Open University of the Netherlands
faculty of Management, Science & Technology
Bachelor Computer Science

Fifty ways to lose your cover

Uncovering the Information Content of Android Attributes

Authors:
Katleen de Nil
850632006
Annet Vink
836559784

Chair(wo)man:
prof. dr. Tanja Vos
Supervisor:
dr. ir. Hugo Jonker

Presentation date:
July 13, 2018

July 3, 2018

Course code:
T61327

Abstract

Mobile phones contain an extensive set of attributes that may be used to track users. Android apps are known to use device attributes and user settings for fingerprinting, resulting in privacy loss for users. Currently the amount of privacy loss that is caused by fingerprinting Android apps is not fully known.

In order to calculate the entropy of attributes, a large dataset is needed. To facilitate data collection we developed an Android app that generates a fingerprint, which is in fact a large collection of attributes. App users can inspect the generated values and gain understanding of their privacy loss. They can upload the fingerprint to the research database and receive a short message about the result.

To comply with European privacy legislation both software and data are secured. Privacy-sensitive attributes are rendered anonymous on the device before uploading them to the research database. Data exchange between app and backend is secured by SSL. The app also contains some background information on the project and the privacy measures.

Entropy calculation is performed in the database, which makes the results available during data collection. The preferred method for comparing list attributes is the Jaccard index, that expresses the percentage of similarity in a set.

Contents

1 Introduction **3**
- 1.1 Objective . . . 3
- 1.2 Related work . . . 4
- 1.3 Legal context . . . 5
  - 1.3.1 Personal data . . . 5
  - 1.3.2 Principles . . . 6
  - 1.3.3 Pseudonymisation and anonymisation . . . 7
  - 1.3.4 Conclusion . . . 7

2 Security and privacy **8**
- 2.1 Introduction . . . 8
- 2.2 Privacy sensitive data . . . 8
- 2.3 Securing our tool . . . 9
  - 2.3.1 Securing the source code . . . 10
  - 2.3.2 Securing the data in transit . . . 10
  - 2.3.3 Securing the data at rest . . . 10
- 2.4 Security decisions . . . 13

3 Implementation of security and privacy **14**
- 3.1 Introduction . . . 14
- 3.2 Implementing HTTPS with SSL . . . 14
- 3.3 Hashing the privacy sensitive attributes . . . 17

4 Information content of attributes **18**
- 4.1 Introduction . . . 18
- 4.2 Self-information . . . 18
- 4.3 Calculation of entropy . . . 19
- 4.4 Acquiring information from a combination of attributes . . . 19
- 4.5 Data processing . . . 20
  - 4.5.1 Missing attribute values . . . 20
  - 4.5.2 Multiple fingerprints from the same device . . . 20

5 Matching fingerprints **22**
- 5.1 Using entropy to select attributes . . . 22
- 5.2 Matching fingerprints . . . 22
  - 5.2.1 Classification of attributes . . . 22
  - 5.2.2 Matching algorithm . . . 23
  - 5.2.3 Implementation . . . 24
- 5.3 List attributes . . . 24
- 5.4 Android versions and accessibility of attributes . . . 25

6 Results **27**
6.1 Software Requirements 27
6.2 Application architecture 27
6.2.1 Process flow 27
6.2.2 Design decisions 28
6.3 Android 28
6.3.1 User interface 28
6.3.2 Android versions 29
6.3.3 Android permissions 30
6.3.4 Implementation 30
6.4 PHP 33
6.5 MySql 34
6.6 Application tests 35

7 Conclusions and Discussion **36**

Appendices **39**

A List of collected attributes **40**

B Installation instructions backend **45**
B.1 Installing XAMPP 45
B.2 Creating the fingerprint database 46
B.3 Setting up the PHP engine 47
B.4 Installing Android Studio 48

C Attention points for further development **49**

D Cover Song **50**

Chapter 1

Introduction

1.1 Objective

Mobile devices are constant companions in our daily life. They contain apps for many different goals and usually carry a lot of personal information. In current society this customer information is very valuable to companies. Advertizing companies want to know their customers in order to show them relevant ads. Fraud prevention is also a reason to collect user data: If a user always uses the same device to log into an application, a login from a different device may indicate that something is wrong. For these purposes apps collect all kinds of data, sometimes without letting the user know exactly what is collected and why.
Collected data may be used in two slightly different ways. The purpose of fingerprinting is to recognize the user from different apps. This is the only way to perfom cross-app tracking for apps that do not require the user to log in. The collected data is only used to compute the number of similar fingerprints. On the other hand, profiling occurs when the actual content of the collected data is used to derive information about the user.
This paper focuses on datacollection for fingerprinting.

When apps collect user data, questions rise about the resulting loss of privacy. Questions like: How much privacy are we giving up when using apps and maybe grant permissions to read sensitive data? Which data are sensitive and will reveal your identity among thousands of others? How unique is your fingerprint? Does it matter if you grant or revoke permissions to an app?
The answer is that for Android devices there is not much knowledge available about privacy loss. One study [Wu+16] calculated the privacy loss for a limited set of data, but we presume that many more device characteristics and user preferences can be used.
Inspired by Eckersley et al., who collected and analyzed an extensive amount of browser fingerprints [Eck10], a similar study for fingerprinting Android devices would answer the above questions.

A study to collect device-related as well as user-related data from Android devices will result in:

- Entropy values of many characteristics of Android devices, that can be collected by apps. This includes characteristics that need permission from the user.
- Entropy of combinations of characteristics, to find out which combinations may be used to identify users.
- The influence of granting or revoking permissions on privacy loss.

This thesis is the account of an Open University bachelor project that is part of the Android fingerprint study. The actual collection and analysis of the data are explicitly not included in our project. Our contributions are:

- To facilitate data collection: design and development of an environment to collect device attributes that may be usable in a fingerprint.
- To facilitate data analysis: design and development of software for entropy calculation for collected data.
- To comply with European privacy legislation: securing software and data.

When collecting data, special attention must be paid to privacy, the application must comply with the new stringent EU General Data Protection Regulation (GDPR). This means that all personal data must be protected.
The remainder of this introductory chapter is used to provide some background information about fingerprinting literature and the new privacy legislation GDPR.
Besides the data collection environment, two topics received extra attention during this project. The first is the application security and its implementation. These are the subjects of chapters 2 and 3, that show possible security measures and explains which ones are chosen for the application. The other special interest topic is the information content of device attributes, chapter 4 contains the theory of data analysis with an explanation of self-information and entropy and how this is incorporated in the application. Matching fingerprints is covered in chapter 5.
Chapter 6 contains a description of the software implementation and database and design decisions. In the last chapter we come to the conclusions and discussion part. We show whether all requirements have been met and discuss improvements.

1.2 Related work

Mobile device fingerprinting may use physical hardware and software characteristics or any combination of these elements.
Android devices can easily be identified by unique identifiers, e.g. IMEI or Android ID. Apps need user permissions to access unique hardware identifiers, but Wu et al. showed that a device may also be identified by using implicit identifiers [Wu+16] that don't need any user permission. They created an Android app and used it to collect over 50,000 fingerprints. Each fingerprint contained 38 identifiers that were freely accessible. The features that contained most information were the list of user packages, the current wallpaper and the list of system packages. The acquisition of the lists of user and system packages can be blocked by security software, therefore these identifiers are not reliable enough for fingerprinting.
The researchers compiled a list of 10 identifiers that accounted for a fingerprint recall of 98.55% and a precision of 99.68%.

A similar study was executed for Apple devices. In iOS7, Apple restricted access to hardware identifiers, because these are excellent for fingerprinting. Kurtz et al. [Kur+16] conducted a study to find alternative identifiers to fingerprint Apple devices. Their results showed that all fingerprints are unique and distinguishable, but they also change over time. By using a supervised learning method, the researchers were able to recognize 97% of the devices.
The study also tried to find a minimal set of characteristics that showed optimal diversity and stability. The most informative characteristic was the list of installed apps combined with the device name, leading to 99% discrimination and 96.67% re-identification. Including protected resources in the fingerprint like top 50 music, did not increase the accuracy of the results.

Many commercial libraries use a form of fingerprinting to track users for fraud prevention or advertising. Jonker et al. [JF17] investigated 8 commercial fingerprinting libraries and made an overview of the characteristics that were used. They found that all libraries use the globally unique Android ID, extended with a diverse set of other characteristics. About 19% of the free apps in the Google Play store use a fingerprinting library.

Because Android apps can use browser windows, it is also interesting to see which browser characteristics are usable for fingerprinting. The leading study in this field is Eckersley's Panopticlick website[1], which is a browser fingerprinting website built for research and for user awareness. This site collects version and configuration information from web requests and measures the unicity of browser fingerprints [Eck10]. The conclusion from this study is that browser fingerprints carry a lot of information and browsers supporting Flash or Java expose even more information.

One of the objectives of Eckersley's research was to find out if successive fingerprints from the same browser are recognizable without using cookies. The researchers used cookies only to confirm that a fingerprint was recognized correctly. Although fingerprints change frequently, previously taken fingerprints could mostly be traced with a simple heuristic.
An additional method of browser fingerprinting is by using the HTML canvas [MS12]. This technique uses a script to draw some letters on the canvas, captures the resulting binary canvas data and creates a hash that is used as a fingerprint. Small differences in Graphics Processing Unit (GPU) and graphics driver lead to different binary data. The authors claim that this method leads to an entropy of 5.7 bits. The entropy in itself is not high enough to provide a unique fingerprint, but this method is valuable when combined with other techniques. Both techniques are completely transparent, the user is unaware of the fingerprinting.

1.3 Legal context

This section describes the new data privacy regulation that is in force since May 2018 and that we must comply with in our project.
The European General Data Protection Regulation controls how businesses must strengthen and unify data protection of all EU citizens. This new law replaces a database protection directive from 1995 which is no longer adapted to current standards. The GDPR entered into force in May 2016 and companies will have until 25 May 2018 to comply with this new directive. Subsequently, everyone can demand that organisations comply with this law. Failure to comply with this guideline may result in fines of 4 percentage of a company's maximum turnover or 20 million euros, where the highest variant applies [2].

1.3.1 Personal data

This regulation applies to personal data of EU citizens and defines it as follows (Article 4)[3]:

'Personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;

[1]https://panopticlick.eff.org/
[2]https://en.wikipedia.org/wiki/General_Data_Protection_Regulation
[3]http://eur-lex.europa.eu/legal-content/NL/TXT/?uri=CELEX%3A32016R0679

1.3.2 Principles

The General Data Protection Regulation is based on 8 basic principles[4]:

- *Transparency*:
  The person whose details are processed is aware of this, has given permission and knows his rights. The GDPR includes a mandatory list of information which must be given to users whose data is obtained directly or indirectly.

- *Purpose limitation*:
  The collected data is only used for the original stated purpose and may not be used for other purposes.

- *Data minimisation*:
  Only the data necessary for the original stated purpose may be collected.

- *Accuracy*:
  The personal data must be correct and kept up to date.

- *Storage limitation*:
  Once the personal data is no longer required for the original stated purpose, the data must be deleted.

- *Right of access, right to erasure and data portability*:
  Users have the right to access their personal data and information about how these data are being processed (Article 15). Upon request of the user, the responsible must provide an overview of the category of collected data and a copy of the actually stored data.

  In some cases, users also have the right to request that their data must be deleted (Article 17). The data must then be physically erased or it must be anonymised.

  A user must be able to transfer his personal data from one electronic processing system to another (Article 20). This does not apply to data that are sufficiently anonymised but it does apply to data that still can be linked to a person.

- *Integrity and confidentiality*:
  The personal data must be protected against unauthorized access, loss or destruction. The data must be protected at a level that corresponds to the sensitivity of the data.

- *Accountability*:
  The responsible must be able to demonstrate compliance with these rules. Many organizations are also required to nominate a Data Protection Officer who is responsible for everything related to the security of personal data.

  A data breach must be reported to the Personal Data Authority within 72 hours (Article 33). Users must be informed if disadvantageous effects are identified (Article 34).

  However, notification is not required if the controller has taken appropriate technical and organizational security measures that make the personal data incomprehensible to persons who do not have access, such as encryption and pseudonymisation (Article 34).

[4]https://www.smartconnections.nl/basisprincipes-gdpr/

1.3.3 Pseudonymisation and anonymisation

A reference is made to this section later on in section 2.4, where will turn out that our solution balances between pseudonymisation (GDPR applies) and anonymisation (GDPR does not apply). To understand later on to what extent our solution meets the GDPR, it is important to understand the difference between these techniques.

The GDPR mentions pseudonymisation as a privacy enhancing technique, whereby personal data is processed without the possibility to link it to a specific person. This is achieved by making information non-assignable without additional information, which must be retained separately and is subject to various technical and organizational controls. The data processor is still able to identify the data subject based on this additional information. Pseudonymisation is therefore reversible, but it reduces the chance of misusing personal data in case of a data breach.

The GDPR states encryption, a technique to enable pseudonimisation, as a safe technique but does not require it. In fact, the word 'encryption' is only mentioned 4 times in the legal text and no further technical context has been outlined. Is it about encryption in transit or encryption at rest? What kind of algorithm should be used? Which key length?

The GDPR applies to all pseudomyzed personal data. GDPR does not apply to anonymous data, this is personal data that has been made anonymous so that the data subject is no longer identifiable, not even by the data processor. This is typically used for statistical purposes or research purposes. Opposite to pseudonimisation, anonimization is not reversible.

1.3.4 Conclusion

In this section, we have tried to give a general overview of what GDPR entails and it contains general legal information that should be kept in mind when reading this thesis and in particular when reading the chapter about security. The conclusions and interpretations of this regulation for our project are given in section 2.4.

Chapter 2

Security and privacy

2.1 Introduction

Security nowadays is a hot item and not without reason. Every day, individual hackers and professionally organized criminals are active on the internet. Sometimes hackers are rather 'good guys' who only want to track security breaches in a system, but mostly hackers also have malicious intentions. Because we do not want the data gathered by us to be leaked just like that, extra attention in our assignment must be payed to the security aspect. This thesis is obviously not a thesis on security, but we also do not want our data to be intercepted by third parties.

In the case that our data would be intercepted, it would not be a problem for the biggest part of the collected attributes. The biggest part is rather technical information that cannot be misused much by third parties. However, a smaller proportion of the data is privacy sensitive and this must be handled with extra care. Section 2.2 enumerates which privacy sensitive attributes are gathered. Because of this privacy aspect, it is extra important to secure our data. We want to ensure that no personal data is leaked and we want at all times avoid problems with the GDPR regulation.

In section 2.3 we consider how we can secure our tool and this section is rather technically oriented. An overview is given of the structure of the system and for each component is explained which security issues might occur. Then, for each of these components, we will discuss how these issues can be dealt with. Extra attention is paid to the 'database encryption' part, because we have considered for a long time to encrypt the database. We thought for quite a long time that this was the ultimate solution to secure our data.

The reasons why we finally have not chosen 'database encryption' are explained in section 2.4. This section explains what security solutions we have chosen and why.

2.2 Privacy sensitive data

This section describes in detail which privacy sensitive information our tool will be able to collect. It can be considered as an exhaustive list. All attributes that are not quoted in this section are technical attributes and are not privacy sensitive.

The attributes that contain personal information can be directed specifically to a person. These attributes are therefore considered as personal data according to the GDPR legislation as stated in 1.3.1. Hence, we must adhere to the principles cited in 1.3.2.

The attributes *phone number* and *user name* certainly contain personal information. They undoubtedly must be treated privacy sensitive. The list of *Local IP Addresses*, the *WiFi Ssid* and *WiFi Bsid* are debatable. To avoid any discussion, we also treat these attributes as privacy sensitive.

Some doubt had arisen about the *Google Advertising ID.* This ID can be collected and it is gathered by Google itself, the user can at all times reset this Advertising ID[1]. The Google policy states that the Advertising ID may not be associated with personally identifiable information or other identifiers. In addition, there must also be transparency to the users. From this sentence it can be concluded that the Advertising ID is not privacy sensitive, as long as it is not linked to personally identifiable information.

The remaining attributes collect rather technical information about the screen, storage, OS, installed app's and fonts, etc. They can not be traced back to a specific person and are certainly not privacy sensitive.

The following section discusses the ways to protect the data and especially the privacy sensitive data, but our final decision only follows in section 2.4.

2.3 Securing our tool

To think up an approach to protect the data, an insight is needed into the structure of the system and the types of attacks against which the data must be protected.
The system consists of a fingerprint app that collects data about the mobile phone, an Apache webserver and a MySQL database. In the source code of the fingerprint app, a connection is made to a PHP page which is installed on the Apache webserver. The PHP page then connects to the database and ensures that the data is inserted into the database. Since we are dealing with privacy sensitive information, extra attention must be paid to the security aspect.

This privacy sensitive data must be secured in all layers of our application:

- *The source code of the fingerprint app.* The Android app will be published as open source, so the source code is publicly accessible and freely customizable by everyone. One could for example maliciously adapt the source code so that the phone data would be intercepted before it is even being sent or one could adapt the source code to insert malicious data in our database.
- *The data in transit.* The mobile phone sends this data over the unsafe internet to the Apache webserver, so without additional measures, this data could be easily intercepted.
- *The data at rest.* The MySQL database must be protected against unauthorized visitors. If a hacker succeeds nevertheless in breaking into the database, the privacy sensitive data must be unusable. As a DBA, we prefer to have no access to data that can be traced back to a specific person because we want to comply as much as possible with the GDPR legislation.

In the following sections, a number of possibilities are described to secure the data on all these layers.

[1]https://play.google.com/intl/nl_ALL/about/monetization-ads/ads/ad-id/

2.3.1 Securing the source code

The source code can be secured by means of code obfuscation. In the process of code obfuscation, an executable file is modified so that it can no longer be used by a hacker while it remains fully functional. This process changes the method-instruction or the metadata, but it does not change the output of a program.

During this process, an executable file is translated into code that is unreadable for humans. This obfuscated file is difficult for a hacker to use, but it remains fully functional since the output of the program is not changed.

The goal is to make it more difficult to reverse engineer the app, but provided sufficient time and effort, it is unfortenately not impossible.

2.3.2 Securing the data in transit

The privacy sensitive information sent from the mobile phone to the webserver must be protected against leaks. To ensure that this is done in a safe manner, one can use a VPN tunnel or SSL to avoid the traffic from being eavesdropped.

A *VPN tunnel* can be used to encrypt the connection between mobile phone and webserver. The traffic then flows through a VPN server. This principle is frequently used by companies to allow employees to log in from external networks other than the company network.
In the Android world there are a number of VPN providers[2], but one should then depend on an external party and this is therefore not for free. A VPN connection can be manually set on the mobile phone, but this is obviously not feasible for our application. There are also a number of interesting VPN apps, but these are also not useful for our application. Since a VPN connection has to be configured manually or a VPN app must be installed, a VPN tunnel is not usable to secure our data in transit.

Another way to secure data in transit is by using *Secure Sockets Layer*, a technique that is frequently used for web applications. Web applications handling sensitive information such as banking apps, use an HTTPS connection. This is an HTTP connection that uses an SSL certificate for communication between client and server, in our case mobile phone and webserver. An SSL certificate can be purchased from a Certification Authority. The role of a CA is to check companies and to confirm that a specific company is really who they claim to be. An SSL certificate is of course not for free and the price is generally an indication of the authenticity checks that have been carried out. For the use of an SSL certificate no additional configuration is required on the mobile phone, only on the webserver. Using SSL, the communication between mobile phone and webserver is encrypted.

2.3.3 Securing the data at rest

The *data at rest* is in our case the data stored in the MySQL database. According to Elovici's theory [Elo+04], securing a database can be split into 4 layers.

1. *Physical security*
   The preferred solution for storage is on physically separated machines.

2. *Operating system security*
   Different rights can be assigned to users who log on to the operating system (authorization).

[2]https://www.androidplanet.nl/tips/vpn-android-instellen-en-tips/

3. *Database Management System security*
   As with the OS, there is also authorization at DBMS level.

4. *Database encryption*
   When a hacker breaks through these three security levels, the database, backup files, metadata and log files can be accessed in plain text. Using encryption, it can be ensured that the data is stored in unreadable form.

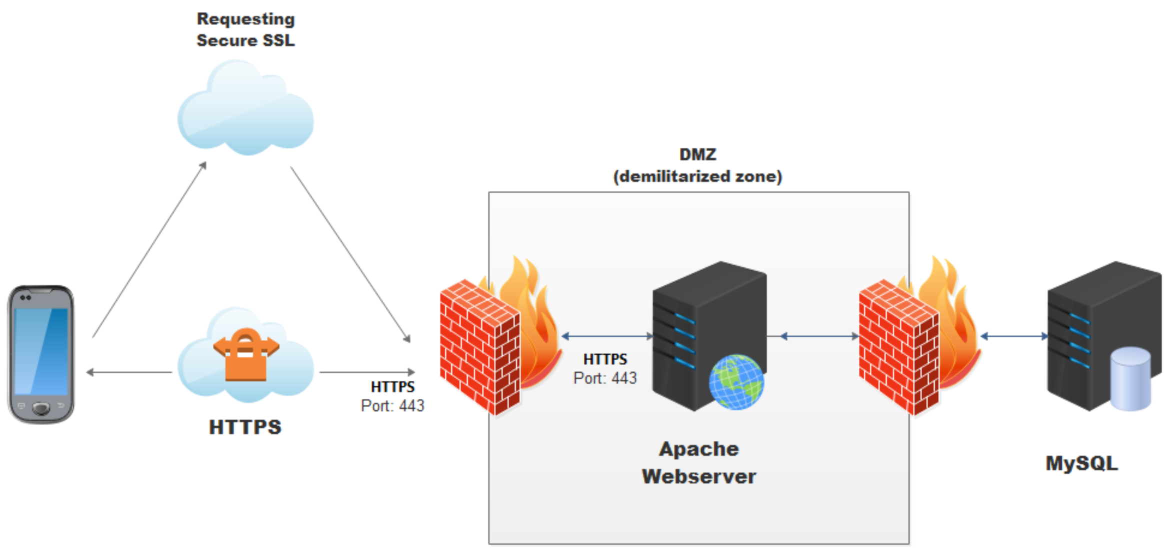


Figure 2.1: Physical security

Physical security

For this assignment we strongly recommend to install the webserver and database server on two physically separated machines. We also advise to put the webserver in the DMZ zone (demilitarized zone), the network segment between internal and external network. See figure 2.1

Operating system security

Different access rights / privileges can be assigned to the OS users. Varying from read only rights to full rights (system administrator).

Database Management System security

Different rights can be assigned to users who log on to the database. The DBA must have all rights for the management of the database and the database user with which the Android app connects to the database, must only have the rights that are strictly necessary.

Database encryption

We have investigated a number of options for encrypting the MySQL database. According to Wikipedia [3] and PrivacyPerspectief [4], several possibilities exist:

[3] https://en.wikipedia.org/wiki/Database_encryption
[4] http://privacyperspectief.nl/2017/06/01/encryptie-een-overzicht-van-verschillende-toepassingen/

- With *Full disk encryption*, the encryption engine is placed at storage management level. It encrypts all data that is stored and decrypts all data that is edited and changed. Every file is encrypted, including the temporary storage. This form of encryption protects the data only when it would get into the wrong hands due to theft of hardware.
- With *Instance based encryption*, the encryption engine is placed on the operating system that controls the storage. Instead of the server where the data is located, the encryption engine can also be placed on a separate server, a so-called proxy server. The latter is better, because an extra step is needed to obtain the key.
- With *File level encryption*, the system encrypts individual files or directories while in full disk encryption the entire partition or disk is encrypted. This does not happen automatically but there are certainly a number of advantages:
  - It is flexible since every file can be encrypted with a separate key. In this way, encrypted files can be individually managed.
  - The keys are only kept in memory as long as needed, hence as long as the file decrypted by them is held open.
  - It allows flexible user management since the encryption key is only available to users who are authorized for a specific folder or file.

  Transparent Data Encryption is an example of File Level Encryption and is employed by IBM, Oracle and Microsoft. TDE encrypts the database both on the hard drive and accordingly on backup media. The advantage of TDE is that the existing application does not have to be changed.
- With *Database encryption*, encryption is done by the database itself. This type of encryption can also be applied to a limited number of tables where the sensitive data is stored. This way of encryption protects to a limited extent against attacks by hackers, since the key is usually stored on the same medium as the database.
- With *Application level encryption*, the encryption engine is placed at application level. The data is already encrypted by the application before it is inserted into the database. The encryption is therefore done entirely by the application. Decryption of the data depends on the rights of the user accessing the application. Application Level Encryption offers sophisticated protection against unauthorized access, but it has a disadvantage. Since all records that are requested by a user have to be decrypted, it can cause performance problems. If the performance of an application is inadequate, this type of encryption is not a valid option. Database Encryption combined with Full Disk Encryption can provide sufficient protection in that case.

Instead of choosing encryption for securing the data at rest, one could also opt for *hashing* the privacy sensitive data. The main difference is that when using encryption, data still can be decrypted by anyone who possess the decryption key. Hashing on the other hand is a one-way function. Once the data is being hashed, it is practically impossible the recover the original data.

A hash function H is an algorithm that converts a message of arbitrary length M to a string of fixed length, namely *H(M)*, which can easily be stored in a database. A hash function is deterministic: the same value M always results in the same value *H(M).*

A hash function is collision-free if [5]:

[5] http://mathworld.wolfram.com/Collision-FreeHashFunction.html

- It is a one-way function. It should be impossible to reconstruct the original message *M* or even part of it for a given hash.
- It should be almost impossible to find different messages *M* and *M'* resulting in the same hash values *H(M) = H(M')*. Inputs that differ only in one point must have a completely different hash.

Although in theory hashes are not reversible, it is now practically feasible to use rainbow tables and brute force to derive the original messages from hashes.

2.4 Security decisions

To secure the *data in transit*, we ultimately opted for Secure Sockets Layer, since VPN tunneling requires a configuration per mobile device. We ensure a secure connection between mobile phone and webserver through HTTPS using a self-signed certificate. Because we do not want to buy a certificate, we opted for the self-signed version.
Drawback of this solution is that the end user receives a security warning. To avoid this warning, the security certificate is automatically accepted in the Android code, transparent to the user. Automatically accepting this certificate does not imply a security issue because the communication with the fingerprint app can only originate from our webserver.

To secure the *data at rest* we have considered database encryption for a long time. In the end we are not convinced of using encryption because of:

- Loss of performance: encryption and decryption is time consuming;
- Complexity of this technique;
- Key management, the keys must always be kept on a separate server;

Finally, we prefer using hashing. Attributes with privacy sensitive information can already be hashed in the Android code so that even the data processor does not know its contents and it is almost impossible to trace back the original value.
When the domain of original values is small, certain patterns in the hashed values could be derived. The attributes to be hashed are *phone number*, *user name*, *Local Ip Addresses*, *WiFi Ssid* and *WiFi Bsid*. None of these attributes has a small domain since many different values are possible. It is therefore almost impossible to deduce a certain pattern in the database based on these hashed values.
For the entropy calculation it is not necessary to know the original values, we only have to count the number of matching values. Since a hash function is deterministic, ie. the same value always yields the same hashed value, hashed values can be compared perfectly.
Wikipedia [6] states that hashing is a form of pseudonymisation and the GDPR therefore applies, see 1.3. In case of this project however, the data processor can no longer link the data to a specific person. We are balancing between pseudonymisation (GDPR applies) and anonymisation (GDPR does not apply). Nevertheless, we try to comply as much as possible.

[6] https://nl.wikipedia.org/wiki/Hashfunctie

Chapter 3

Implementation of security and privacy

3.1 Introduction

This chapter is rather practical oriented and it explains how we have further elaborated our security and privacy decisions as set forth in section 2.4. It is explained in general terms what steps we have taken to make the security and privacy aspect concrete. We focus on the rough lines and further details for the implementation can be found through the referenced links.

Section 3.2 contains an explanation of how we have set up an HTTPS connection with SSL for our system to protect the data in transit.

In order to protect our data at rest, we ultimately opted for hashing, as motivated in section 2.4. How we have practically implemented hashing for our assignment is elaborated in section 3.3.

3.2 Implementing HTTPS with SSL

A global explanation of what SSL entails is already given in section 2.3.2. To refresh our memory, we first briefly repeat the SSL characteristics.

SSL protects against message forgery, eavesdropping and theft and it is based on:

- *Encryption*: all transmitted data is encrypted so that fraudsters can not do anything with the intercepted data;
- *Data integrity*: it offers certainty that nobody has changed the transmitted data;
- *Authentication*: the client must know for sure that the server really is who he claims to be. The client has to verify the certificate, containing the public key of the server. Only when the client has checked and approved this certificate, a secure connection is set up.

To understand how an HTTPS connection has to be set up, we dig a bit deeper into the SSL principles. An SSL connection is established in 5 steps[1]:

1. The client visits a secure website and requests a secure connection via HTTPS.

[1]https://www.ibm.com/support/knowledgecenter/en/SSZHFX_9.1.0/securing/topics/sslauth.html

2. The server sends a X.509 certificate containing the public key. This public key is used by the client to encrypt the data that is sent to the server. Decoding is only possible with the private key and only the server ownes this key.

3. Website owners can register their keys towards Certification Authorities. The client can inquire via the Certification Authority whether the public key is indeed used by that specific server.

4. The sent messages from the server to the client can not be encrypted with the same public key, since the client can not decrypt them. This problem is solved as follows. The client generates a master key that is then encrypted with the server's public key. The encrypted master key is sent to the server and is then decrypted with the server's secret key.

5. From now, both parties possess the master key used for the SSL session. A secure connection is set up and data is encrypted.

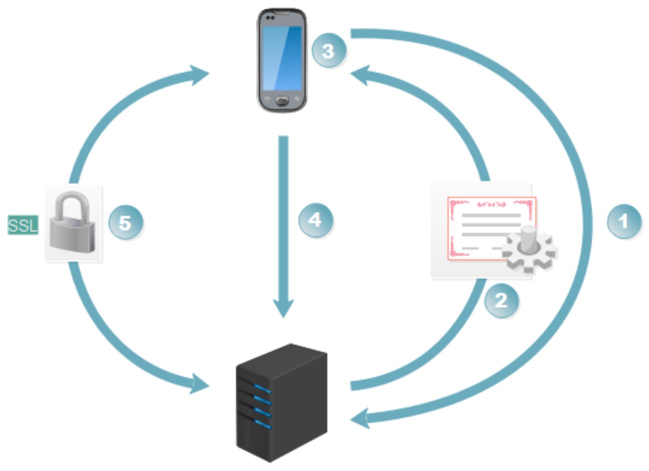


Figure 3.1: SSL handshake

In this project, the client is the mobile phone and the server is the Apache webserver. Since we have opted for a self-signed certificate, step 3 is no longer applicable. The mobile phone therefore does not check the validity with the Certification Authority, but the certificate is automatically accepted in the source code.

Creating a self-signed certificate on the webserver

We installed our webserver on an Ubuntu machine, but the installation principles are almost the same for other Linux distributions. The detailed configuration steps can be found on the internet[2]:

1. *Activation of SSL on the webserver*
   SSL comes standard in Ubuntu 16.04 and it only needs to be enabled.

2. *Creation of the self-signed certificate*
   After choosing a subdirectory within Apache's configuration hierarchy, the key and certificate can be created in that directory with the command:

[2]https://www.digitalocean.com/community/tutorials/how-to-create-a-ssl-certificate-on-apache-for-ubuntu-14-04

sudo ***openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout*** */etc/apache2/ssl/apache.key* ***-out*** */etc/apache2/ssl/apache.crt*

- **openssl**: command line tool of OpenSSL
- **req**: subcommand for signing request
- **-x509**: self-signed version
- **-nodes**: the key file is not password protected
- **-days 365**: the certificate's period of validity
- **-newkey rsa:2048**: a certificate request and a new 2048 bits private key are created at the same time
- **-keyout**: output filename for the new private key
- **-out**: output filename for the certificate

3. *Configuration of the Apache Webserver*
After the creation of the certificate and key, the Apache virtual host file has to be configured to use these files. The server admin, server name, server alias and document root must be adapted in the host file, as well as the location where Apache looks for the SSL certificate (apache.crt) and key (apache.key).

4. *Activation of the SSL Virtual Host*
The SSL Virtual Host has to be enabled after configuration.

5. *Testing*
At testing, a security warning appears in the fingerprint app because the identity of the webserver can not be verified, as it is not signed by a trusted CA. This problem does not occur with a purchased certificate. The preferred solution is to buy a good certificate but in this case, the certificate is bundled in the Android app and trusted at compile time.

A number of extra steps are needed for the implementation[3]:

1. *Converting .crt format to .pem format*
openssl x509 -inform der **-in** < *apache.crt* > **-outform** pem **-out** < *includesprivatekey.pem* >

2. *Creating the Network Security Config file*
The *network_security_config.xml* file must be created in the Android folder */res/xml* and it must point to the newly converted PEM file.

```
<?xmlversion = "1.0"encoding = "utf − 8"? >
< network − security − config >
< base − config >
< trust − anchors >
< certificates src = "@raw/includesprivatekey"/ >
< certificates src = "system"/ >
< /trust − anchors >
< /base − config >
< /network − security − config >
```

[3]https://medium.com/@johnmarktagle/using-self-signed-ssl-in-android-nougat-6d59593a3b6f

3. *Adding the Network Security Config file to the Android manifest*
   Each Android app has an AndroidManifest.xml file, residing at the root of the project source set. It contains essential information about the app[4].

```
<?xmlversion = "1.0"encoding = "utf − 8"? >
< manifest... >
< application
android : networkSecurityConfig = "@xml/network_security_config"
... >
...
< /application >
< /manifest >
```

After building and running the source code, the Apache webserver can be reached.

Implementation remark

For this project, we have created a self-signed certificate to test the SSL connection. The self-signed certificate solution only works from Android 7 Nougat (API 24), lower versions are not supported. In particular, the Android *networkSecurityConfig* tag is only supported from the Nougat version.

In practice this is not a problem, because when the Fingerprint app will actually be rolled out, no self-signed certificate will be used but a real certificate/domain must be purchased. With a real certificate, our solution works without any additional configurations of the certificates in the app. Since we do not want to purchase a certificate and our solution does work with a purchased certificate, we have removed the self-signed certificate from our server and we have continued working with *http* instead of *https*.

3.3 Hashing the privacy sensitive attributes

The hashing of the privacy sensitive attributes *phone number*, *user name*, *Local IP Addresses*, *WiFi Ssid* and *WiFi Bsid* is done in the Android code, before the data is sent through HTTPS. The privacy sensitive attributes are therefore first hashed and then encrypted for transport via HTTPS. The cryptographic hash function is SHA-256, which generates a 256-bit hash. SHA-256 is currently the most used hash function and the 256-bits key is a good compromise between safety and speed.

Salting is used, this is random data that is concatenated and processed with the value to be hashed. The salt is added to the attribute value, and it is hashed in its entirety (attribute value + salt). The salting adds extra security to prevent dictionary attacks and rainbow table attacks. Currently a fixed salt value is used, because we need to compare attribute values to calculate the entropy.

[4]https://developer.android.com/guide/topics/manifest/manifest-intro.html

Chapter 4

Information content of attributes

4.1 Introduction

An Android fingerprint gathered by the fingerprint app consists of values from about 90 different attributes. In this chapter we will explore the possibilities to extract information from the collected data and find a measure that indicates the unicity of a fingerprint. In other words: we will express the probability to match the fingerprint of an arbitrary device to the set of previously collected fingerprints.
A commonly used measure for the amount of information in an attribute is entropy. This concept was introduced in information theory by Shannon in 1948 [Sha48]. In the next paragraphs, entropy and its application in fingerprinting will be described in more detail.

4.2 Self-information

First, we explore how much information we can derive from the value of a single attribute in a fingerprint. The various attributes in the collected fingerprints are of different types and therefore convey different amounts of information. Some of those attributes can take only two values: e.g. the only possible values for the auto time zone setting are 'on' or 'off'. This type of attributes reveal little information, since probably about half of the devices will share the same setting. Other attributes may take more values: e.g. the most recent Android api version is 28, therefore this attribute can take at most 28 different values. Still other attributes can take many more values, like a wallpaper or a font list. Attributes like hardware identifiers (e.g. IMEI) or phone number are unique among all devices and therefore carry a lot of information.
In information theory the measure of information content in an event is called self-information or surprisal. This is expressed as:

$$I(x) = -log_2 p(x) \tag{4.1}$$

where x denotes the attribute value and p the probability of that value. Surprisal I is expressed in bits. Note that always $0 \leq p(x) \leq 1$, which means that $log_2 p$ has a negative value. The negation sign is used to convert $I(x)$ into a positive number.

The probability of any attribute value can be calculated from the collected fingerprint attributes in the database. It equals to the number of occurrences of that specific value in the database divided by the total number of fingerprints. When the probability of an attribute value is known, we find the surprisal by taking $-log_2$ of the probability.
As an example we calculate the surprisal of the collected auto-time setting with values 'off' and 'on', with $p(\text{'off'}) = 0.4$ and $p(\text{'on'}) = 0.6$. The surprisal of value 'off': $I(\text{'off'}) = -log_2 0.4 = 1.32$ bits. Value 'on' yields a surprisal $I(\text{'on'}) = -log_2 0.6 = 0.73$ bits. This example shows that the value with the lowest probability contains more information, it is more 'surprising' to find this

value.
The surprisal of a phone number is much higher, because each phone number is unique. If every human being on earth did have one phone number, we would be able to find a specific phone number in 7.6 billion[1] others. The surprisal of a phone number X is $I(X) = -log_2 P(X) = -log_2(1/7.600.000.000) = 32.8$ bits.

4.3 Calculation of entropy

To decide which attributes are useful in a fingerprint, it would be helpful to have a measure that indicates the total amount of information in an attribute. This is called entropy and is interpreted as the average number of bits of information for this attribute, considering all possible values. Entropy is calculated by multiplying the self-information of each value with its probability and then sum all the values. Formula 4.2 shows the calculation of entropy $H(x)$:

$$H(X) = \sum_{i=1}^{n} P(x_i) I(x_i) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{4.2}$$

We can now extend the example of the auto-time setting in the previous paragraph by calculating the total entropy for this attribute.

$$H(\text{auto-time}) = P(\text{'off'}).I(\text{'off'}) + P(\text{'on'}).I(\text{'on'}) = 0.4 * 1.32 + 0.6 * 0.73 = 0.966 \tag{4.3}$$

The auto-time attribute contains less than 1 bit of information and is in itself not very useful to recognize devices. On the other hand, the entropy of an attribute containing only unique values is much higher. If the database contains 50,000 fingerprints, the entropy of the phone number attribute will be

$$H(X) = -\sum_{i=1}^{50,000} P(x_i) \log_2 P(x_i) = -50,000 * 0.0002 * log_2 0.00002 = 15.61 \tag{4.4}$$

4.4 Acquiring information from a combination of attributes

Attributes having a high entropy carry more information and are therefore more suited to identify a device. Unique identifiers provide enough information to recognize every device fingerprint. But when unique identifiers are not available, we can combine the information from a set of attributes and try to identify a fingerprint.
Shannon already proved that the entropy of independent events may be added [Sha48]:

$$H(x, y) = H(x) + H(y) \tag{4.5}$$

In case of fingerprints it is not realistic to assume that all attributes are independent. Settings like language, date format and timezone are very likely to show some correlation. There are also less obvious correlations. Hardware types might correlate with location, because some brands of devices are more popular in a region than others. Device manufacturers use a limited set of hardware suppliers, therefore camera characteristics may correlate with screen dimensions. Another possibility is that each device model has a set of preset characteristics. If there is a group of users who never change the presets, the attributes in this set will not be independent.
In the case of correlated events x and y we can also calculate the entropy, provided the probability of event y is known under the condition of x. This conditional probability may be

[1]http://www.worldometers.info/world-population/

derived from the collected database data. According to Shannon the formula for calculating the entropy of correlated attributes is:

$$H(x, y) = H(x) + H(y|x) \tag{4.6}$$

Shannon also proved that the entropy of the combination of two dependent attributes is always lower than the entropy of independent attributes.

4.5 Data processing

4.5.1 Missing attribute values

In the previous paragraphs we implicitly presumed that all collected fingerprints can be used for entropy calculations. But what if some fingerprints are not complete because not all attributes can be collected? This may be the case when the user denies permission to collect certain attributes or the device lacks some features, e.g. tablets without a SIM card. There is also a difference between Android versions in the attributes that apps may access.
When calculating the entropy of an attribute, we cannot simply ignore the missing values. Suppose that half of the fingerprints do not have a value for a certain attribute. If we do not include the missing values count in the total, the probability of any value would be twice as high as in reality. This means we would end up with a calculated entropy value that is too high.
Therefore we will use the count of all values of an attribute for entropy calculation, including missing values.

We categorized the missing values in two groups: attributes that have no user permission and attributes that cannot be collected due to other reasons. Attributes without user permission are assigned the value "NO PERMISSION" and are part of the obtained fingerprint. The remainder of the attributes that cannot be collected are assigned value NULL to indicate they have no value.

4.5.2 Multiple fingerprints from the same device

We must also decide what to do when users upload multiple fingerprints. One solution is to block uploads completely after the initial fingerprint is submitted. This is not a valid option, because the collected data are valuable for analysis, they can be used to track changes in fingerprints and find out how often the values of attributes change. Besides, when users reinstall the app, they still can upload another fingerprint from the same device.

When multiple fingerprints from the same device are included in the entropy calculation, the entropy will be lower because we have more fingerprints but the same number of attribute values.
An example will clarify this:
Suppose we acquired ten fingerprints, containing an attribute with ten different values, the probability of each value therefore is 0.1. Application of equation 4.2 leads to $H = 10 * 0.1 \log_2 0.1 = 3.32$ bits.
Adding an extra fingerprint that matches a previous one changes the probabilities to nine values having $P = 0.091$ and one value having $P = 0.182$. Application of equation 4.2 now yields $H = 9 * 0.091 \log_2 0.091 + 0.182 \log_2 0.182 = 3.24$ bits.

Implementation: When the user uploads multiple fingerprints, we will include only the last submitted fingerprint in the entropy calculation. A device fingerprint is identified by a unique identifier (uuid), generated in the app during install time. Users can submit a fingerprint with a different uuid by reinstalling the app, so it is still possible that entropy is calculated with values from duplicate fingerprints. In the analysis phase after all fingerprint data are acquired, we may deduce which fingerprints originate from the same device and exclude those from entropy calculation.

Chapter 5

Matching fingerprints

5.1 Using entropy to select attributes

Entropy calculation can be used to find subsets of attributes that may uniquely identify a mobile device.
The first step in calculating the combined entropy of a fingerprint is to calculate the entropy of each single attribute. This is a straightforward application of equation 4.2.
Unique attributes like hardware identifiers will yield the maximum entropy. The absolute value of the entropy of unique identifiers depends on the number of collected fingerprints. For the remainder of this paragraph the unique attributes will be disregarded, because when included in any set of attributes, a unique attribute will always render the complete set unique. This is not compatible with our goal of identifying subsets of attributes that may be used as a fingerprint.

The second calculation step follows when we know the entropy of each attribute, we can select the most promising subsets and calculate their entropy. We must take into account that the selected attributes may not be independent, so we cannot simply add their entropies. It is necessary to calculate the correlation between attributes and then calculate the conditional entropy following equation 4.6.
In previous studies some identifying subsets of attributes are already put together. Wu et al. [Wu+16] found that a subset of 9 or 10 features is enough to recall over 99% of the devices. Kurtz et al. [Kur+16] established that by using a single feature, the Icon Cache, he could re-identify over 97% of the devices. Another set of two features, namely a subset of the list of installed apps combined with the device name was enough to re-identify over 96% of the iPhones.

5.2 Matching fingerprints

5.2.1 Classification of attributes

Which attributes are most suitable to identify a fingerprint? Attributes having a high entropy are good candidates, but we must also consider other characteristics. We propose to classify the attributes into 5 categories, starting with the most suitable category. The classification criteria are uniqueness, number of possible values and how often the value of the attribute will change.

1. Attributes that are unique and immutable will immediately identify a device. Hardware identifiers like the serial number of the device are in this category.

2. Attributes that are unique, but may be changed by the user. Subsequent fingerprints from the same device may show a different value for such an attribute, but fingerprints from other devices will never contain this value. Examples in this category are the UUID that is generated during installation of the fingerprint app, or the Android advertising ID.

3. Non-unique attributes that are not variable, like the device brand or the camera type. In itself, this kind of attribute is not sufficient for fingerprinting, because many devices will share the same value. But combined with other attributes the entropy may be enough to distinguish individual devices. Besides, when two fingerprints contain different values for one of these attributes, they will certainly originate from different devices.

4. Attributes that are not unique and frequently change value do not seem very suitable for fingerprinting. But if the attribute value changes in a predictable way, we might still extract some information from it. A few examples:
   - The Android version changes when the phone software is updated, but the new version will practically always have a higher version number. It is possible to return to the previous version, but very uncommon to do so.
   - The last boot time of a device will always be later than the previous boot time.
   - Battery level combined with time of last charging. If the latter did not change, battery level must be lower.

5. Attributes that are not unique but can contain many different values have a high entropy. We place many attributes that contain personal items in this category, e.g. the list of installed applications or the wallpaper. The entropy of these attributes is expected to be high, but the results of the data analysis will have to reveal which attributes are constant enough to be used in fingerprint detection.

6. Attributes with few possible values that may be changed by the user are probably useless for fingerprinting. E.g. device orientation can be horizontal or vertical, but changes often.

5.2.2 Matching algorithm

Matching fingerprints is easy when unique attributes are available. We can check if the uploaded fingerprint contains any attribute from category 1 or 2 that matches a value in the dataset. If a matching attribute is found, the fingerprint will originate from the same device in most cases. There are some exceptions: when the value of an attribute is 'null' or 'NO PERMISSION', the attribute cannot be used for matching. A hardware attribute like WIFI BSSID is protected on the latest Android versions and will return a constant value. The WIFI MAC address can be spoofed and is therefore not reliable. We must also be careful when it comes to attributes that originate from the SIM card because that card may be transferred from one phone to another. Therefore a match on the attributes of the SIM card alone is not conclusive.
In this thesis we are trying to match fingerprints without using the unique attributes, therefore they will be used only to check the matches obtained from the other attributes.

The following algorithm can be used to check if a fingerprint matches a previously uploaded one:

1. We limit the set of possible matches to the fingerprints with matching attributes in category 3, the invariable attributes like brand, model, screen resolution and camera-info.

2. Next, we remove the fingerprints where the attributes of category 4 do not match, e.g. the Android version is lower or the last boot time is earlier.
3. Depending on the outcomes of the entropy calculations we try to match fingerprints in the resulting set on attributes with high entropy. Particularly promising for identification are list attributes, that contain a lot of user-specific information. Comparing list attributes is not completely straightforward, as will be explained in subsection 5.3.

5.2.3 Implementation

The matching algorithm can not be fully implemented, because the choice of attributes depends on entropies that are only available after data collection and analysis.
The current code to check for matching fingerprints is programmed as stored procedures in the database, because a database is optimally equipped to perform selections on large quantities of data. Furthermore we do not know at this time which software will be used for data analysis, but by implementing the matching code in the database it may be reused by other programs. The result of the matching is available as soon as a fingerprint is submitted and may be incorporated in user feedback.

We created a stored procedure that takes a fingerprint_id as parameter and returns a set of possible matching fingerprint_ids based on steps 1 and 2 of the matching algorithm. The fingerprint to compare must already be added to the database. The procedure should be extended to limit the result set by other matching attributes.

5.3 List attributes

A special category of attributes are list attributes, such as lists of installed packages, music or contacts. List attributes have a high entropy because most lists are highly user specific. List attributes are also very variable, they change often over time when a user installs or removes apps, adds contacts or music.
To calculate the entropy of an attribute, we need to count the number of entries for each value. Counting entries poses a challenge when it comes to list attributes. When can we conclude that two lists originate from the same device? When both lists contain exactly the same entries the conclusion is simple, in that case the lists are identical. But what happens when most of the entries occur in both lists, with only a few exceptions? When the user removed or added an item, can the list attribute still be matched to previous entries?

A number of methods have been developed to compare strings and assign a value that indicates the similarity of two strings. A list with entries can be transformed into a string by sorting the entries first and then concatenating them. Sorting the list entries ensures that the comparison method does not have to take transitions into account.
A frequently used string comparison method is the edit distance [Nav01], which is the number of operations that is needed to transform one string into another. The most used implementation of the edit distance is the Levenshtein distance, where the operations are insertion, removal or replacement of a character in the string. Other implementations use other operations to calculate the edit distance. The Hamming distance e.g. uses only character substitution, which means that only strings of similar length can be compared.

Is the edit distance applicable for comparing the similarity of two attribute lists? Ordering and concatenating the list is simple, but what would the result of the comparison mean? Suppose we have two lists that differ in only one item. The comparison by edit distance would

yield a different result depending on the length of the item name.
A second reason not to use any form of edit distance for comparison is that list item names may be different by only one or a few letters, which would lead to a high similarity score, while in reality the list entry is a different one.

From the above examples it is clear that edit distance is not reliable to compare list attributes, mainly because the method uses the string as a whole and does not take into account that the attribute lists consist of items. What we need is a comparison method that tells us how many items the two lists have in common and how many appear only in one of both lists. The criterion whether two entries are similar is simple: their text must be exactly the same.

A simple measure for set comparisons is the Jaccard index, which is the percentage of similarity in the set. Formula 5.1 shows the calculation method for two sets A and B:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

A similar method is the Sørensen–Dice coefficient S, expressed in formula 5.2:

$$S(A,B) = \frac{2|A \cap B|}{|A| + |B|} \tag{5.2}$$

Both the Jaccard index and the Sørensen–Dice coefficient range from 0 (no similarity) to 1 (2 identical sets). The two indices can be converted into each other via $J(A,B) = S(A,B)/(2 - S(A,B))$. The Sørensen–Dice coefficient is higher than the Jaccard index, because the number of matching set items is counted twice, thus emphasizing the similarities between the sets. We choose the Jaccard index to work with, but the Sørensen–Dice coefficient is also usable.

Now that we have established a measure for similarity, the next step must be to find out what a resulting value means. Given two attribute lists with a high Jaccard index, how can we infer whether they originate from the same device?
If we have another means of identifying fingerprints, like a unique ID, it may be possible to derive a threshold value for the Jaccard index. This can only be done after an amount of data is collected. If we have successive fingerprints from the same device, we can calculate the Jaccard index of their list attributes. We can also calculate the Jaccard distance of attribute lists that originate from different devices. If there is a reliable difference between the two indices, we have found a way to use partial matches in list attributes to identify fingerprints.
If there is no difference between Jaccard indices from the same or from different fingerprints, there is no way to conclude if two attribute lists originate from the same device, which makes such an attibute useless for device fingerprinting.

5.4 Android versions and accessibility of attributes

Google tries to improve the data protection with each new Android version. Android Marshmallow (API 23) and higher versions block programmatic access to the device's local hardware identifiers for apps using the Wi-Fi and Bluetooth APIs. The WifiInfo.getMacAddress() and the BluetoothAdapter.getAddress() methods return a constant value of 02:00:00:00:00:00. Soon after the release, workarounds appeared on the Internet where the Wi-Fi MAC Adress is derived from the IPv6 address. Until recently, the Wi-Fi MAC adress could be obtained this way with only ACCESS_WIFI_STATE permission. This is a normal permission and therefore automatically granted, users can not block this access by denying this permission. Only in the latest Android version (API 27) does it take location permission to access Wifi characteristics. The Bluetooth MAC address could still be retrieved using reflection, but starting from Android Oreo (API 26) it will return a random value.

Other unique identifiers like IMEI and IMSI need READ_PHONE_STATE permission in all our targeted Android versions, which means that the security of those items is not improved at all.
Android Advertising ID is explicitly designed to track devices, which means it will remain freely accessible. Users may change their Advertising ID, but cannot block apps from accessing it.

Chapter 6

Results

6.1 Software Requirements

After consulting with our supervisor, we decided that the fingerprinting application should meet the following requirements:

- The system collects attribute values from Android devices that can be used for fingerprinting. At first the list that was already identified by Jonker et al. [JF17], afterwards extra attributes may be added if they look promising for identifying devices.
- Provide a context for the user, offer some incentive to download and use the app. The user is informed about the research goal, the incentive can be feedback about the collected attributes and the extent of privacy loss.
- The app user can control which types of attributes are collected by granting or revoking permissions.
- The user can view the collected data and then decide if his data will be uploaded to the database.
- Special attention must be paid to privacy, the application must comply with the new EU General Data Protection Regulation (GDPR). This means that all personal data must be protected.
- The system must be secure, third parties may not have access to collected data.
- The system must be scalable, because the data will be collected in a short period of time. Busy moments are expected if a group of users send their fingerprints at the same time.

A nice to have feature is feedback: can we show the user how much privacy is lost by giving special permissions or what measures can be taken to prevent loss of privacy?

6.2 Application architecture

6.2.1 Process flow

In figure 6.1, the architecture of the fingerprint application is explained. The data collection takes place on the Android device. The user can inspect the data and can opt to send it to the backend, but he is not forced to do so.
The data is then forwarded to the REST API on the webserver as a JSON object. Next, the software program on the webserver parses the JSON object, connects to the database and inserts the data.

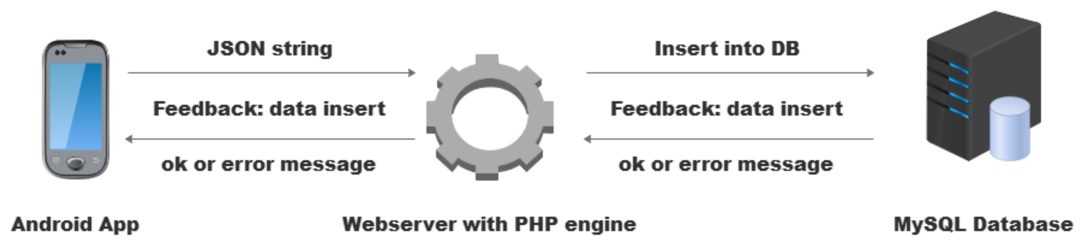


Figure 6.1: Architecture of the Fingerprint app

The REST API returns feedback to the user to report whether the data has been inserted correctly or to report that an error has occurred. If the data upload succeeded the app displays a notification for the user, containing the number of fingerprints that were submitted in total and how many fingerprints originate from the currect device.

6.2.2 Design decisions

When it comes to Android programming, there are a few options. We chose to use Java, because it is the official Android programming language and it is also part of the curriculum at the Open University. Google offers a complete and free Integrated Development Environment (IDE) for the Android platform, named Android Studio. We used it to develop our Android code.
The language of our choice for server-side developing is PHP. We chose a PHP engine because it is an open source server-side scripting language. Furthermore, PHP is widely used for website development and it is compatible with Unix-based operating systems. PHP is not a complex scripting language, we had no experience with it, but we were able to set up the PHP engine in a limited amount of time. Moreover, the PHP engine makes optimal use of the Apache webserver and the MySQL database.

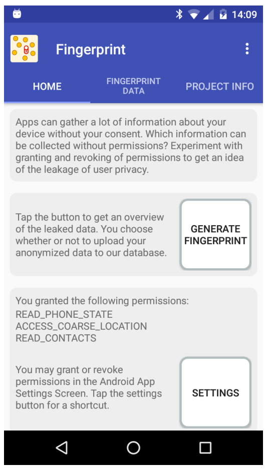
For the database we chose the *MySQL Community Edition*. It is free to download and it is the most used open source database. MySQL is a relational database that works optimal for web applications. MySQL also offers a good performance, availability and scalability (in the range of free open source databases).
The fact that MySQL supports the *LAMP stack* (Linux as OS, Apache as webserver, MySQL as database and PHP/Perl/Python as scripting language) makes these components work well together.

For the data exchange between device and webserver we chose JSON over XML, because the JSON messages are a bit smaller and easier to read than XML. Both Java and PHP offer easy-to-use libraries for composing and parsing JSON strings.

6.3 Android

6.3.1 User interface

The Android app is the part of the system the user is interacting with. Figure 6.2 shows the opening screen of the app, where the user sees a short introductory text and, depending on the Android version, one or two buttons. The first button generates a fingerprint and then shows a second screen to present the data. The second button is only visible on devices running



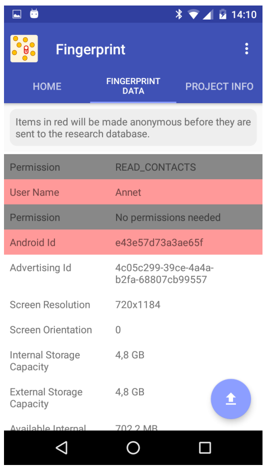
Apps can gather a lot of information about your device without your consent. Which information can be collected without permissions? Experiment with granting and revoking of permissions to get an idea of the leakage of user privacy.

Tap the button to get an overview of the leaked data. You choose whether or not to upload your anonymized data to our database.

You granted the following permissions:
READ_PHONE_STATE
ACCESS_COARSE_LOCATION
READ_CONTACTS

You may grant or revoke permissions in the Android App Settings Screen. Tap the settings button for a shortcut.

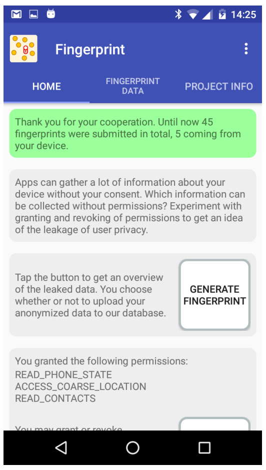
Figure 6.2: Start



Items in red will be made anonymous before they are sent to the research database.

Permission	READ_CONTACTS
User Name	Annet
Permission	No permissions needed
Android Id	e43e57d73a3ae65f
Advertising Id	4c05c299-39ce-4a4a-b2fa-68807cb99557
Screen Resolution	720x1184
Screen Orientation	0
Internal Storage Capacity	4,8 GB
External Storage Capacity	4,8 GB
Available Internal	702,2 MB

Figure 6.3: Data



Thank you for your cooperation. Until now 45 fingerprints were submitted in total, 5 coming from your device.

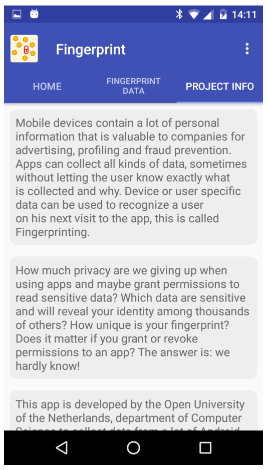
Apps can gather a lot of information about your device without your consent. Which information can be collected without permissions? Experiment with granting and revoking of permissions to get an idea of the leakage of user privacy.

Tap the button to get an overview of the leaked data. You choose whether or not to upload your anonymized data to our database.

You granted the following permissions:
READ_PHONE_STATE
ACCESS_COARSE_LOCATION
READ_CONTACTS

You may grant or revoke

Figure 6.4: Feedback



Mobile devices contain a lot of personal information that is valuable to companies for advertising, profiling and fraud prevention. Apps can collect all kinds of data, sometimes without letting the user know exactly what is collected and why. Device or user specific data can be used to recognize a user on his next visit to the app, this is called Fingerprinting.

How much privacy are we giving up when using apps and maybe grant permissions to read sensitive data? Which data are sensitive and will reveal your identity among thousands of others? How unique is your fingerprint? Does it matter if you grant or revoke permissions to an app? The answer is: we hardly know!

This app is developed by the Open University of the Netherlands, department of Computer Science to collect data from a lot of Android

Figure 6.5: Info

Android 6 or higher, it opens the Settings screen, where the user can review and change the permissions for the app.

The 'Fingerprint Data' screen (figure 6.3) is initially empty, only after the first fingerprint is generated it shows a scrollable list containing the collected data. The privacy-sensitive values are shown in red and will be hashed when the fingerprint is uploaded. A message presenting this is also displayed at the top of the screen. When the collected fingerprint data are presented to the user, the upload button is activated en stays in view even when the user swipes to another screen. Clicking the upload button shows a short explaining text and the definitive upload button.
After the upload, the user is directed to the opening screen, that now contains a green message indicating that the fingerprint was received correctly. The user is informed how many fingerprints are uploaded in total and how many are coming from his own device. If the upload did not succeed a red error message is presented.
Finally, the 'Project Info' screen (figure 6.4) contains an explanation of the goals of the app and the privacy statement.

6.3.2 Android versions

Table 6.1: Android versions market share May 2018

Name	Version	Release date	API level	Market share
Oreo	8.0 - 8.1	August 21, 2017	26 - 27	5.7%
Nougat	7.0 - 7.1.2	August 22, 2016	24 - 25	31.1%
Marshmallow	6.0 - 6.0.1	October 5, 2015	23	25.5%
Lollipop	5.0 - 5.1.1	November 12, 2014	21 - 22	22.4%
KitKat	4.4 - 4.4.4	October 31, 2013	19 - 20	10.3%
Older	1.0 - 4.3		1 - 18	5.0%

Table 6.1 shows the market share of the most used Android versions[1]. The majority of the

[1]https://developer.android.com/about/dashboards/index.html

devices use Lollipop, Marshmallow or Nougat, while the latest version Oreo is not yet spread widely. Version KitKat was released over four years ago but is still present on 10% of the devices.
We will therefore target KitKat as the lowest version that our fingerprint app is designed for. The market share of older versions is declining and will be even lower by the time the fingerprinting app will be actually used.
Not all Android versions offer the same possibilities to collect data. Older attributes are deprecated and newer versions offer new commands. As a result, not all fingerprints will contain the same attributes.

6.3.3 Android permissions

Users of the Android OS can prevent apps from accessing sensitive data, by setting permissions. Apps may require permissions e.g. to read the contacts list or access the internet. The Android manufacturer (Google) distinguishes two classes of permissions:

- PROTECTION_NORMAL permissions don't pose a threat to user privacy or the operating of the device. When apps ask for a PROTECTION_NORMAL permission, it is granted automatically. Examples are INTERNET permission, SET_ALARM permission or ACCESS_WIFI_STATE. Users are not able to revoke those permissions.
- PROTECTION_DANGEROUS permissions like READ_PHONE_STATE or ACCESS_COARSE_LOCATION are considered dangerous. Users must explicitly grant this type of permissions.

Permissions are organized in groups, related to the capabilities and features of a device. The PHONE permission group contains all permissions that are related to phone functionality, such as READ_PHONE_STATE, CALL_PHONE and WRITE_CALL_LOG. Android permissions are granted on group level, which means that when an app is granted one permission from a group, all other permissions from that group are granted also. A user that permits an app to READ_PHONE_STATE, also permits the app to make phone calls.

When programming, we need to take permissions into account. All permissions that an app wants to use, should be stated in the manifest, which is an initialisation file. Depending on the Android version, there is a different way of asking for permissions. The versions prior to Android 6 (Marshmallow) asked for all necessary permissions during installation of the app. Revoking permissions afterwards was not possible and could be achieved only by reinstalling the app.

Starting with Android 6, apps ask permission when they want to use a protected functionality. Users may grant permissions, but they can revoke them via the settings screen. This means that apps must check that the necessary permissions are granted every time they access a protected resource. If access is not permitted, then ask for that permission. Google added the possibility to add a rationale when asking for permission, to explain to users why the app is asking for this specific permission. Programmers also need to take care that an app should not crash when necessary permissions are not granted.

6.3.4 Implementation

Android classes

Figure 6.6 shows the main classes of the Android app. Every Android app must contain at least one instance of the Activity class. The app contains one Activity, called MainActivity,

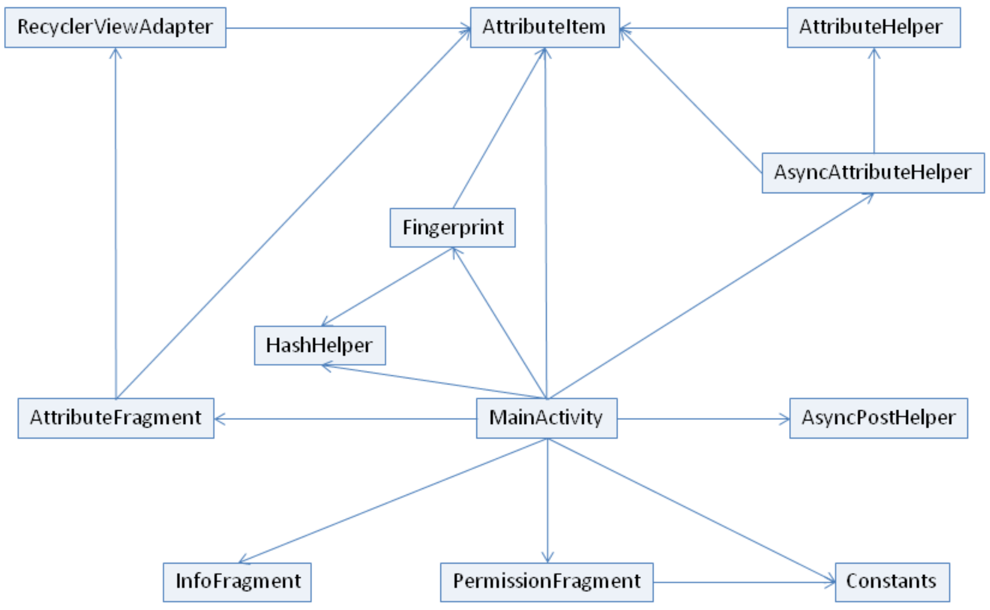


Figure 6.6: Android app class diagram

that creates three Fragment instances, one for each user screen. The Fragments contain the logic to show the user interface.
All code to collect the fingerprint attributes is contained in the AttributeHelper. Most attributes cannot be collected on the main thread, therefore we created an AsyncAttributeHelper that starts a new thread.
After the attributes are collected, they are sent back to the MainActivity, which in turn sends the attribute list to the AttributeFragment. The sole purpose of this fragment is to present a scrollable list with data to the user. The RecyclerViewAdapter is a specialized class that ensures the optimal use of memory.

When the user chooses to upload the collected data, the Fingerprint class converts the data to a JSON string that is sent to the database by the AsyncPostHelper.
Finally the Constants class holds the list of attributes to collect. Each attribute is identified by a unique number and the display name in the string resources file references the same number. The numbers contained in the Constants class also correspond with the id of the 'attribute' table in the database. As a consequence, when adding a new attribute, the id is added to the Constants class, the screen text is added to the string resources file and a new record is added to the database.

Attribute values are acquired in three different ways:

- Read out Android attributes via Java code.
- Execute commandline instructions to get values of system attributes.
- Creating a webview, which is a simple browser window in the app, and collect browser characteristics.

The Android code has extensive comments, but contains a few constructions that need further explanation. These will be covered in the next subsections.

Asynchronous tasks

Some tasks may not be executed on the main thread, because they take too much time and will block user interaction. Java will not compile synchronous code to access certain attributes. To maintain a uniform way of collecting attributes, all attributes are collected in a background thread. To this end the AsyncAttributeHelper class was created, subclassed from the standard AsyncTask. Two methods are overridden:

- doInBackground(Params...) calls the AttributeHelper class to do the actual collecting of attributes.
- onPostExecute(Result) transfers the resulting attribute collection back to the main thread. It specifies the method to execute in MainActivity: processFinish().

User permissions

From Android 6 Marshmallow (API 23) user permissions are no longer requested at installation time, but may be granted and revoked while running an app. This implies that it is always necessary to check if a permission is granted before executing a priviliged command. If the permission is not granted, the app may ask for permission at that point.
The Android flow for asking permissions also uses asynchronous methods.

Permissions are implemented in method MainActivity.requestPermission(), this method takes a permission name and a requestCode as parameters. The first action is to check if the permission is already granted, if that is the case, we can collect the attributes with that permission. If the requested permission is not yet granted, we issue an asynchronous requestPermissions command. The result of this call is obtained by overriding the OnRequestPermissionsResult() method, where we can check if the permission is granted and execute the proper methods according to the result.

Getting data from a Webview

Android Webview is a component that allows apps to display content from the web directly inside an application and it can also use Javascript. The app employs a webview to get the http user agent and a canvas fingerprint. The Webview is created in PermissionFragment, but

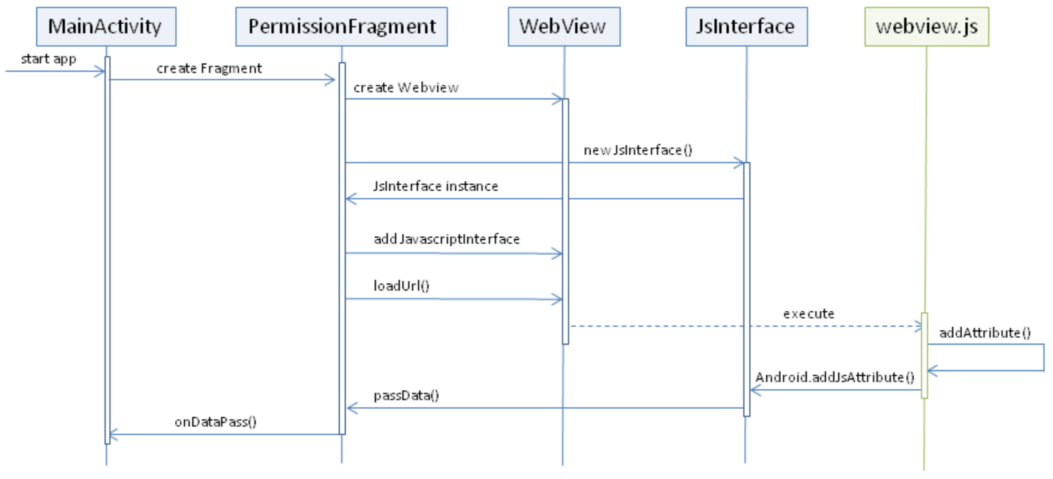


Figure 6.7: Webview sequence diagram

it is hidden from the user because there is no interaction. From the asset directory it loads the file 'webview.html', that contains a link to a Javascript file to get the canvas fingerprint. The Javascript file also contains a method addAttribute() that calls an Android procedure to transfer the attributes to Android. Figure 6.7 shows how the attribute values are calculated in Javascript and transported back to the MainActivity, which is responsible for showing the attribute values.

Localization

In order to reach a large audience, the user interface language of the app is English. To take into account future translations, all texts are saved in a string resource file *res/values/strings.xml*. If another language should be added, it is sufficient to copy the string resource file, translate the texts and put it in *res/values-**xx**/strings.xml*, where xx denotes the country code. Android handles localization automatically based on the device default langauage.

Hashing privacy-sensitive data

Values of privacy-sensitive attributes are hashed before they are uploaded to the database. The safe way to hash a value is by applying a random salt for each hash and then prepend the salt to the hash. In our case this is not applicable, because although we do not need to know the value of the attribute, we do need to recognize a value for entropy calculation. We therefore apply a constant salt value when hashing.

Extending the application

In the current app version all attributes are hardcoded. This means that attributes can only be added or changed by distributing a new version. It would be nice if the attribute list could be extended by downloading new instructions from the backend. This is feasible for attributes that can be collected via the commandline.
Attributes like directory structure are at the moment acquired by starting a commandline process and executing a predefined string. The app could retrieve a script from the server, that contains a list of attributes, their codes and the commands to execute. Showing the attribute values to the user and uploading to the database doesn't have to change.

6.4 PHP

The database is addressed through a REST API. REST (REpresentational State Transfer) is an architectural style for developing web services[2]. The API takes the request and inserts the data into the database, then the API sends the response back to the user (succeeded or error message).

The JSON format is used for forwarding the attributes of the fingerprint app to the webserver. JSON (Javascript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate[3].

The JSON string used in this project is structured as follows:

{*"attributes":*{}*,"timestamp":"20171220T11:05:42.120+0100","uuid":"ae36394b-36ef-48c2-9150-9cbd81a827b4"*}

[2]http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer
[3]https://www.json.org/

The value of *"attributes"* is a *key/value* list like [{*"attribute_id"* : *"attribute_value"*}], the value of *"timestamp"* is the moment at which the fingerprint is inserted and *"uuid"* is a unique number attributed at installation time of an app. The *"uuid"* only changes when the app is being reinstalled.

The PHP directory also contains some files to show a dashboard during data collection. The dashboard shows the number of fingerprints, how many of them are unique and the maximum possible entropy. The entropy values for all attributes are shown in a chart. The PHP files are contained in the web root directory of Apache */opt/lampp/htdocs/*:

- *fingerprint/create.php*
  The fingerprint app calls *create.php*, in this file the posted data from the app is being processed. *Create.php* includes the files *database.php* and *fingerprint.php.*
- *fingerprint/getentropy.php*
  A script to get the entropy data from the database.
- *fingerprint/dashboard.php*
  Shows a dashboard containing the number of fingerprints and the entropy. Used for the presentation, can be extended to monitor the uploads during the data collection phase.
- *config/database.php*
  Database.php provides a connection to the database. This file contains the *getConnection()* function that returns a database connection.
- *objects/dbprocs.php*
  This wrapper for database procedures exposes functions to get the number of fingerprints, the maximum entropy and the number of previously uploaded matches for a given fingerprint.
- *objects/entropy.php*
  Getentropy.php includes *entropy.php. Entropy.php* contains the query that finally must be executed to request the entropy.
- *objects/fingerprint.php*
  The *fingerprint.php* file decodes the posted data from the fingerprint app (JSON string) into PHP and contains queries to insert the fingerprint into the database.
- *objects/phpgraphlib.php*
  Library to show a graph on the dashboard.
- Folder *js* contains Javascript libraries needed for the entropy graph on the dashboard.

6.5 MySql

The *"attribute"* table holds all attributes, the primary key *"id"* corresponds with the Constants-list in the app. The permissions and permission group that an attribute belongs to are contained in separate tables. The *"category"* table holds the type of the attribute. As the set of attributes is fixed in the Android app, the contents of these tables will not change during the data collection period.
The collected data are saved in two tables. For every fingerprint that is submitted, one record is added in the *"fingerprint"* table. The *"uuid"* field in the *"fingerprint"* table contains a unique indentifier generated on the device. The collected attributes are saved in the *"attributevalue"* table, which is a link table to *"fingerprint"* and *"attribute"*.

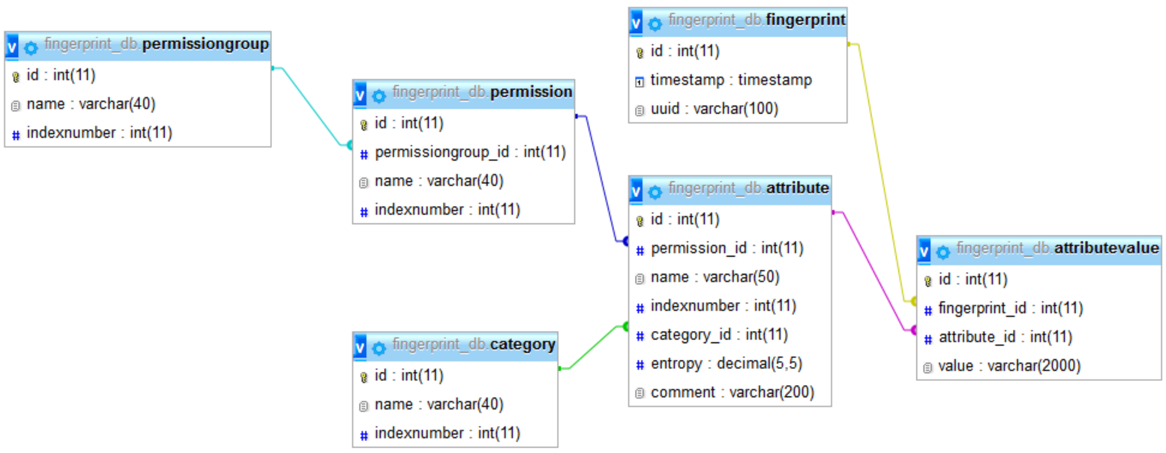


Figure 6.8: Database schema

To calculate the entropy, a view *"v_entropy"* was created. This view uses a helper view, namely *"v_attvalues_count"*, which counts the number of values per attribute. The entropy calculation is based on the attribute values of the last submitted fingerprint per device. The results of the view can be seen directly in phpMyAdmin or on the dashboard page in PHP.

6.6 Application tests

The application is tested manually, by installing the Fingerprint app on various devices and checking the results that were uploaded in the database.
The following characteristics were tested this way:

- Type of device: phone, tablet, emulator
- SIM card: available or not
- Android versions 5, 6 and 7
- Charactersets containing diacritical characters or Chinese characters

A small performance test was also executed with a test tool called Postman, which is a free download. The test script posts 2500 fingerprints to the API, at a rate of 10 per second. The average response time was 92.4 ms, and only two responses took over 200 ms.
From this we conclude that the database and API are fast enough to accomodate multiple requests per second.

Chapter 7

Conclusions and Discussion

The result of this bachelor project is an Android app to collect fingerprint data, store it in a database and give a little feedback to the user. The main purpose of the app is to collect data for calculating the privacy loss of various attributes. The app is focused on collecting attributes that could be useful to recognize a device. The actual collection of data was not part of the project.

During development we focused on two subjects:

- Privacy and security
- Entropy and fingerprint matching

In the event of data breach, privacy sensitive data like user name or phone number must be unusable. Furthermore, we must comply to the GDPR legislation: in case of data leak, it must be impossible to link the data to an 'identifiable natural person'.
Hashing is an excellent technique to meet these requirements. The advantage of hashing is that entropy calculations can still be performed. A hash function is deterministic (the same value always yields the same hashed value) and we must only be able to verify whether values are equal but it is not necessary to know the exact content of these values.

Privacy and security are closely linked. Privacy means that data must remain secret, while security is the technical implementation of it. To secure the data, we came up with an approach to study the 3 layers of our application: securing the *source code*, the *data in transit* and the *data at rest*.

- To secure the *source code*, we advise code obfuscation. This solution is not 100% conclusive, but it does offer protection up to a certain point.
- To secure the *data in transit*, we opted for Secure Sockets Layer. VPN tunneling was not an option since it requires a configuration per mobile device.
- Securing the *data at rest* is achieved through hashing.

Entropy is a measure that indicates how much information is available in the collected fingerprints. The entropy is higher as an attribute contains more distinct values. Attributes with high entropy and little variability offer the best use for fingerprint recognition. Attribute entropy may be added to calculate a combined entropy, provided that the attributes are not correlated.

Users may upload more than one device fingerprint, therefore we calculate the entropy based on a subset of the uploaded fingerprints. This subset contains only the last submitted fingerprint from every device.

Apart from the unique attributes, list attributes provide a high entropy and thus are canditates to be used for fingerprinting. Many lists are highly user specific, but they do vary over time, so we cannot use an exact comparison. The Jaccard Index is a measure to express how many items two lists have in common and is well suited for this type of partial matching.

The dynamic nature of software implies that there is always room for improvement. More attributes can be collected, especially when the goal is not only to recognize devices, but also profiling. For profiling, the more variable attributes may be interesting.
The user interface could be fancier, with more extensive feedback to the user, although the most interesting results will only be available after analysis of an extensive data set.
Before data collection is started the security measures should be extended, because the server scripts do not yet test the origin or the validity of the uploaded data.

Bibliography

[Eck10] Peter Eckersley. “How Unique Is Your Web Browser?” In: *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*. Ed. by Mikhail J. Atallah and Nicholas J. Hopper. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–18. ISBN: 978-3-642-14527-8. DOI: 10.1007/978-3-642-14527-8_1. URL: https://doi.org/10.1007/978-3-642-14527-8_1.

[Elo+04] Yuval Elovici et al. “A structure preserving database encryption scheme”. In: *Workshop on Secure Data Management*. Springer. 2004, pp. 28–40.

[JF17] Hugo Jonker and Christof Ferreira Torres. “Craving for Cookies: Commercial Fingerprinting in Mobile Apps”. In: *Manuscript submitted for publication* (2017).

[Kur+16] Andreas Kurtz et al. “Fingerprinting mobile devices using personalized configurations”. In: *Proceedings on Privacy Enhancing Technologies* 2016.1 (2016), pp. 4–19.

[MS12] Keaton Mowery and Hovav Shacham. “Pixel perfect: Fingerprinting canvas in HTML5”. In: *Proceedings of W2SP* (2012), pp. 1–12.

[Nav01] Gonzalo Navarro. “A Guided Tour to Approximate String Matching”. In: *ACM Comput. Surv.* 33.1 (Mar. 2001), pp. 31–88. ISSN: 0360-0300. DOI: 10.1145/375360.375365. URL: http://doi.acm.org/10.1145/375360.375365.

[Sha48] Claude Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27 (1948), pp. 379–423.

[Wu+16] Wenjia Wu et al. “Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers”. In: *IEEE Access* 4 (2016), pp. 8073–8083. DOI: 10.1109/ACCESS.2016.2626395. URL: https://doi.org/10.1109/ACCESS.2016.2626395.

Appendices

Appendix A

List of collected attributes

Additional information about these attributes can be found at Android Developer[1].

Table A.1: Collected attributes.

Attribute	Permission	Summary
Identifiers		
AndroidId	None	From API level 26, a 64-bit number (expressed as a hexadecimal string), unique to each combination of app-signing key, user, and device.
AdvertisingId	None	The advertising ID is a unique, user-resettable ID for advertising, provided by Google Play services.
FacebookId		
Carrier		
SimOperatorName	None	The Service Provider Name.
SimOperatorCountry	None	The Service Provider Country.
NetworkOperatorName	None	The alphabetic name of the current registered operator.
NetworkOperatorCountry	None	The country of the current registered operator.
MsisdnPhoneNumber		The phone number.
Imsi		IMSI number of the device, a unique 64-bit number to identify a user in a cellular network[2].
SIMSerialNumber	READ_ PHONE_ STATE	The serial number of the SIM.
Settings		
AutomaticTimeSync	None	Value to specify if the user prefers the date, time and time zone to be automatically fetched from the network (NITZ). 1=yes, 0=no.
		Continued on next page

[1]https://developer.android.com/
[2]https://en.wikipedia.org/wiki/International_mobile_subscriber_identity

Table A.1 – continued from previous page

Attribute	Permission	Summary
AutomaticTimezone	None	Value to specify if the user prefers the time zone to be automatically fetched from the network (NITZ). 1=yes, 0=no.
TimeScreenLocking	None	The amount of time in milliseconds before the device goes to sleep or begins to dream after a period of inactivity.
Notification WifiAvailability	None	Whether a notification is shown when wifi is available
PolicyWifiSleeping	None	The policy for deciding when Wi-Fi should go to sleep (which will in turn switch to using the mobile data as an Internet connection).
LockPatternEnabled	None	Whether lock pattern is used to secure device
PhoneUnlockingVibration	None	Whether lock pattern will vibrate as user enters (0 = false, 1 = true).
SoundEffectsEnabled	None	Whether the sounds effects (key clicks, lid open ...) are enabled.
ShowingPassword InTextEditor	None	Setting to showing password characters in text editors.
ScreenBrightnessMode	None	Control whether to enable automatic brightness mode.
DeviceOrientationLocked	None	Control whether the accelerometer will be used to change screen orientation.
CurrentWallpaper	None	If the current wallpaper is a live wallpaper component, return the information about that wallpaper.
ListDefaultRingtones	None	Returns the list of ringtones.
InstallNonMarketApps	None	The ability to install apps from other sources than Google Play
GrantedPermissions		
ListUserAccounts	None	
HttpProxySettings	None	Http proxy settings
SystemVolume	None	Info about system volume.
Localisation		
LocaleCountry	None	Returns the country/region code for this locale, which should either be the empty string, an uppercase ISO 3166 2-letter code, or a UN M.49 3-digit code.
LocaleCurrency	None	Local currency.
LocaleLanguage	None	Local language.
TimeZone	None	Local time zone.
TimeDateFormat	None	Date format string mm/dd/yyyy dd/mm/yyyy yyyy/mm/dd. Display times as 12 or 24 hours 12 24.

Continued on next page

Table A.1 – continued from previous page

Attribute	Permission	Summary
Geolocation	None	
GsmCdmaCidLac	None	
Device		
Imei	READ_ PHONE_ STATE	Returns IMEI for GSM.
Hardware		
ScreenResolution	None	The absolute width and height of the available display size in pixels.
ScreenOrientation	None	Returns the rotation of the screen from its natural orientation. The returned value may be 0, 1 (90 degrees), 2 (180 degrees) and 3 (270 degrees).
InternalStorageCapacity	None	Total number of blocks on the file system $*$ the size, in bytes, of a block on the file system of the user data directory.
ExternalStorageCapacity	None	Total number of blocks on the file system $*$ the size, in bytes, of a block on the file system of the primary shared/external storage directory.
AvailableInternalStorage	None	Number of blocks that are free on the file system and available to applications $*$ the size, in bytes, of a block on the file system.
BatteryLevel	None	Returns the current battery charge.
ProximitySensor	None	Mobile phones use IR-based proximity sensors to detect the presence of a human ear.
CpuInformation	None	Returns OS architecture, eg. armv7l.
RamInformation	None	RAM-size in bytes.
HardwareBuild	None	The name of the hardware (from the kernel command line or /proc).
HardwareSerial	None	A hardware serial number, if available. Alphanumeric only, case-insensitive. For apps targeting SDK higher than N_MR1 this field is set to UNKNOWN.
CameraInformation	None	Supported hardware level: LEGACY (2) $<$ LIMITED (0) $<$ FULL (1) $<$ LEVEL_3 (3).
Network		
LocalIpAddresses	None	Getting the list of interfaces in the local machine.
LocalHostname	None	Device name.
ConnectionType	None	Returns details about the currently active default data network. When connected, this network is the default route for outgoing connections.

Continued on next page

Table A.1 – continued from previous page

Attribute	Permission	Summary
WifiMacAddress	ACCESS_WIFI_STATE	Returns the hardware address (usually MAC) of the interface *wlan0* if it has one and if it can be accessed given the current privileges.
WifiSsid	None	Returns the service set identifier (SSID) of the current 802.11 network.
WifiBssid	None	Returns the basic service set identifier (BSSID) of the current access point.
BluetoothMacAddress	None	Returns the Bluetooth MAC-address.
ListAvailableProxies	None	List of all available proxies.
Operating System		
OsName	None	Should be Linux.
OsVersion	None	OS version.
RootStatus	None	Checks if device is rooted.
IsEmulator	None	Checks if device is an emulator.
KernelInformation	None	Output from the kernel command line $cat/proc/version$.
ApiLevel	None	SDK version of the Framework.
BuildId	None	Either a changelist number, or a label like "$M4-rc20$".
BuildDisplay	None	A build ID string meant for displaying to the user.
BuildFingerprint	None	A string that uniquely identifies this build.
BuildHost	None	
BuildTime	None	
UserAgent	None	User Agent string.
SystemStorageStructure	None	Output from the kernel command line df.
RootDirectoryStructure	None	Output from the kernel command line $ls-la$.
InputMethods	None	Returns a list with input methods.
SystemFontSize	None	Current user preference for the scaling factor for fonts, relative to the base density scaling.
ListSystemFonts	None	List of installed fonts.
Applications		
ListInstalledApplications	None	List of installed app's.
AppPackageName		
AppPackageVersion		
AppPackageHash		
AppPackageSignatures		
DebugFlag	None	Checks if debug flag is set.

Continued on next page

Table A.1 – continued from previous page

Attribute	Permission	Summary
IsUserMonkey	None	Checks if the user is a test user by automatic testing. Different versions circulate on stack overflow.
IsUserGoat	None	Checks whether the goat simulator is installed. Different versions circulate on stack overflow.
ListSpecialFilesProperties		
ListRunningProcesses		
UsageAlternative App-Markets		
ListSocialSharingServices		
Android Webview and Javascript		
ListPlugins		
ListMimeTypes		
HttpUserAgent	None	The http user agent string reported by a browser.
CanvasData	None	Hashvalue obtained by canvas fingerprinting.

Appendix B

Installation instructions backend

This section describes how to install the backend. Prerequisite is that Ubuntu 16.04 is already installed on the machine. Installation of the backend consists of the installation of XAMPP, setting up the fingerprint database and setting up the PHP engine.

B.1 Installing XAMPP

XAMPP is a free and open source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P).[1]

1. **Step 1 : Download XAMPP**
   - https://www.apachefriends.org/download
   - Go to *XAMPP for Linux* and select the latest version.
   - Save the installer file in the *Downloads* folder.
2. **Step 2 : Go to the *Downloads* folder**
   Open a terminal window in Linux and go to the *Downloads* folder, the location where the installer file is located:

   $ cd Downloads

   The installer file *xampp-linux-x64-x.x.x-x-installer.run* should be in this folder:

   $ ls
3. **Step 3 : Set the executable permission (x) for the run-file**

   $ sudo chmod +x xampp-linux-x64-x.x.x-x-installer.run

   Verify if the permission for this file has changed (x must be added):

   $ ls - al xampp-linux-x64-x.x.x-x-installer.run

[1]https://en.wikipedia.org/wiki/XAMPP

4. **Step 4 : Run the XAMPP installer**

   $ sudo ./xampp-linux-x64-x.x.x-x-installer.run

   - *Next*
   - *XAMMP will be installed to /opt/lampp (default directory)*
   - *Next*
   - *Next*
   - *Finish + Launch XAMPP*

5. **Step 5 : Launch XAMPP and start services**

   The XAMPP control panel will automatically appear after installation if the option is checked in the installation wizard. The control panel can be started manually via the following commands:

   $ cd /opt/lampp/
   $ sudo ./manager-linux-x64.run

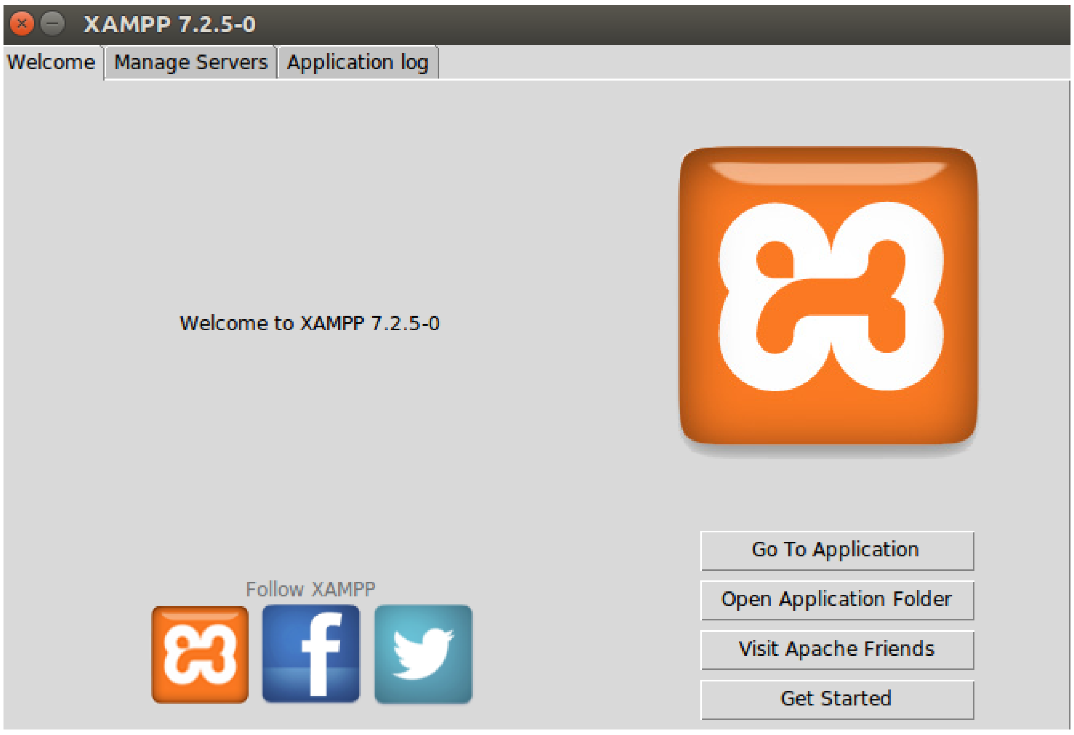
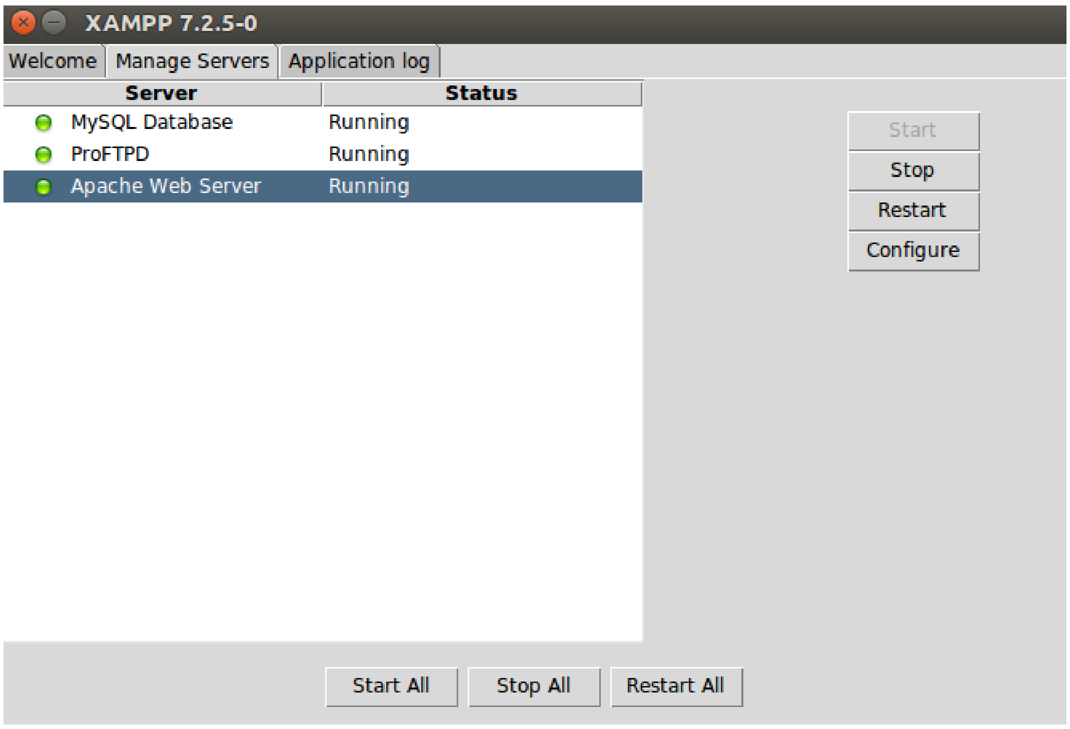


Figure B.1: XAMPP Control Panel

Select the *Manage Servers* tab. Ensure that the *MySQL Database* server, the *ProFTPD* server and the *Apache Web Server* are running.

B.2 Creating the fingerprint database

- Start phpMyAdmin via http://localhost/phpmyadmin.
- Create a new database *fingerprint_db*.



Server	Status
MySQL Database	Running
ProFTPD	Running
Apache Web Server	Running

Figure B.2: *Manage Servers* Tab

- Run the script *fingerprint_db.sql* by copying its contents in the *SQL* tab and press *Start*.
- Select the database *fingerprint_db* and go to Tab *Privileges*, then click on *Add user account*.
- This user only needs *Select* and *Insert* privileges.

B.3 Setting up the PHP engine

Figure B.3 shows the PHP directory structure. The PHP engine consists of the following files in the web root directory of Apache /opt/lampp/htdocs/:

- fingerprint/create.php
- fingerprint/getentropy.php
- fingerprint/dashboard.php
- config/database.php
- objects/dbprocs.php
- objects/entropy.php
- objects/fingerprint.php
- objects/phpgraphlib.php

Modify the script *database.php* to include the appropriate settings:

- *private $host = "public ip address"* or *"domain"*;

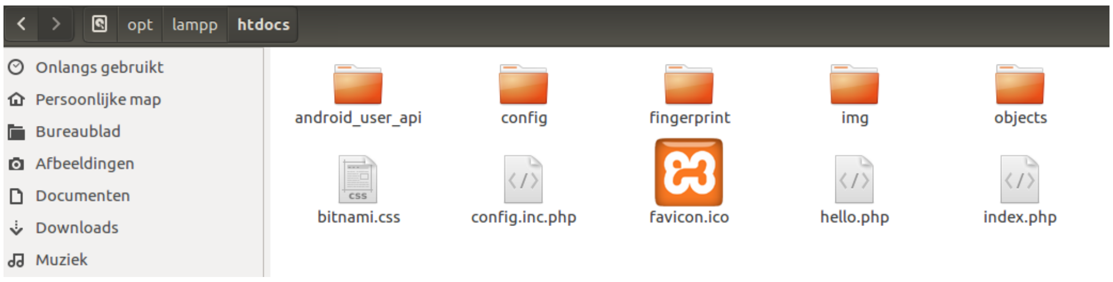


Figure B.3: Directory Structure

- *private $db_name = "fingerprint_db"*;
- *private $username = "User Name"*;
- *private $password = "Password"*;

Create the *fingerprint*, *config* and *objects* directories in */opt/lampp/htdocs* and copy the php files to the corresponding directories.

B.4 Installing Android Studio

We used Android Studio as the IDE for Android coding. This is a free download from https://developer.android.com/studio/.
The IDE is available for Windows, Mac and Linux.
Installation is very simple after downloading.

Android Studio comes with plugins for version control, like Git and Subversion. It also contains emulators to test the software on specific Android versions and devices.
When opening an existing project, Android Studio may find missing sdk-files for specific versions and will download them automatically.

Appendix C

Attention points for further development

The application was coded with attention for privacy and security, but is not ready to go live. The following points must be taken care of:

- The PHP-code in the backend should verify if the submitted JSON-string is originating from the fingerprint app.
- The PHP-code should also check the data in the JSON-string, before inserting it into the database.
- The correct url must be inserted in the config-file of the Android app.
- An SSL-certificate must be installed on the server.

Appendix D

Cover Song

Dedicated to Hugo

Your phone is always your companion nowadays
You take it from your home to any other place
But take precautions not to leave a single trace
There must be fifty ways to lose your cover

A new device is surely looking very fine
But threats are plenty when you start going on line
You should keep looking for a danger sign
There must be fifty ways to lose your cover
Fifty ways to lose your cover

Secure your phone, Joan
Your data is hot, Todd
Don't click on the ad, Brad
You're under attack
Facebook is bad, Chad
Permission's unsafe, Dave
Don't try this hack, Jack
You're under attack