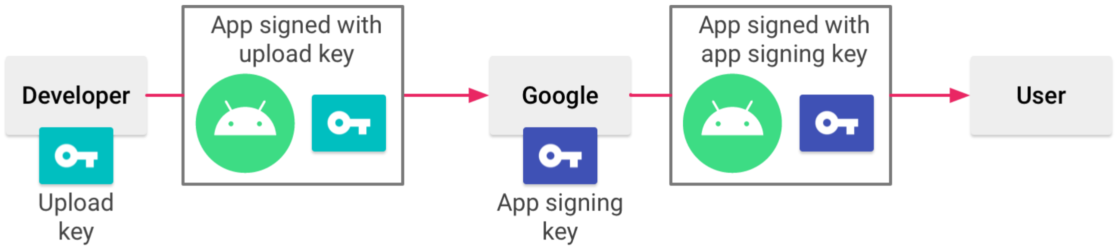

Figure 4 | Scheme for signing Android applications [10].

Under Android there are two ways to sign an application as shown in figure 4. On the one hand, there is the possibility for developers to create a signature locally themselves. This is done automatically during development, for example. There the application is signed with a debug certificate that is generated by the development environment beforehand. This variant can also be used in the same way, with a release certificate, to publish the application [4, 10].

In the second variant, two certificates are generated, one by the developer locally and one at Google. The developer certificate is used to sign the application for upload to Google. Google verifies the signature and then signs the application again with the certificate located at Google [4, 10].

3.3.5 Keyboard

Another point of attack is the auto-completion of the keyboard. This saves parts of the input to further improve the prediction. This prediction is not only valid for the programmed application, but also for other applications of the system. Under Android the auto-completion can be ensured by an attribute of the input field. This does not completely solve the problem of the keyboard cache under Android. The user is always free to install his own keyboards. If this installation is compromised or if the keyboard itself has already been compromised by an attacker, the user can write down all input completely. In order to further limit the attack surface, it may be better to provide a keyboard for the user, depending on the threat analysis. However, the question arises here as to what can actually be done if an attacker has already infiltrated the device security to this extent. For the proof of concept, only the inputType shown was therefore adapted [6].

3.3.6 Background

One point that should not be forgotten when developing a mobile application is that the application can be moved into the background. A screenshot is taken just before the application is moved into the background. If the device is now lost, an attacker only has to change the application and can extract the desired information from the screenshot. In order to prevent this, the information must be cleaned up beforehand [5].

4 Evaluation

Under Android the mechanisms are very easy to implement. However, the biggest challenge in programming for Android is the patchwork of versions. It is difficult to develop an application for different Android versions with a consistent security standard. This is especially true for the encryption of local files. If the goal is to develop under API level 23, it is necessary to switch to asymmetric encryption or additional libraries for cryptography must be added. However, external libraries are often a reason for security holes. Last but not least, due to their wide range of functions, they can no longer be checked in the same way as an own implementation. However, using only asymmetric encryption is only possible for smaller files. These operations are very computationally intensive and thus increase the computing effort and battery consumption. Overall, the guideline shows effective methods for the basic scenarios that all mobile applications must consider. However, the list is in no way exhaustive. Especially when the mobility aspects are further discussed. There are many other scenarios where additional measures are recommended which can increase safety. An example of this would be to use a so-called obfuscation on Android. This term describes a technique that makes it more difficult to obtain readable code when decompiling an application.

References

[1] Josh Aas. *Why ninety-day lifetimes for certificates?* Nov. 2015. URL: `https://letsencrypt.org/2015/11/09/why-90-days.html` (visited on 06/01/2020).

[2] Android Developers. *Android 9 features and APIs.* 2019. URL: `https://developer.android.com/about/versions/pie/android-9.0` (visited on 05/31/2020).

[3] Android Developers. *Android keystore system.* 2019. URL: `https://developer.android.com/training/articles/keystore` (visited on 05/31/2020).

[4] Android Developers. *Application Signing.* 2020. URL: `https://source.android.com/security/apksigning/` (visited on 05/31/2020).

[5] Android Developers. *Guide to background processing.* 2020. URL: `https://developer.android.com/guide/background/` (visited on 05/31/2020).

[6] Android Developers. *Handle input method visibility.* 2019. URL: `https://developer.android.com/training/keyboard-input/visibility` (visited on 05/31/2020).

[7] Android Developers. *KeyStore.* 2019. URL: `https://developer.android.com/reference/java/security/KeyStore` (visited on 05/31/2020).

[8] Android Developers. *Network security configuration.* 2019. URL: `https://developer.android.com/training/articles/security-config` (visited on 05/31/2020).

[9] Android Developers. *Security with HTTPS and SSL.* 2019. URL: `https://developer.android.com/training/articles/security-ssl` (visited on 05/31/2020).

[10] Android Developers. *Sign your app.* 2020. URL: `https://developer.android.com/studio/publish/app-signing` (visited on 05/31/2020).

[11] M J Dworkin. *Recommendation for block cipher modes of operation :* tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Nov. 2007. DOI: 10.6028/NIST.SP.800-38d.

[12] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. Tech. rep. Fermont: Internet Engineering Task Force (IETF), June 2014. DOI: 10.17487/rfc7230.

[13] Adam Langley. *ImperialViolet - Public key pinning*. May 2011. URL: https://www.imperialviolet.org/2011/05/04/pinning.html (visited on 05/31/2020).

[14] Michael Muckin and Scott C. Fitch. *A Threat-Driven Approach to Cyber Security*. Tech. rep. Bethesda: Lockheed Martin Corporation, 2017. URL: http://ce.sharif.edu/courses/95-96/2/ce746-1/resources/root/Resources/Lockheed%20Martin%20Threat-Driven%20Approach%20whitepaper.pdf.

[15] E. Rescorla. *Diffie-Hellman Key Agreement Method*. Tech. rep. Fermont: Internet Engineering Task Force (IETF), June 1999. DOI: 10.17487/RFC2631.

[16] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Tech. rep. Fermont: Internet Engineering Task Force (IETF), Aug. 2018. DOI: 10.17487/RFC8446.