Kärt Ilja

Intercepting Network Traffic of the Smart-ID Android Application

Bachelor’s Thesis (9 ECTS)

Supervisor: Arnis Paršovs, MSc

Tartu 2020
Intercepting Network Traffic of the Smart-ID Android Application

Abstract: This thesis analyzes the technical means on how to monitor network communication between the Smart-ID Android application and the server. It gives an overview of the Smart-ID solution and then introduces the concept of man-in-the-middle attack used to intercept the traffic. To implement successful traffic interception attack, the certificate pinning mechanism had to be disabled in the Smart-ID application. This thesis provides step-by-step instructions on how to modify the Smart-ID application’s network security configuration and implement traffic interception using mitmproxy tool. Using the proposed methods network requests can be monitored to verify that no obvious personal data is being sent out from the user’s Android mobile device.

Keywords: Smart-ID, network interception, man-in-the-middle attack,

CERCS: P175 Informatics, systems theory

Smart-ID Android rakenduse võrgusuhtluse pealtkuulamine


Võtmesõnad: Smart-ID, võrgu pealtkuulamine, vahendusrünne

CERCS: P175 Informaatika, süsteemiteooria
## Contents

Introduction ..................................................................................................................4

1 Smart-ID solution.......................................................................................................5

1.1 Background ............................................................................................................5

1.2 SplitKey® Authentication and Digital Signature Platform ..................................5

1.3 Additional security .................................................................................................6

1.4 Network communication .......................................................................................7

1.4.1 JSON & RPC ....................................................................................................7

1.4.2 Communication security ....................................................................................7

2 Intercepting network traffic ......................................................................................8

2.1 Man-in-the-middle attack .....................................................................................8

2.2 Mitmproxy .............................................................................................................9

3 Implementing the MiTM attack ...............................................................................11

3.1 Setting up mitmproxy ..........................................................................................11

3.2 Simple HTTPS interception .................................................................................12

3.3 Smart-ID certificate pinning code analysis .........................................................13

3.4 Bypassing Smart-ID certificate pinning ..............................................................16

3.5 Results .................................................................................................................18

Conclusion ..................................................................................................................20

References ..................................................................................................................21

Appendix .....................................................................................................................24

I. Code snippet from Http.smali file ........................................................................24

II. Authentication HTTPS requests/responses .........................................................25

III. Licence .................................................................................................................32
Introduction

Estonia has been one of the fastest digitally evolving countries for many years. Amongst other things, a national-scale system has been developed, where a person can use electronic identity for everyday life operations. Electronic identity, also known as eID, is a collection of data, that connects person’s physical and digital environment identity [1]. With this solution it is possible to sign documents and verify user identity electronically.

Since electronic identity is widely used for many important operations, the process of authentication has to be secure and unsusceptible to misuse. There are many ways to authenticate eID, but one of the latest developed and perhaps the most user-friendly is Smart-ID service that is provided by SK ID Solutions (SK) [2,3].

As personal data and data processing is part of Smart-ID solution and it is available to use in the European Union [4], it falls under the General Data Protection Regulation (GDPR), which dictates how personal information should be processed [5]. In order to be GDPR protection and accountability principles compliant, SK has defined [3] what personal data is used and where, in the provision of Smart-ID service.

The purpose of this thesis is to find the technical means on how to monitor network traffic exchanged between Smart-ID application for Android and the server to verify the app is not exposing more about the user’s personal data than is described in the Smart-ID privacy policy [3]. To view the transferred data, application’s security protocols need to be bypassed, so that the network requests and responses will be shown in plain text. This requires the modification of Smart-ID application, which will be described in this thesis.

In the first section, the Smart-ID service and its security protocol is outlined, this also includes the description of network connection structure between a Smart-ID application and the web server. Second section describes a man-in-the-middle attack and gives an overview of mitmproxy tool. Mitmproxy is then used to listen in on Smart-ID network traffic, which is described in the third section of this thesis. In addition to that, step-by-step instructions are given on how to modify the Smart-ID application’s network security configurations for network interception. Finally, the network requests are analysed and compared to the data privacy policy provided by SK ID Solutions.
1 Smart-ID solution

The Smart-ID solution is an electronic identity (eID) product that allows a person to authenticate and give electronic signatures using a mobile device that does not require a special purpose hardware, such as SIM-card or Trusted Platform Module chip, and works without any extra-ordinary permissions [6]. It does not depend on state borders and is therefore applicable all over the world [2], meaning that the same Smart-ID could work in different countries. This chapter gives an overview of the Smart-ID application and its security details with a focus on network connection components.

1.1 Background

Smart-ID solution was first introduced by SK ID Solutions (SK) in November 2016, but was not taken to use as an authentication method until February 2017. SK was founded by three of the biggest telecommunication companies in Estonia: Telia Eesti, SEB Pank and Swedbank, and they are working in partnership with government of Estonia in issuing certifications for identity documents. At the moment there are more than 2 million Smart-ID users from 3 different countries [2] and it is verified as a QSCD (Qualified Signature Creation Device), which is the highest security level for signature giving service in European Union [6].

1.2 SplitKey® Authentication and Digital Signature Platform

The authentication technology used in Smart-ID was developed by Cybernetica AS as part of the SplitKey® Authentication and Digital Signature Platform and it is based on public key cryptography, digital signature schemes and follows public key infrastructure policies for maximum security and reliability [7]. Public key cryptography works on the concept of key pairs. The key pair consists of a public and a private key, where the public key is published and bind with the verified identity of the end-user. When executing an important function it is required that all key “holders” are present [8].

Smart-ID security is guaranteed by shared key management technology and PIN codes. User’s generated PIN codes are not stored locally in mobile device and they are only used to decrypt the private key in application itself. When user enters a PIN code the user’s private key share stored on the mobile device is decrypted and is used to create signature share that is sent to Smart-ID server. Smart-ID server completes the signature using user’s private key.
share stored on the server. If the resulting signature can be verified then the correct PIN was used by the user to decrypt user’s private key share stored on the mobile device [6]. Smart-ID account is made secure with 2 PIN codes [4]:

1) PIN1 is at least 4 digits long and is used for authentication to get access to e-services,
2) PIN2 is at least 5 digits long and is used to sign documents or confirm actions such as bank transactions,

which are either generated randomly or chosen by the user during Smart-ID account registration process.

Using SplitKey® authentication is only one Smart-ID solution’s building block. There are additional security measures to achieve maximum availability, reliability and security [9].

1.3 Additional security

SplitKey® authentication is only the first level to ensure that all the data is secure both server and end-user side. This is the basic verification on a device level, but any application with such degree of responsibility has to make sure that all possible brute-force and clone attacks are deflected.

For example, there is a time-delay lock. If user enters wrong PIN in three consecutive times, then the user’s account gets locked for 3 hours. During those 3 hours, authentication and signature transactions are not accepted and PIN tries are not allowed. After that 3-hour wait, there can be another three tries, and if those fail too, then user gets locked out for 24 hours. When those last 24 hours are up and the final three tries fail, user’s account gets closed and the certificates are revoked [6,10].

Clone detection is the function which attempts to recognize the situation when the user’s PIN and “Private Key Password” have been leaked and the attacker actively attempts using electronic signature function as the user. Smart-ID solution has been designed in such a way that all methods, which include the shares of the private keys, require specific one-time content from the previous invocations, to later compare it and then accordingly act [6].
1.4 Network communication

Smart-ID solution allows end-users to authenticate themselves to relying party (RP) systems, which include different websites, apps and call centres. This is done by means of Mobile device API (MD-API) amongst others Smart-ID service APIs. Different APIs are described in detail on the Smart-ID wiki page [9]. As this thesis is focused on the Smart-ID app, a more detailed analysis of the MD-API is given in the following sections.

Smart-ID App instances use MD-API for all communications between Core and the app itself, as well as run transaction management and different protocols. Smart-ID Core is a component that mainly implements business logic and manages the database. MD-API is built on the principle that access to it is protected with the HTTP Basic Authentication and uses JSON standard for text encoding with RPC styles. To authenticate all requests and queries a unique identifier and random password are assigned to each Smart-ID App instance, so that only associated app instances get access to the data [9].

1.4.1 JSON & RPC

JavaScript Object Notation (JSON) is a standard data format that facilitates applications to communicate over a network and it is human and machine readable so that meaning can be derived from it [11,12].

Remote Procedure Call (RPC) is a client invocation of a service by making a local procedure call that hides the details of the network connection and placing them in client’s local procedures [13]. According to RPC specification [14] the caller first sends a call message, that includes procedure’s parameters, to the receiver process and waits for a reply message. The receiver then extracts the parameters, computes the results and sends the reply message, from which the waiting caller extracts results of the procedure. By a description of MD-API and the Smart-ID authentication process [9], we could say that the caller in the scope of this thesis is Smart-ID app, the receiver is Smart-ID Core and procedure parameters are sent as JSON formatted data.

1.4.2 Communication security

Smart-ID service is a web-based application and uses Hypertext Transfer Protocol Secure (HTTPS), meaning that HTTP web requests and responses are exchanged between the Smart-ID app and the server over Transport Layer Security/Secure Sockets Layer (TLS/SSL) channel [15]. Smart-ID app authenticates the server using public key certificates and therefore as long as private key of the server is not compromised, network attackers cannot eavesdrop on the communication [9,16].
2 Intercepting network traffic

There are several ways to monitor network traffic exchanged between an app and the server. One of the options is to use hooking on a TLS library calls to obtain plaintext communication data from the device before it gets encrypted and after it has been decrypted. Another possible way is to dump the symmetric keys, that are used to encrypt TLS traffic, from the device’s memory, capture encrypted network traffic and decrypt it. Patching an application to accept untrusted certificate and performing a man-in-the-middle (MiTM) attack is also a possibility. As the last option is the least challenging to implement, it was chosen and will be discussed in the rest of this thesis.

In this section, the concept of a MiTM attack is described and an overview of mitmproxy tool for implementing that type of an interception is given.

2.1 Man-in-the-middle attack

Man-in-the-middle attack is an active eavesdropping on the network traffic that exploits the fact that HTTPS relies on the trustworthiness of the certificate authority (CA) and as part of such an attack the initial CA is replaced with a new one [16]. Certificate authority system is designed to prevent MiTM attacks from happening and in the event of identifying a non-trusted party the client will drop the connection and refuse to proceed [17]. In HTTPS connection, the transferred data is encrypted using TLS with a shared secret between a client and a server, so that the middle-man/proxy cannot decipher exchanged data packets [18].

When a client is connecting to the secure web server (HTTPS connection) there are two conditions that have to be met in order for the client to assume that the connection is secure. Firstly, it has to be verified that the server’s host name matches the one it is connecting to and secondly, it needs to be checked if the certificate was signed by a CA trusted by the client [18]. In the context of Smart-ID, the CA that signed the certificate is actually not important, because Smart-ID app has hardcoded the certificate that it expects from the Smart-ID server.
2.2 Mitmproxy

Mitmproxy\(^1\) is an interactive man-in-the-middle proxy application for real-time HTTP and HTTPS traffic interception, inspection, modification and replaying. Mitmproxy sits in between the client (such as a mobile or a desktop browser) and a web server and forwards network traffic from both sides [17]. The idea is to pretend to be the server to the client and the client to the server.

Mitmproxy has four modes of proxies [17]:

1. **Regular** – client will be configured to explicitly connect to mitmproxy and mitmproxy then explicitly connects to the target server.
2. **Transparent** – client is not configured, network traffic will be directed to the mitmproxy, before being sent to the Internet.
3. **Reverse** – client is configured to use mitmproxy as a destination web server, meaning the proxy will sit between the Internet and the server.
4. **Upstream** – client will be configured to explicitly connect to mitmproxy, which unconditionally forwards all requests to a specified upstream proxy server, which in turn is connected to the destination server.

As Android applications bypass the system HTTP proxy settings, which would be the result if a regular proxy method is used, a transparent mode of mitmproxy will be implemented in the practical part of this thesis. In transparent mode no client configuration is needed and the traffic is directed at the network layer (Figure 1) for the server running proxy to be able to intercept and decipher sent IP packets [17,18].

---

\(^1\) [https://mitmproxy.org/](https://mitmproxy.org/)
Figure 1. Flow of network traffic between a client (Smart-ID application) and the server with mitmproxy listening [18].

More detailed description on how transparent type of proxy was set up, is given in the next section of this thesis.
3 Implementing the MiTM attack

This section describes the method for disabling certificate pinning of Smart-ID application in order to man-in-the-middle the network connection and try to see authentication requests and responses in plain text. This includes a step-by-step description of setting up a MiTM proxy to listen in on the network traffic between a mobile phone’s application and a server.

Experimental testbed consisted of LG Nexus 5X smartphone (running Android 10) with a laptop HP EliteBook 840 G4 (running Windows 10) acting as an attacker’s laptop and a proxy server. All experiments were done with the latest version of Smart-ID application for Android (v18.4.177) that was the client in the proxy setup. Mitmproxy was used to intercept the network traffic and to implement a man-in-the-middle attack. Before using mitmproxy with the Smart-ID application, Smart-ID app was downloaded and installed to the Android smartphone and then the smartphone was registered and connected to the user’s Smart-ID account.

Authentication feature of Smart-ID service was used for testing to access the user account on https://portal.smart-id.com/login and the related network calls were then viewed in mitmproxy UI.

3.1 Setting up mitmproxy

To intercept and view Smart-ID application network traffic a proxy has to be set up, which will be the “man-in-the-middle”. For this mitmproxy was used, as it requires very minimal effort to set it up and has all the necessary functions for network connection interception [19].

Following mitmproxy documentation guidelines [17] a proxy between an Android phone and the Internet was set up using a computer as the mitmproxy server to view the traffic. This includes configuring the client to use proxy machine’s IP as the default gateway IP address, so that when the traffic reaches mitmproxy the destination IP is still intact.

Setting up mitmproxy consisted of the following steps:

1. Downloading and installing mitmproxy to the computer.
2. Connecting computer and phone to the same Wi-Fi network.
3. Changing the address of phone’s network gateway to point to the IP of the computer, where mitmproxy is running.
4. Launching mitmproxy server on the computer.
5. Opening “mitm.it” webpage from phone’s browser to verify that connection to Mitmproxy server is there and to download its CA key file for Android.
6. After downloading the CA key file, it was saved and added to the phone’s Android trust store to make the phone (client) trust mitmproxy CA. This CA is used to dynamically generate dummy certificates to whatever TLS secured site/hostname connection the client initiates [17,18].

As a result, mitmproxy decrypts the TLS connections made by the phone [19] and the requests are shown in mitmproxy UI.

3.2 Simple HTTPS interception

First test consisted of logging in to Smart-ID webpage account, without any additional configurations and using just a simple transparent proxy setup, which was described in Section 2.2 of this thesis. For authentication, Smart-ID login page was opened on the computer and the Smart-ID option for entering the portal was chosen. The first step in authentication was to enter user’s personal ID code, to validate the user. ID code acts as a user identifier, when authentication feature is used [4]. After that a notification was sent to the user’s registered phone, to enter the authentication PIN (PIN1).

When the authentication request was received by the Smart-ID application, the app recognized that the X.509 certificate of the Smart-ID server is not as expected and an error was shown (Figure 2). The Smart-ID app detected this, because Smart-ID app has an additional certificate pinning mechanism to ensure that the app can make connection to the Smart-ID server only if the server presents in TLS handshake server’s certificate that is expected by the app.
In order to disable the certificate pinning, Smart-ID source code needs to be changed and in the next section of this thesis, the respective code will be analysed and the method for finding the right code snippet will be explained.

### 3.3 Smart-ID certificate pinning code analysis

Bhor and Karia have described [20] certificate pinning as a way of ensuring that the certificate exchanged by the server is the one which is expected by the client. As it was found out, in case of Smart-ID the certificate pinning is implemented by hardcoding the SHA256 (Secure Hashed Algorithm function that produces 256-bit length message digest [21]) digest of the certificate to be used by the Smart-ID server in the Smart-ID mobile application. The Smart-ID application compares the locally stored hash of the certificate with one provided by the Smart-ID server. Alternative way of certificate pinning is that the app hardcodes the name of certificate authority and trusts any certificate issued by that CA, but this is not the case in Smart-ID app.

To analyse the code responsible for certificate pinning, it first had to be found from the application’s source code. The latest version of Smart-ID application, that is used in this thesis, is obfuscated, meaning that the code has been deliberately made difficult for humans to read [22] by changing the names of methods and files to meaningless symbols and/or number sequences. The latest not obfuscated version of Smart-ID application is v10.4.88.
This app version was downloaded from the Apkpure\(^2\) and then the APK file was decompiled, using Android APK decompiler online\(^3\) to be able to read the application resource Java files. The resulting ZIP file was then downloaded and its contents extracted. Decompiled Java files can be found from the \sources\ directory in the extracted contents.

Smart-ID uses OkHttp library to make HTTPS requests with certificate pinning, which can be seen from the Java file \sources\com\stagnationlab\p026sk\util\Http.java method’s return type - OkHttpClient (see Figure 3, line 54).

```
private static OkHttpClient createClient() {
    Builder client2 = new Builder();
    CertificatePinner certificatePinner = pinCertificates(BuildConfig.PORTAL_BASE_URL,
    BuildConfig.PORTAL_PINS);
    client2.connectTimeout(30, TimeUnit.SECONDS);
    client2.readTimeout(30, TimeUnit.SECONDS);
    client2.writeTimeout(30, TimeUnit.SECONDS);
    client2.cookieJar(cookieJar);
    if (certificatePinner != null) {
        client2.certificatePinner(certificatePinner);
    }
    return client2.build();
}
```

Figure 3. Code snippet from Http.java file with line numbers.

The same code snippet contains the certificate pinning part (see Figure 3, line 56). CertificatePinner Java class constrains which certificates are to be trusted by the application \[23\]. From \sources\com\stagnationlab\p026sk\BuildConfig.java file, the variables used in pinCertificates method were found:

a) PORTAL_BASE_URL = "https://portal.smart-id.com"

b) PORTAL_PINS = "sha256/loUeIWZ1KN66wo2lcTdAJwJ7sqK3esJ/kVOXJ/U="

PORTAL_BASE_URL appoints the Smart-ID server’s hostname, which can be trusted and PORTAL_PINS is the hashed public key value(s) of said server’s certificate(s). When Smart-ID application initializes a HTTPS connection, it checks if the end server’s issued certificate meets these values and if not, the client will terminate the connection.

From the Http.java file, lines 61-63, it is shown that if the certificatePinner variable is not initialized or has a value of null, the certificate pins are not added to the client, meaning that the certificate pinning will be disabled.


\(^3\) [http://www.javadecompilers.com/apk](http://www.javadecompilers.com/apk)
To view the Smart-ID APK contents without decompiling it, the .apk extension of this file was renamed to .zip and the ZIP file was then unzipped. From the extracted files, the corresponding compiled code (see Appendix I) to createClient method from Http.java file can be found in a file smali\out\com\stagnationlab\sk\util\Http.smali. By commenting out the certificatePinner initialization line (invoke-static (..)) from that file, the certification pinning will not be added to the client (application) network connections.

As the newest version of Smart-ID application (v18.4.177) is obfuscated, it is possible to use the older version’s disassembled APK Http.smali file location and structure to find the certificatePinning method in the newer version. The newest version was downloaded from Apkpure and decompiled using Android APK decompiler online as was done with the older version. The resulting ZIP file’s contents were extracted. Because the source code can change with new application versions, the most similar code snippet was searched and found from sources/com/stagnationlab/p144sk/util/C3809c.java file (see Figure 4). From the code snippet, it can be seen that there are two lines with certificate pinning variable initialization (lines 83 and 84).

```java
private static CS151x m11675a() { 
    C5153a aVar = new C5153a();
    C5004a aVar2 = new C5004a();
    n1062a(getVar2, "https://portal.smart-id.com",
            "sha256/j1O64W11X9#4o21C7O3hJyj/L73g3eza3lKV3ZxJ/f-/u_/sha256/gxKGTLaA426ccodhH1TLzy2cPYovwgo
            LymvK8enSj0=,sha256/f0Dq4xqG4x4Kv7Uw1LJjeAYkCiP2lx8BHiHiLvFPlpve=-,sha256/tnasfFkcYrKemcwpW
            cv2S8coq/ctWacy12M2I=");
    ml1062a(getVar2, "https://rob1.smart-id.com/v1/",
            "sha256/jk1ApzC8o1o7vV9968G5we2ttBD8ukkJmNe9oN1Ype=-,sha256/EvmWv1qquFf5rjpKg/XYDupsqXoB3U7Re
            73D05YJbo=");
    aVar.m102799a(30, TimeUnit.SECONDS);
    aVar.m102799b(30, TimeUnit.SECONDS);
    aVar.m102795c(30, TimeUnit.SECONDS);
    CS12m nVar = f10658c;
    if (nVar == null) { 
        aVar.f155201 = nVar;
        aVar.m102791a(aVar2, m102529a());
    if (C3679c.m1351a() { 
        C4976a aVar3 = new C4976a(new C4976b) { 
            public final void log(String str) { 
                c3679c.m1354d("http");
            }
        };
        aVar3.m102798a(C4977a, f14708d);
        aVar.m102796a(C1540up aVar3);
    }
    return aVar.m102791a();
} 
    throw new NullPointerException("cookieJar -- null");
}
```

Figure 4. Obfuscated createClient method code snippet from C3809c.java file with line numbers.

Similar to the not obfuscated version of Smart-ID app, a corresponding .smali file was found: smali\out\com\stagnationlab\sk\util\c.smali, which will be used in the next section of this thesis to disable certificate pinning.
3.4 Bypassing Smart-ID certificate pinning

In order to bypass certificate pinning, the pinning needs to be disabled from the source code. This was done by modifying the certificate pinning code in the original Smart-ID Android package file (.apk file extension). The file was repackaged and rebuilt using the Apktool\(^4\). The steps to modify Smart-ID APK file were as follows:

1. Java 11 SDK was installed and the needed directories were added to the PATH system variable:
   a. $\text{AppData\Local\Android\Sdk\build-tools\29.0.3}$,
   b. $\text{Program Files\Java\jdk-11.0.2bin}$.

   This will be needed for the step 13 commands to work.

2. Apktool was installed to the computer’s directory $C:\text{Users}\text{Kart}\text{apktool}$ and its location was also added to the PATH system variable to later use this tool from a command line.

3. Smart-ID APK file was downloaded from Apkpure webpage [24], and was named $\text{SmartID-v18.apk}$.

4. Command prompt was opened under the directory of a previously downloaded APK file.

5. For APK file unpacking, the command $\text{apktool d SmartID-v18.apk}$ was executed. This created a folder with the same name as the file, that contains all the extracted files for a Smart-ID app to work, including resource files and the decompiled Java classes to be run on the mobile device.

6. The file $c\text{.smali}$ was located and opened in the directory $\text{SmartID-v18smali\com\stagnationlab\sk\util}$ This file is a decompiled Java code and was created during step 4, it contains the code for certificate pinning.

7. In order to disable the certificate pinning, the lines of code that activate the pinning method were commented out by adding a \#-symbol in front of the lines number 240 and 247 (Figure 5), which correspond to the code snippet lines 83 and 84 from Figure 4.

\(^4\) [https://ibotpeaches.github.io/Apktool/](https://ibotpeaches.github.io/Apktool/)
Figure 5. Screenshot of c.smali file code snippet with lines that were commented out.

8. File `tse.properties` was located in the directory `\SmartID-v18\assets\` and modified so that the “pins” property value was deleted (Figure 6). This property contained the trusted server’s hostname and its certificate public key hashed values, which had to be deleted, so that the application will later trust the certificate offered by mitmproxy.

![tse.properties](image)

Figure 6. Screenshot of `tse.properties` file snippet after the deletion of "pins" property value.

9. The file `network_security_config.xml` was located in the directory `\SmartID-v18\res\xml\` and then modified. Its contents were replaced with the new configuration settings (Figure 7) to remove the limitations on trusting only the Smart-ID custom CAs and make the app trust additional CAs [25], like the mitmproxy CA.

![network_security_config.xml](image)

Figure 7. Modified network security configuration of Smart-ID app.
10. Command prompt was opened again under the directory of the original Smart-ID APK file location.

11. To rebuild the modified Smart-ID application, command `apktool b SmartID-v18.apk` was executed, that created a new folder `\SmartID-v18\dist\`, which includes the new Smart-ID APK file, and the command prompt was opened under this new folder.

12. In order to be able to run this new APK file, it has to be resigned as Android requires that all APKs are to be digitally signed with a certificate [26]. To sign the APK file the following commands were executed in the `\dist\` folder:

   a. `keytool -genkeypair -v -keystore app.keystore -alias key0 -keyalg RSA -keysize 2048 -validity 10000` (generated the key and the keystore);
   b. `jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore app.keystore SmartID-v18.apk key0` (signed the new APK file with the generated key);
   c. `zipalign -f -v 4 SmartID-v18.apk sid.apk` (this was done to optimize the APK file and compress the data within the file to reduce the RAM consumed, when running this application on the mobile phone [27]).

As a result, modified Smart-ID application `sid.apk` was generated and ready to be installed to the test device for use. With the new APK file installed, the phone was registered and connected again to the user’s Smart-ID account with the conditions described in the introduction of Section 3.

To test if certificate pinning was disabled and mitmproxy was able to decrypt the HTTPS network requests, a new try for authenticating the user account on the Smart-ID webpage was constructed. This time no insecure connection errors were shown and Mitmproxy could see the traffic between Smart-ID application and the server.

### 3.5 Results

Four consecutive network calls were made during the Smart-ID authentication process, which were viewed in mitmproxy UI (see Figure 8). The HTTP requests and responses are shown in detail in Appendix II.
Figure 8. Smart-ID authentication requests shown in mitmproxy UI.

Based on the information of requests, no obvious personal data from user’s mobile device is being sent out, but to fully answer the question more effort would be needed to be put in to monitor Smart-ID app communication over longer period and investigating the contents of binary encoded data that is sent out. Only the authentication feature was monitored in the MiTM attack, but there are also other network communications that Smart-ID application performs with the server, which could also contain sensitive user data. Such extensive analysis falls outside the scope of this thesis.
Conclusion

The aim of this thesis was to find the technical means on how to monitor network communication between the Smart-ID app and the server to ensure that Smart-ID app follows the protocol and does not send more data to the server than is needed. For this to work, network call requests and responses needed to be shown in plain text. Since the application uses certificate pinning as an additional security for data transfer, the modification of Smart-ID app’s source code was required.

Certificate pinning code of Smart-ID app was analysed and a description was given on how to find the corresponding code snippets. For the man-in-the-middle attack implementation, step-by-step instructions were given on setting up a mitmproxy between a mobile device and the Internet to listen in on any network traffic. This setup was then used to attempt to view the Smart-ID authentication calls. With the combination of transparent proxy and the disabling of certificate pinning, network communication data was seen from the mitmproxy UI. No extensive personal data was found from the network requests made by the Smart-ID application’s authentication feature.
References


[5] What is GDPR, the EU’s new data protection law. GDPR webpage. https://gdpr.eu/what-is-gdpr/ (06.05.2020)


[22] Rouse, M. What is obfuscation (obfu)? https://searchsoftwarequality.techtarget.com/definition/obfuscation (07.05.2020)


Appendix

I. Code snippet from Http.smali file

Below is the decompiled executable file’s representation of the corresponding Http.java file code snippet from the version 10.4.88 of Smart-ID application:

```
.method private static createClient()Lokhttp3/OkHttpClient;
.registers 6
.prologue
const-wide/16 v4, 0x1e
.line 58
new-instance v1, Lokhttp3/OkHttpClient$Builder;
.line 59
.local v1, "client":Lokhttp3/OkHttpClient$Builder;
const-string v2, "https://portal.smart-id.com"
const-string v3, "sha256/loUeIW21KN6wo21cTDAJwJL7sqK3esJ/kVOXJ/Um"
.line 59
invoke-direct {v1}, Lokhttp3/OkHttpClient$Builder;-><init>()V
.line 59
LOCAL v0, "certificatePinner":Lokhttp3/CertificatePinner;
.LINE 61
.invoke-static {v2, v3}, Lcom/stagnationlab/sk/util/Http;->pinCertificates(Ljava/lang/String;Ljava/lang/String;)Lokhttp3/CertificatePinner;
MOVE-result-object v0
.line 59
LOCAL v2, Ljava/util/concurrent/TimeUnit;->SECONDS:Ljava/util/concurrent/TimeUnit;
.line 63
.invoke-virtual {v1, v4, v5, v2}, Lokhttp3/OkHttpClient$Builder;->build()Lokhttp3/OkHttpClient;
MOVE-result-object v2
return-object v2
.end method
```
II. Authentication HTTPS requests/responses

Below are provided the four network calls made from the Smart-ID app that were made during the Smart-ID authentication process. The requests were viewed in mitmproxy UI.

Request 1:
POST https://mobile-api.smart-id.com/v2/protected HTTP / 1.1
Accept-Charset: utf-8
Authorization: Basic
OWVhMzhiNjgtY2U5NC00MjJkLWFmODUtNWIyNzdkNTyyYjU2OjZzNVVGUl1KdXNrrS1kUzk =
Smart-X-ID-Library: 14.1.120-sk-release
Smart-X-ID-UI: 18.4.177
Smart-X-ID-Platform: Android
Content-Type: application / JSON-RPC
Content-Length: 176
Host: mobile-api.smart-id.com
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp / 3.12.2
{
  "id": "6c9c918e",
  "jsonrpc": "2.0",
  "method": "getRpRequest",
  "params": {
    "accountUUID": "e673b8ad-4ffd-44fa-be29-22b666eeec9",
    "rpRequestUUID": "941b4579-e213-41d1-abdb-dfa01389db97"
  }
}

Response 1:
HTTP / 1.1 200
Access-Control-Expose-Headers: X-Plumbr-transactionId
X-Plumbr-transactionId: f5f8f898-bd80-12aa-a8d7-aa3c4ee6a987
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode = block
Cache-Control: no-cache, no-store, max-age = 0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application / json
Transfer-Encoding: chunked
Date: Wed, 06 May 2020 13:04:45 GMT
Keep-Alive: timeout = 20
Connection: keep-alive
{
  "id": "6c9c918e",
  "jsonrpc": "2.0",
  "result": {
    "rpRequest": {
      "requestType": "AUTHENTICATION",
      "rpName": "SMART-ID PORTAL",
      "rpRequestUUID": "941b4579-e213-41d1-abdb-dfa01389db97",
      "ttlSec": 99
    }
  }
}
Request 2:

POST https://mobile-api.smart-id.com/v2/protected HTTP / 1.1
Accept-Charset: utf-8
Authorization: Basic OWVhMzhiNjgtY2U5NC00MjkJLWFMdDUtNW1yNzkNTYyYjU2Oj2zNVVGU1KdXNrS1kxUzk =
Smart-X-ID-Library: 14.1.120-sk-release
Smart-X-ID-UI: 18.4.177
Smart-X-ID-Platform: Android
Content-Type: application / JSON-RPC
Content-Length: 193
Host: mobile-api.smart-id.com
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp / 3.12.2
{
  "id": "4ea5a668",
  "jsonrpc": "2.0",
  "method": "createTransactionForRpRequest",
  "params": {
    "accountUUID": "e673b8ad-4ffd-44fa-be29-22b666eeec9",
    "rpRequestUUID": "941b4579-e213-41d1-abdb-dfa01389db97"
  }
}

Response 2:

HTTP / 1.1 200
Access-Control-Expose-Headers: X-Plumbtr-transactionId
X-Plumbtr-transactionId: c5afeeced-c5dc-ff2c-498f-ed38d1132e04
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode = block
Cache-Control: no-cache, no-store, max-age = 0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application / json
Transfer-Encoding: chunked
Date: Wed, 06 May 2020 13:04:48 GMT
Keep-Alive: timeout = 20
Connection: keep-alive
{
  "id": "4ea5a668",
  "jsonrpc": "2.0",
  "result": {
    "freshnessToken": "kQZC3Y0oLAs =",
    "transaction": {
      "accountUUID": "e673b8ad-4ffd-44fa-be29-22b666eeec9",
      "displayText": "Log in to SMART-ID portal",
      "hash": "e9gOK90dptC2HdTRsAjdDo2p1Ko8 + QvIcYJegqUAxOIsE6ktgHMB87b5zF / U8tuS6b916RkCZS0o / P8xHb3wzQ ==",
      "hashType": "SHA512",
      "rpName": "SMART-ID PORTAL",
      "state": "RUNNING",
      "transactionSource": "RELYING_PARTY",
      "transactionType": "AUTHENTICATION",
      "transactionUUID": "7cf45b45-ba66-4814-b7c6-5c764f41d241",
      "ttlSec": 89
    }
  }
}
Request 3:

POST https://mobile-api.smart-id.com/v2/protected HTTP / 1.1
Accept-Charset: utf-8
X-SplitKey trigger: external
Authorization: Basic OWVhMzhiNjgtY2U5NC00MjJkLWFmODUtNWYyNzdNNTYyYjU2OjZzNVVGUl1KdXNrs1kzuZk =
Smart-X-ID-Library: 14.1.120-sk-release
Smart-X-ID-UI: 18.4.177
Smart-X-ID-Platform: Android
Content-Type: application / JSON
Content-Length: 2525
Host: mobile-api.smart-id.com
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp / 3.12.2
{
   "id": "d1742743",
   "jsonrpc": "2.0",
   "result": {
   "ResponseData": "eyJhdWQiOiJDTElFTlQiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2Iiw1YXN0IjoiZGlyIiwia2V5VVVJKCI6i1MTdiNDNmLTM3MzItNDQ2NC11MzltLVVm2DQ2Mz1M1Yz1CIsImtpZCI6IjAiM
TdindNDNmsLTm3MzItNDQ2NC1zMiIzLTVm2DQ2Mz1M1Yz1CJ9..GwUp7WOXHYJUzcFEzFudQ.b
TXuLR5draKRXofrVKpAgpI26heSQQptdUml7JwvXXv-
KgMteh68X8RumzMNb.Oyc07jRhsZq6LL41Lzww"
   "responseDataEncoding": "JWE",
   "result": "OK"
   }
}

Response 3:

HTTP / 1.1 200
Access-Control-Expose-Headers: X-Plumber-transactionId
X-Plumber-transactionId: a1151393-643e-dfc8-e749-06a32cbff0c
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode = block
Cache-Control: no-cache, no-store, max-age = 0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application / json
Transfer-Encoding: chunked
Date: Wed, 06 May 2020 13:04:53 GMT
Keep-Alive: timeout = 0
Connection: keep-alive
{
   "id": "d1742743",
   "jsonrpc": "2.0",
   "result": {
   "ResponseData": "eyJhdWQiOiJDTElFTlQiLCJlbmMiOiJBMTI4Q0JDLuhTMjU2Iiw1YXN0IjoiZGlyIiwia2V5VVVJKCI6i1MTdiNDNmLTM3MzItNDQ2NC11MzltLVVm2DQ2Mz1M1Yz1CIsImtpZCI6IjAiM
TdindNDNmsLTm3MzItNDQ2NC1zMiIzLTVm2DQ2Mz1M1Yz1CJ9..GwUp7WOXHYJUzcFEzFudQ.b
TXuLR5draKRXofrVKpAgpI26heSQQptdUml7JwvXXv-
KgMteh68X8RumzMNb.Oyc07jRhsZq6LL41Lzww"
   "responseDataEncoding": "JWE",
   "result": "OK"
   }
}
Request 4:

POST https://mobile-api.smart-id.com/v2/protected HTTP / 1.1
Accept-Charset: utf-8
Authorization: Basic OWVhMzhiNjgtY2U5NCO0MjJkLWFmODUtNWIyNzdNNTYyYjU2OjZzNVVGUl1KdXNRslUxkUz =
Smart-X-ID-Library: 14.1.120-sk-release
Smart-X-ID-UI: 18.4.177
Smart-X-ID-Platform: Android
Content-Type: application / JSON-RPC
Content-Length: 125
Host: mobile-api.smart-id.com
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp / 3.12.2
{
    "id": "c3ac3522",
    "jsonrpc": "2.0",
    "method": "getAccountStatus",
    "params": {
        "accountUUID": "e673b8ad-4ffd-44fa-be29-22b6666eeec9"
    }
}

Response 4:

HTTP / 1.1 200
Access-Control-Expose-Headers: X-Plumbir-transactionId
X-Plumbir-transactionId: d510f5d8-85ce-e31d-b2fc-393d9f6da0be
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode = block
Cache-Control: no-cache, no-store, max-age = 0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application / json
Transfer-Encoding: chunked
Date: Wed, 06 May 2020 13:04:56 GMT
Keep-Alive: timeout = 20
Connection: keep-alive
{
    "id": "c3ac3522",
    "jsonrpc": "2.0",
    "result": {
        "accountUUID": "e673b8ad-4ffd-44fa-be29-22b6666eeec9",
        "keys": [
            {
                "certificate": {
                    "qscd": 0,
                    "subject": {
                        "C": "EE",
                        "CN": "ILJA \, KÄRT \, PNOEE-49801120822",
                        "GN": "KÄRT",
                        "O": null,
                        "OU": null,
                        "SN": "ILJA",
                        "serialNumber": "PNOEE-49801120822"
                    }
                }"type": "QUALIFIED",
```
"status": "OK"

"certificate": {
  "qscd": 1,
  "subject": {
    "C": "EE",
    "CN": "ILJA, KÄRT, PNOEE-49801120822",
    "GN": "KÄRT",
    "O": null,
    "OU": null,
    "SN": "ILJA",
    "serialNumber": "PNOEE-49801120822"
  },
  "type": "QUALIFIED",
  "validSince": "2020-05-06T13:02:39Z",
  "validUntil": "2023-05-06T13:02:39Z",
  "value": "MIIIUDCCBjigAwIBAgIQf6MryNxET7FesrvHv6twwyDANBgkgkiG9w0BAQsFADBgMQswCQYD
VQQGEwJFRTExMECAAgEAwIBAgIeQDAWgMBQYDVQQIEw9KdEYHMRMwFwAwRBBDAdBgNVHQ8BAf8w
DQYJKoZIhvcNAQEFBwYwYI<...>
  
"}
III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I. Kärt Ilja,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   **Intercepting Network Traffic of the Smart-ID Android Application,**

   supervised by Arnis Paršovs.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons’ intellectual property rights or rights arising from the personal data protection legislation.

_**Kärt Ilja**_

_08/05/2020_