UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Automated Activation of the Acoustic Scene Classification in the Corona-Warn-App

Automatische Aktivierung der Akustischen Szenenklassifizierung in der Corona-Warn-App

Bachelorarbeit

im Rahmen des Studiengangs
IT-Security
der Universität zu Lübeck

vorgelegt von
Timothy Imort

ausgegeben und betreut von
Prof. Dr. Esfandiar Mohammadi und B. Sc. Johannes Liebenow

Lübeck, den 29. März 2022

Abstract

After more than two years, we still fight the COVID-19 pandemic. To prevent big infection chains the German government ordered the development of the Corona-Warn-App. It should not only prevent but also give further information on how the virus spreads and all intending to keep an individual safe against anonymization attacks or leak sensitive private information.
In this work, we want to combine the information of the Exposure Notification API from Google/Apple and automatically activated Acoustic Scene Classification. They will provide context to the Contact. All this information is then stored in the Exposure Notification database and can then be used in multiple ways to fight the COVID-19 Pandemic.
We accomplished this goal with a combination of Motion Detection and finding differences in the Exposure Notification database. While running all this task in the background we created an energy efficient app which automatically adds context to the contacts.

Zusammenfassung

Nach mehr als zwei Jahren kämpfen wir immer noch gegen die COVID-19-Pandemie. Um lange Infektionsketten zu verhindern, hat die Bundesregierung die Entwicklung der Corona-Warn-App in Auftrag gegeben. Sie soll nicht nur vorbeugen, sondern auch weitere Informationen über die Ausbreitung des Virus geben und den Einzelnen vor Deanonymisierungsangriffen oder dem Offenlegen sensibler privater Daten schützen.
In diesem Projekt wollen wir die Informationen der Exposure Notification API und die automatisch aktivierte Akustische Szenenklassifikation kombinieren. Sie sollen den Kontakten einen Kontext geben. Des Weiteren sollen all diese Informationen dann in der Exposure Notification Database gespeichert werden. Somit kann dann auf vielfältige Weise zur Bekämpfung der COVID-19-Pandemie beigetragen werden.
Wir haben dieses Ziel mit einer Kombination aus Bewegungserkennung und dem Auffinden von Unterschieden in der Exposure Notification-Datenbank erreicht. Weil all diese Aufgaben im Hintergrund laufen, haben wir eine energieeffiziente App entwickelt, die den Kontakten automatisch Kontext hinzufügt.

Glossary

ADB Android Debug Bridge. 11, 25

API Application Programming Interface. 1, 5–9, 12–15, 21, 22, 24, 33

CWA Corona-Warn-App. 1, 5, 6, 9, 12, 13, 33

GPS Global Positioning System. 1, 3

OS Operating System. 1, 5, 8, 12–14, 25, 27

QR Quick Response. 1, 3

RPI Rolling Proximity Identifier. 1, 6, 7

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

_____

Lübeck, 29.03.2022

Contents

1 Introduction

The Exposure Notification API developed by Google and Apple is an API that is used by all Corona-apps to detect contacts to other smartphones. The Corona-Warn-App uses the API to receive a risk report. When a user is evaluated as positive, the CWA can verify the test. The API then packs up all daily keys of the last 14 days. With these keys, all used RPI-Keys could be restored. Then other users can download the positive keys of other users automatically and check whether there are contacts found in their database. If a contact is found in their database, the user is warned by the app that a high-risk contact has occurred sometime in the course of the last 14 days [15]. This system does not supply further context to the contacts. The environment in which the contact occurred is important with regard to a decision on how critical this interaction was. To solve this problem, we want to develop an automatically activated Acoustic Scene Classification. This can give context and a better risk analysis for the user and virus researchers.
Therefore, this work has the following challenges. First, there was the question which OS and Framework we needed. There was a wide range of OS-Rom and Framework combinations that we could use for our work. We could use MicroG with the official ROM or LineageOS for MicroG or we could use the official OS with the Google Play Service API and Root Privileges. All these combinations contain new challenges and road blocks that we must solve. The access of the corona-contact database itself is a challenge, as is the task to provide audio recordings for the Acoustic Scene Classification. The next thing on our list was to find related work that could help us solve our problem.
There is prior work that tries to solve problems with the unreliable range detection provided by the Low-Energy-Bluetooth module [21]. Therefore, researchers used other sensors like the Wi-Fi module [33] or the GPS Sensor [4] or even use Ultrasonic Sound [26]. Also there are apps like the Luca-App which do not rely on sensors to track persons and instead rely on QR codes which are scanned by the user[2]. But all these solutions were and are not suitable for our problem because neither of them does give any context to the contact.
The implementation we made uses the Official OS with the Google Play Service API [9] and Root Privileges. With this solution the Exposure Notification Database is accessible with the Superuser privileges but also does not disable the features provided by the Corona- Warn-App. We also show that with a background audio recorder we can supply sound samples for the Acoustic Scene Classification. And we are energy efficient because

we only consume less than 6% battery over ten hours compared to our baseline without the app running. We also compared our implementation with an interval-based approach where we scan every 60 seconds. Compared to this approach we consumed 16% less battery level over ten hours. Over the said course of ten hours the interval-based approach consumes more than two times more energy than our approach. We achieve this efficiency because our app only triggers an energy consuming action like the Acoustic Scene Classification when motion and database changes are detected.

1.1 Contribution

We create an app that can

- read the Exposure Notification Database
- add more proto-data-entries to the database

We implement an automatic Acoustic Scene Classification to save the

- label of the current environment
- probability of the label

We implement motion detection that

- saves the label of the current motion
- triggers the database scan

We give the user a possibility to remove the Data without using external tools.
We create a Recorder that provides audio samples to the Acoustic Scene Classification.

1.2 Related Work

The Corona-Warn-App tracks contact only based on the Low Energy Bluetooth Technology. This method has high error rates for the exact distances measurement and therefore it faces criticism [21]. Researchers try to improve the accuracy with machine learning. Another way could be to use other sensor data of the phone [29] , another approach uses ultrasonic sound [26]. These methods can improve the detection rate, but they cannot give any added information on the context of the contact. The environment in which the contact occurred is important with regard to a decision on how critical this interaction was. The most obvious solution for this would be to use GPS data. This approach is realized in the South Korean contact tracking app [4]. The Korean version of the Corona-Warn-App does not only store a movement history, it also uploads the age, gender, and nationality. These data will be uploaded if a person is tested positive. This approach has massive privacy concerns and cannot be realized in the EU and other countries with strict data privacy rules. The user data are uploaded to a server which means that all the data leave the phone and are stored remotely. With this solution, the user completely loses track of his data, which could be used for nefarious purposes.
So, what other approaches are realized already? Nowadays, many places have some sort of Wi-Fi network, for example, a cafe or a restaurant. These data can be used to obtain more information about the contact. Another work shows us that it is possible to track a person via Wi-Fi networks rather than via Bluetooth [33]. Then the system can supply the specific location and the duration. This solution has a major flaw because the user is not always within a Wi-Fi network's reach. So, this approach is not suitable for us either.
A German alternative for the Corona-Warn-App is the Luca-App [2]. This app needs the user to scan a QR code when he attends an event or enters a restaurant in order for the host to get an overview of the guests. If a person is tested positive, all guests who have been at the same event get notified that they had a critical encounter. This app is also flawed because it cannot supply any information on whether a critical contact really occurred, for example, when an event is split into more locations, such as an exhibition. The person who gets notified could have never met the positive person. So, this will create many false positives.

2 Preliminaries

In this section, we focus on three topics which are the most important ones for this research work. First, we take a closer look at the general structure and process of the Corona-Warn-App. Second, we take a closer look at the task of Acoustic Scene Classification. Third, we take a closer look at the Database and how we can add more Proto-Data-Fields.

2.1 The Corona-Warn-App

The coronavirus is spread by way of contact between people. This creates a need to track those people who have met a person who has been tested positive. To support the health authorities in tracing contacts, the German government issued the development of an app. This app is called the Corona-Warn-App [5]. The app can communicate with the Exposure Notification API provided by Apple and Google. The Exposure Notification API is implemented in the OS to supply more security and privacy. The API does not send any information directly to the Corona-Warn-App, but only through API calls. The mechanism to track contact used by the CWA is a decentralized contact tracing mechanism. To better understand the inner workings of the app we will describe the installation of the app, the sending and receiving of different identifiers, the input of the test results and the process of tracing contacts.

2.1.1 Installation

The app can be acquired through the Google Playstore or the Apple App Store. After the download and installation, the app shows the users a notification to accept the data protection guidelines and allows the app to access and use the Bluetooth-Sensor. In the next step, the user needs to give the CWA permission to interact with the Exposure Notification API. When all the rights are granted to the app it can supply a risk report for the user.

2.1.2 Normal Operation/Exposure Notification API

The Exposure Notification API will now track all the incoming Rolling Proximity Identifier Keys the duration and signal strength [14]. These parameters are stored in a database in the system storage *„/data/data“*. The technology used to send, and receive this information is the Low-Energy-Bluetooth Module. This happens completely in the background without any user input. The identifier is changed every ten minutes to ensure no one can be tracked or identified.

2.1.3 Positive Corona Test

After receiving a positive Corona test, the user should upload the test result into the Corona-Warn-App. This result is then verified with the test result in the Test-Lab Database. If it is a valid test result, all diagnostic keys of the positive person are then packed and uploaded to a central server of the health authorities. After 24 hours, the server then combines all positive diagnostic keys of the last 14 days in a single pack.

2.1.4 Tracing

Afterwards, the CWA can download all positive keys. Then the Exposure Notification API receives the pack of High-Risk diagnostic keys. The Notification API compares the user-seen RPI-keys with the High-Risk contact keys. With the duration and signal strength of the contacts, the API generates a risk report. The CWA can now display the result of the report to the user.

2.2 Acoustic Scene Classification

This term describes the task of assigning a label to an audio clip without any further context. For example, we record a ten second audio clip in a train station and we want an AI to return the label train station. This method consists of three steps. First, we need a recording, then the clip is pre-processed to achieve a better prediction rate. After that, this new input is analyzed by a classifier which returns a label and a confidence level.

2.3 Exposure Notification Database

As mentioned before, the Exposure Notification API generates a database with all the gathered information. This database is stored in a root-only area and so cannot be accessed by the normal user. The location is *„/data/data“*, this location is used for all kinds of data. Most apps store their information there. For example, the location *„/data/data/WhatsApp“* is the path where all the photos, videos and messages of WhatsApp are backed up and stored.
The database itself is stored in a database format called Level DB. This database format was developed by Google. The database is a key-value storage. The key and values are stored as byte arrays. As this type of database does not have an SQL-based Primary-key, they can store the contacts directly with a combination of date and RPI-key. Over the day a person can meet a vast number of other persons, so a performant database is needed which can manage many write operations. Google showed that Level DB can outperform SQLite and Kyoto Cabinet [16].
The structure of the Exposure-Notification API database is such that the key is a combination of UNIX-Timestamp and RPI-key which in turn form the Key-entry in the LevelDB. The information in the data-part is stored in Proto-Buff entries. Proto-Buff is a data format that is used to serialize structured data. Google also developed Proto-Buff, and they supplied a code generator for multiple languages. The entries stored are *„timestamp“*, *„rssi“*, *„rssi_multiple_values“*, *„aem_multiple_values“*, *„aem“*, *„previous_scan_timestamp“* [23]. For every scan of the API, a new entry is generated. With this information, the API can decide how dangerous the contact was because the entries include the duration and signal strength for each RPI-key met by the user. In our work, we want to copy the database with all the existing information and add new data fields.

2.4 Root Privileges

For this work it is necessary to acquire root rights on the phone. With root privileges we can access the Exposure Notification Database. These privileges can be granted with a root tool like Magisk [38].

2.5 MicroG

MicroG [27] is a rebuild of Google Services without the supervision of Google. It is open source which makes it quite easy to customize, and this is necessary for our work. The big problem is that we need a new OS that supports MicroG, and we also need to spoof the signatures. This step is necessary because apps that expect the Google Play Service API modules do not want to interact with other signatures. That is the reason why we need to spoof the official Google API Signatures and lead them to our MicroG modules that replace their functionality.

3 Overview & Challenges

This Chapter discusses the problem and therefore the challenges that occurred in this work.

3.1 The Problem Overview

A powerful tool to fight a pandemic is contact tracing. Contact tracing is used to inform a user that he had a critical contact sometime during the last 14 days [19]. This is extremely helpful to evaluate the risk factor and to prevent further spreading of the virus. To help this method, we must reconstruct the infection chain. This is only possible if the positive person shares his or her keys with all other users. With this method, all other users can search for the High-Risk keys in their databases. The Exposure Notification API of Google and Apple provide the contact tracing mechanism. The health authorities provide the server and the High-Risk key database. The Corona-Warn-App was developed to combine these two methods. This approach is helpful but there are still many problems that are not addressed by this solution. The contact tracing which is provided by the Exposure Notification API is based on the Low-Energy-Bluetooth sensor. This sensor is a part of every modern smartphone. In the past, this sensor was used to determine how many people were in a room or at an event. For example, a library or a canteen can show their capacity online and how many customers they are serving at any given time [3]. With this sensor, other phones can create a database of all seen phones plus further information. All these keys are stored for 14 days. As the name implies, the Low- Energy-Bluetooth sensor is not powerful and can only send and receive over a few meters. The range is not the most important key factor to decide risk. The environment is a key factor in the spread of a virus. A contact in a park is not as risky as a contact in a crowded bus with almost no fresh air or air movement.
There is already an approach made by the government to let the user share more information like age or state and country. This can then be donated as a data set to the RKI. Also, the Deutsche Telekom AG provided anonymized data to the researchers [1]. The donations can be used to calculate statistics and subsequently lead to a clearer view of the pandemic situation. With the current implementation of the CWA there is still no information regarding the environment. The problem is that the real location of the user is not relevant but only the environment. Therefore, an Acoustic Scene Classification should be

a promising approach to our problem. With the label and the confidence rating, we can then use this information to give the user a better understanding of his risk factor.
The Corona-Warn-App cannot access the database directly. The solution for this is to add a tool that can access the database with all the contacts and then add three data fields for the label, the confidence, and the activity. This leads to another problem. The activation of the recording needs to be started automatically. Also, automated activation must be energy efferent because post-processing on the phone can drain the battery much faster than regular usage. With this goal in mind, we decided to use a combination of sensors 3.1. With this, we first check for motion with the Google Activity Detection. This then triggers a job to check whether any new database entry has been made. The combination of the two sensors prevents us from recording unnecessarily. If we find a new entry, the Acoustic Scene Classification is triggered, and a new label is set for all incoming contacts after this point.
If the user has any privacy concerns, he can decide to delete this database at any given point in time. This solution can be used to enable the potential of Low-Energy- Bluetooth in combination with Acoustic Scene Classification.

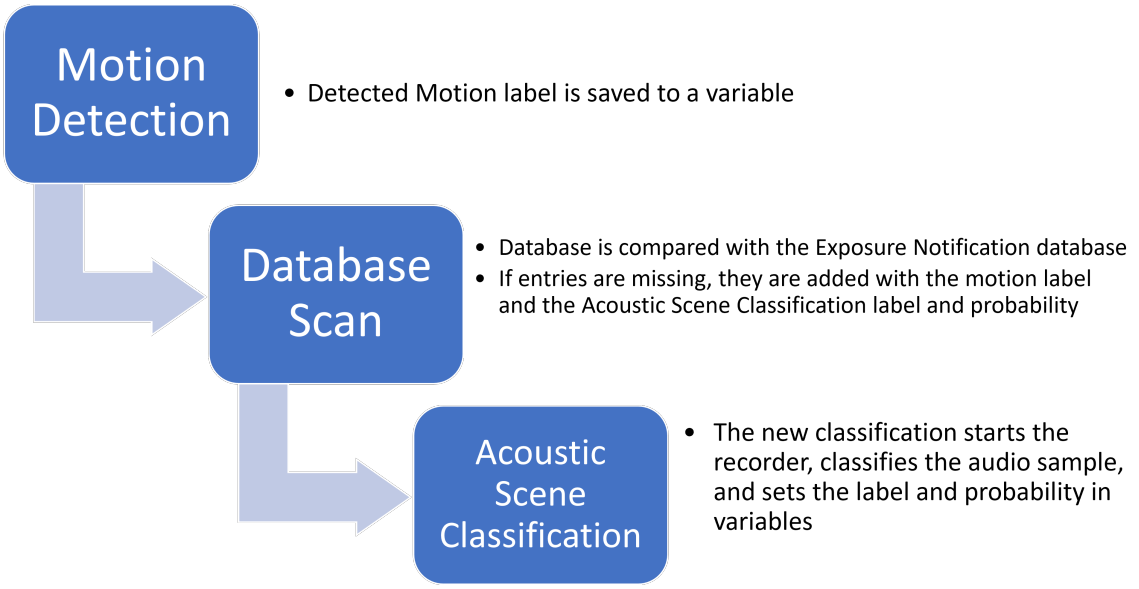

Figure 3.1: Simplified program activation states of the app

3.2 Challenges

With the general idea how we want to solve our problem we face the following challenges.

3.2.1 Software and Hardware Choice

The first challenge was to find the right setup for this work. We decided that an Android-based Smartphone with Android 10+ would be a good fit for this work. After this decision, we searched for a mid-tier smartphone. The Google Pixel 4 was best for this and can be modified very easily with the ADB-Tools provided by Google [17]. The IDE was an easier decision to make because Android Studio[8] is free and easy to use. It also uses the IDEA Community Edition [18] as its engine which we have used in the past can provide us with all necessary plugins. The plugins are Proto and Kotlin which were both necessary for this work.

3.2.2 Why We Do Not Rely on Low-Energy-Bluetooth Scanning

The Low-Energy-Bluetooth Sensor has been in use for a while now, so it is not hard to find a proper scanner for this technology. The app we used for our tests is *„nRF Connect for Mobile“* from Nordic Semiconductor ASA [31].
This app can show us the Low-Energy-Bluetooth signal we want to detect. But with this solution comes the first limitation. The Bluetooth Broadcast scan can only be activated when the screen is on. This is a permission option that cannot be removed by user permission because it is system based locked. Also, we can scan for contacts in a 10-metre-range, but we need to store the resulting information in a database. After that we would need to combine these scans with the Exposure Notification Database. This would be difficult because we need Superuser rights to access the database.

3.2.3 Why We Cannot Use MicroG with the Official ROM

The official builds of Google-Pixel-Rom are not compatible with MicroG [27] because we need to replace the Play Service API signatures. The first attempt was to root the ROM and try to remove the Play Service API packages with Superuser rights. This was possible but left the ROM in an unusable state because some parts of the API remained signatures in the OS and therefore prevented MicroG to take over the signatures.

3.2.4 Why We Cannot Use LineageOS for MicroG

This is a build of LineageOS with MicroG included[25]. This OS works great with the CWA, all other features are enabled, and the Google signature is spoofed without a problem. The biggest problem for us was that we could not replace the MicroG implementation with our own complied version without destroying the signature spoofing that was pre-installed. So, this OS still cannot supply the features we need.

3.2.5 Why We Do Not Use drFone as Our Root Tool for Android

After realizing it is necessary for us to get Root rights, we need to decide on how to get them. One of the biggest solutions for this is called drFone [35] Root. This Windows- based tool claims to supply Root rights on our OS. This claim is not realized for non-official OS-like LineageOS. Therefore, this tool is not suitable for us. Also, it is closed source and has a purchase model underneath it. Consequently, we decided to look for another solution.

3.2.6 Why We Use Magisk as Our Root Tool for Android

We then discovered an open-source and free solution which is called Magisk [38]. This Root- tool fits best because many MicroG components can be installed with their package manager on the root level [37]. First, we install the app, then we need to give Magisk the boot section of our OS. Then Magisk creates a modified version of this boot section, and this is the entry point for Magisk to claim Root privileges on the OS. After successfully booting in the new OS, we can check whether the correct rights are granted and complete the installation of Magisk.

3.2.7 Why We Cannot Use Manually Installed MicroG on LineageOS

We evaluated a LineageOS [32] without a pre-installed MicroG. Therefore, we needed to install MicroG and all the needed signature spoofing ourselves. This was a troublesome way to proceed because the spoofing of signatures is very unstable and can lead to errors and problems if that spoofing does not work properly. First, we need to get Root rights on our OS. This is necessary because installing MicroG and spoofing the signatures can only be realized on a root level.
After that, we install MicroG with this installer [30]. This install grants us the possibility to change the MicroG package [37] to our own and this install a self-compiled version with our new additions. The next problem is that we still need to spoof the signatures. The built-in solution for this is Magisk Riru [40]. Riru is used as a basis module in many custom root-level apps. First, we tried to use an App called Xposed. This app is only supported on Android versions before 10 [36]. Consequently, this would be a dead-end for this work but there is a newer tool available which is called Riru-LSPosed [39]. After installing this we could finally see our MicroG installation and activate the signature spoofing. After starting the CWA we still could not connect to the Exposure Notification API. After troubleshooting each step and reinstalling the whole OS we found out that we needed to install an app called FakeGApps [34]. This app then proved to be the missing part in our puzzle. After this install, we could verify that our modified and self-compiled MicroG version was running and could interact with Google Play Service API-based apps. The problem with this method is that when we update the MicroG app we need to install it manually on the phone. This includes repacking it in the MicroG installer and removing the earlier version and registering it in the signature spoofing app. This takes around 10 minutes, and this is acceptable when we need to do it a few times, but with all the testing and implantation ahead we need to reinstall it at least hundreds of times. Another problem is that our app would be a custom MicroG version and so could only be used by those few people who are willing to update their OS and install MicroG and configure the signature spoofing.

3.2.8 Why We Choose the Official OS with the Google Play Service API and Root Privileges

We used the official OS [9] with the Google Play Service API, but we granted our app Superuser rights and could access the work folders of the Google mobile services. This is where the database stored with all seen contacts and further information we try to enhance is located. So, we rooted the official OS in the same way as the LineageOS, but we did not install the MicroG components. This meant for us that we had to treat the Google API as a black box but can still work with the output.
Getting Root privilege with the Magisk app is far less work than in the other approach, which in turn means that we can reach more people. That we can use Google API comes in handy in our Motion Detection approach.

3.2.9 How We Work with the Exposure Notification Database

First, we need to find the database. A great work which we found early on is the corona-warn-companion [22]. This app can display all information stored in the database and shows the exact strength length and time when a critical contact occurs. This means that the app accesses the database directly because the official Exposure Notification API does not supply any methods that can return this much information about the contact that occurs over the day.
After analyzing and evaluating the implementation of the corona-warn-companion we got a good understanding of how the whole database works and how we can extend the database without losing information or destroying the whole data structure. We found out that the database is stored in the *„/data/data“* directory. This was not a problem for us because we had Superuser rights and could access this directory. There the information is stored in a Key-Value storage called LevelDB [20], where all the keys and scans are stored. The scans themselves are stored in Proto-Buff entries. This structure was hard to understand at first because before this we already looked at the approach of how MicroG [28] implements the Exposure Notification API. This approach is based on SQL which was much more natural for us to understand. Nevertheless, we can now read the database, and we can edit the Proto [12] file provided by the corona-warn-companion to add our new data fields and add them to the existing data fields from the scans. So, we achieved the goal we set to edit the data fields and read the existing Exposure Notification Database.

3.2.10 How We Detect Motion

The most basic approach would be to check the gyro-sensor data and detect when changes occur. This method of tracking motion would work but would lead to many false positives. Consequently, we needed a smarter approach to detect motion. We found out that Google supplies a motion detection API [11].

First Implementation

The First implementation of motion detection was made with a service called Google Activity Recognition API. This tool used some deprecated features which do no longer start on an Android 12 Phone and would lead to app crashes. Nevertheless, the API works well and detects motion with fewer false positives than our basic motion detection. Also, this API returns labels which we also store in the database to supply further information. So, we needed to rework this to have a future-oriented solution.

Final Implementation

Here we based the implementation on the Google Play Location Samples provided by Google for app developers. There is an implementation for the Activity Recognition Client [13]. This is the Implementation we used in our work. We extracted the general usage of the API and implemented all necessary parts in our work. We then evaluated this implementation in Android 10, 11, and 12. All three Android versions now fully support our new Motion Detection approach, and we can progress to the next problem. The next problem is that we need audio recording for the Acoustic Scene Classification.

3.2.11 How We Can Record Audio in the Background

This work needs an audio recorder that can record audio in the background which is a much harder problem than we anticipated. The problem is that Android is overly aggressive with its permissions and has set restrictions that not even the user can revoke. After evaluating the recording with the built-in recorder of Android we were able to figure out that it must be possible to record audio even when the screen is turned off and the phone is locked.

First Implementation

This approach is from the implementation of the Acoustic Scene Classification by Johannes Liebenow [24]. This is a simple audio recorder triggered by the Acoustic Scene Classification. This approach works very well and supplies a good audio clip for the classification.

When we tapped out of the app or turned the screen off while recording, the app crashed. This is normal because an audio recording is only possible when the screen is on. This security feature is implemented to prevent audio recording without the knowledge of the user. Therefore, we needed a solution to this problem. After analyzing some open-source audio recorders we understood that all implementations are using a service to continuously record audio.

Record with a Foreground Service

After understanding the necessity for a service, we needed to know what kind of service we needed [6]. Android supplies two kinds of services: first, we have the background service. This service runs even if the app is in the background but cannot record when the screen is locked. The other service is a foreground service. This foreground service can record even with a locked screen. So, we needed to implement a foreground service. To register a foreground service, we had to add the service in the android manifest and then had to add the necessary permission to start a foreground service there. Then we needed to have a class with the recorder and a recorder thread that wrote the audio buffer to our audio file. After the first working implementation, we detected another problem.

System Permissions for the AudioRecorder

We use the default Android AudioRecorder library [7]. This works well once we have initialized the audio recorder. Then we start the recorder and afterwards the thread. Then we stop the thread after ten seconds and clear all values and wait until we start the next recording. If we want to start an audio recording when the screen is turned off, we get an error message. If the screen is on and the app is in the foreground, we can initialize the AudioRecorder with the same code. This is a big problem because the permission is managed on a system level. Therefore, we cannot initialize the recorder when the screen is locked and cannot avoid this problem. After some experiments, we found out that the length of the recording is not limited, and the AudioRecorder is not stopped by a locked screen. It only cannot be initialized with a locked screen. So, our approach makes use of this loophole and continuously records audio but uses the Dashcam principle. We have two recording slots on which we record. When we recorded for a predetermined length, we saved the clip to slot one and changed it to slot two and recorded it there, and then started over. This approach works and thus disables the limitation from the system. The Acoustic Scene Classification can now take the recording slot which is not used and thus have a sample to classify.

3.2.12 Privacy

We record audio constantly. This is far from an optimal solution but it is the only possible approach that we could come up with. It also uses a mechanism that is not documented but works in a real-world environment. To improve this approach, we do not store the audio if it is not necessary. This means that we have a Boolean in the Acoustic Scene Classification called storerecording. This is set to false as a default value. When we record, we do not write the audio buffer to the file. This way we do not store any information on the disk, but it is only stored in the RAM. If we want to analyze the audio with the Acoustic Scene Classification, we set the Boolean to true and activate the function which ensures that the audio buffer is written into the corresponding audio file. This way, we record constantly but do not store any audio for longer than two * recording lengths, which is reasonable. We will further elaborate on this topic in the limitation section.
Another concern would be the extended database. But all information in the database is already on the phone and the location of the new database is also in *„/data/data“* and therefore does not change level of security.

3.2.13 Energy Consumption

This app is meant to be run alongside the Corona-Warn-App. Therefore, one challenge in this work is to implement a solution that does not consume much energy. The user should not need to change his recharge routine because of the app. In our evaluation we will take a closer look at whether we solved this challenge successfully.

4 Methods Used in the Work

We must decide when it is necessary to update our label and start the Acoustic Scene Classification. The process can be split up into the following three steps: First, the motion detection is activated by my motion like walking or driving in a car or bus. Google provided a library for app developers that can determine whether a user is walking, running or driving. The library is called Google Activity Recognition Client [10].
Second, we have the database access. We verify that a new contact with another person is made. This is necessary to ensure that no unneeded scans occur. For example, when a person is driving alone in a car, the motion detection calls the Acoustic Scene Classification all the time, but the user is alone in the car and has no contact with any other person. This would lead to an edge case where we would do many unnecessary scans. So, what is the problem of skipping the motion detection and just relying on the data? The edge case that can occur here is that a user can work at an open-plan office. There, the phone would detect a new contact every few seconds. For example, if we sit in the same place for four hours, we do not need a new Acoustic Scene Classification every few minutes. That is the reason we need to combine the two sensors.
After that, we have the Acoustic Scene Classification. The Acoustic Scene Classification itself is split into four parts. First, we have the recording which generates a sound clip which is then pre-processed and then classified. After that we can set the new label and confidence that the AI returns. In the following, we will explain the steps in more detail:

4.1 Motion Detection

First, we need motion detection. A user can only change from one location to the next with any kind of movement. The smartphone sensors can detect this movement. To prevent too many false positives, for example warnings triggered by hand movement, we decided to use the Google Activity Recognition Client [10]. This is a Google library provided for app developers. This module can detect activity. If we get the label walking, running, or driving, we then trigger our next step.

4.2 Database Access

On the first boot up we make a copy of the current database state. These entries only get an empty label because we do not have any information on the environment the contacts occur in. Then a recording is started, and a first label is set. This label is then used until we get the motion detection trigger. The function now compares the two databases. If there are no differences in the two databases, the function does not start the Acoustic Scene Classification. If there are differences, the old label is then added to all new entries. And the Acoustic Scene Classification is started.

4.3 Acoustic Scene Classification

4.3.1 Recording

The phone records onto the disk for 20 seconds. After a waiting time, a 10-second audio clip is converted from .pcm to .wav. This file is then passed to the pre-processing and is removed from the disk to ensure privacy.

4.3.2 Pre-processing

This step is needed because machine learning provides much better results when the data is pre-processed in some way rather than given the raw input. After the pre-processing of the audio file, the input gets deleted.

4.3.3 Classification

The classifier then uses a neural network to classify our input. After analyzing, the AI returns a one-dimensional array with the label and the prediction confidence. The model supports ten labels which are returned in the array.

4.3.4 Postprocessing

The top label is then set as the new label and the confidence is saved and can then be used by the database editor to set the label for all incoming contacts until the next Acoustic Scene Classification sets a new label.

5 Implementation

In this chapter, we will describe our setup and implementation in more detail. First, we will take a look at the setup and the used libraries and devices.

5.1 Setup

We use Android Studio [8] Bumblebee 2021.1.1 Patch 2 on Windows 10 as our IDE. The smart- phone is a Google Pixel 4 with Android 11 installed. The phone is rooted with Magisk [38] v23. This solves the challenge of the hardware choice.

5.2 Motion Detection

The motion detection library provided by Google is called Activity Recognition Client [10]. To use this API, we must cast a Google API client. This is implemented in the *„MainActivity“* class and called *„mActivityRecognitionClient“*. After initializing the client, we call the method *„MotionDetectionstart“*. Here we start the task to request activity updates and set parameters like the detection interval that is set to 30 seconds in our setup. With this task, we also get a pending intent dent for every detected activity. These detections are then handled in the *„DetectedActivitiesIntentService“* class. In this class, we use the method *„onHandleIntent“* for every detection report we receive. First, we reset the values of our last detected activities. After this, we get the list with the current state of the device and assign a confidence rating to reach activity. Then we iterate through all activities and try to find a high confidence activity with a confidence score of at least 75. If we find one, we set the variable *„motionfound“* to that type. We initialized this variable to 4 which is the value for unknown activity. If we have found an activity with a confidence score above 75 and the type is 3 (=Still) or 5 (=Tilt) we set the Boolean *„foundStillorTilt“* true. At the end of the method, we check the time which has passed between the last motion detection and now. We set the threshold to 45 seconds to give the Acoustic Scene Classification enough time to complete the last classification. If more than 45 seconds have passed between the activities the next case is decided by the *„foundStillorTilt“* Boolean. If we have found Still or Tilt as movement, we do not want to check the database because we are still in the same environment. If we find motion, we call the method *„writeActivityData“* and trigger the database scan. This is how we solved the motion detection challenge.

5.3 Read Database

The tools for editing the database are implemented in the *„ActivityContactDB“* class. The method *„writeActivityData“* is called when we want to update the database. In this method, we call *„fetchMissingData“* to detect changes from our enhanced database. First, we open the database. Then we map the database entries with our *„readToMap“* method. To work on the map elements is faster and easier than editing every entry directly in the database. Next, we iterate through every entry in our new copied database. If we find a key in the copied database which is missing in our enhanced database, we copy the key and the data and set the activity, the label, and the probability. To set the three extra data fields we use the Proto-Builder which is in the class *„ContactRecordsProtos“*. With *„.set*Data field*“* we can set the value in our predetermined data fields. After we build the whole missing key entry, we save the change with *„getDB.put.()“*. If all elements of a key are the same, we just skip the key. The last case that could occur is that the key exists but one of the Proto entries is different. This could possibly happen when we stay in the range of another person and the API continuously scans the range between the two devices. If we find a new entry, we add the detected activity, the label, and the probability of our Acoustic Scene Classification. The enhanced database is then saved in the program cache. This location can be cleared in the app settings of Android.

5.4 Write Database

The tools for editing the database are implemented in the *„ActivityContactDB“* class. The method *„writeActivityData“* is called when we want to update the database. In this method, we call *„fetchMissingData“* to detect changes from our enhanced database. First, we open the database as described above. Then we map the database in our *„readToMap“* method. To work on the map elements is faster and easier than editing every entry directly in the database. Next, we iterate through every entry in our new copied database. If we find a key in the copied database which is missing in our enhanced database, we copy the key and the data and set the activity, the label, and the probability. To set the three extra data fields we use the Proto-Builder which is in the class *„ContactRecordsProtos“*. With *„.set*Data field*“* we can set the value in our predetermined data fields. After we build the whole missing key entry, we save the change with *„getDB.put.()“*. If all elements of a key are the same, we just skip the key. The last case that could occur is that the key exists but one of the Proto entries is different. This could happen when we stay in the range of another person and the API continuously scans the range between the two devices. If we find a new entry, we add the activity detected, the label, and the probability of our Acoustic Scene Classification. The enhanced database is then saved in the program cache.

This location is user-accessible and can be removed in the app settings of Android.

5.5 Audio Recorder

The audio recorder is started in the *„MainActivity“* class with the call *„StartService.startRec()“*. In this method, we start the recording service as a foreground service. This is necessary because it allows us to record even when the phone is locked. By creating the service, the class *„RecordingService“* is created as an object. After registration, the pending intent, and the notification builder we call the method *„startRecordinginService“*. In this method, we initialize the audio recorder and check whether the state of the audio recorder is correct. This is necessary because when the screen is turned off or the app is not in the foreground, the audio recorder cannot be initialized. After that, we start the recorder and set the Boolean *„endRecord“* false and create a recording thread. This thread uses the class *„RecordingRunnable“* and starts with the method run. First, we create some file variables. These are necessary because the audio recorder uses a Dashcam Principle. This means that we have two slots in which to record. And we constantly write over a slot and then change the slot and overwrite there. We need to do this because we cannot create a new recorder when the screen is turned off. Therefore, we must record constantly and change the output file to avoid the restriction. So, we check whether the files are created and if they are not, we create the files. Then we enter the first while loop which is controlled by the Boolean *„endRecord“*. Then we decide which file slot we take and then start the recording. This while loop is controlled by the Boolean *„recordingInProgress“*. The length of the recording is determined by the value of *„recoringlength“*. In our case, this value is set to ten seconds. After reaching this length the recorder start time is reset, the recording is finished, and the buffer is cleared. Then we change the recording slot and set the Boolean *„recordingInProgress“* true again and start the next recording on the other recording slot. One crucial factor is to delete the audio files when the app is closed by the user. This is implemented in the *„MainActivity“* class in the *„onDestroy“* method. Here we end the recording by setting the Boolean *„endRecord“* true and deleting all audio files. This concludes how we solved the audio recording and privacy challenge.

5.6 Main Activity

In the *„MainActivity“*, we define the start of the app. When the app is started the extended database is compared with the Exposure Notification API database and missing entries are added without a label or activity. After this, the Acoustic Scene Classification is triggered, and a first label is set. After this, motion detection is used to detect whether a new scan is needed.
We also provide user feedback with the help of GIFs. Therefore we first start with this picture 5.1 to show that the components are loading. We show the user with a pulsing Ring that the app is fully functional 5.2.

Figure 5.1: Screen while loading

Figure 5.2: Screen after loading complete

6 Evaluation

In this chapter we present our experimental setup with the software and hardware combination we used. We show the energy consumption and running times of the individual components.

6.1 Experimental Setup

We use a Google Pixel 4 with Android 11 as its operating system to simulate a real-world application of our app. We also use an Android 10 emulator provided by Android Studio. Android Studio and the corresponding ADB-driver allow us to install the app directly on the Pixel 4. We can also see the log and errors in Android Studio. This provides us with monitoring opportunities to detect whether our app works as intended. The OS is an official build provided by Google. We get root privilege through Magisk and grant our app Superuser-rights in advance.

6.2 Efficiency

In this section we will discuss the energy consumption in a time interval over ten hours and the run times of our work components.

6.2.1 Energy Consumption

We used the Android studio build in Profiler to figure out the energy consumption. We marked two time periods in the graphic. Between the green lines is the database scan and edit. This activity has a maximum CPU usage of 12%. The other activity in between the red lines is the Acoustic Scene Classification. There, the maximum CPU usage was 17%. This capture is made while the app runs with the screen turned off. Now we want to compare it with the screen turned on 6.2.
As we can see, the energy consumption changes rapidly. The lines mark the same events as before. The database scan now has a max of 18% CPU usage and the Acoustic Scene Classification a max of 25%. Also, the energy consumption is not light anymore but reaches medium on more than one occasion. This shows us how much energy we can save with our background implementation 6.3.

We also wanted to evaluate our app in a real-life scenario. To measure the energy consumption, we first evaluated only the audio recorder without motion detection and database test and Acoustic scene classification. Then we evaluated a forced Acoustic Scene Classification every 60 seconds. And the last case is our solution, but we let the phone sit still and move it every few hours to simulate a work or office situation.
As we can see in Figure 6.1 our solution is almost identical with the test baseline of the audio only. This means that the motion detection does not consume significant more energy than when it is not using it. We also see the efficacy difference against the time-based approach where we scan every minute.
To estimate how good our implementation would work in a real world scenario we must compare it to the baseline of the phone battery drain without the app running. We can see that the app only consumes about 5% more energy with our app running. This shows that our app can be run in the background without the necessity for extra re-charges and therefore accomplished our challenge goal for the energy consumption.

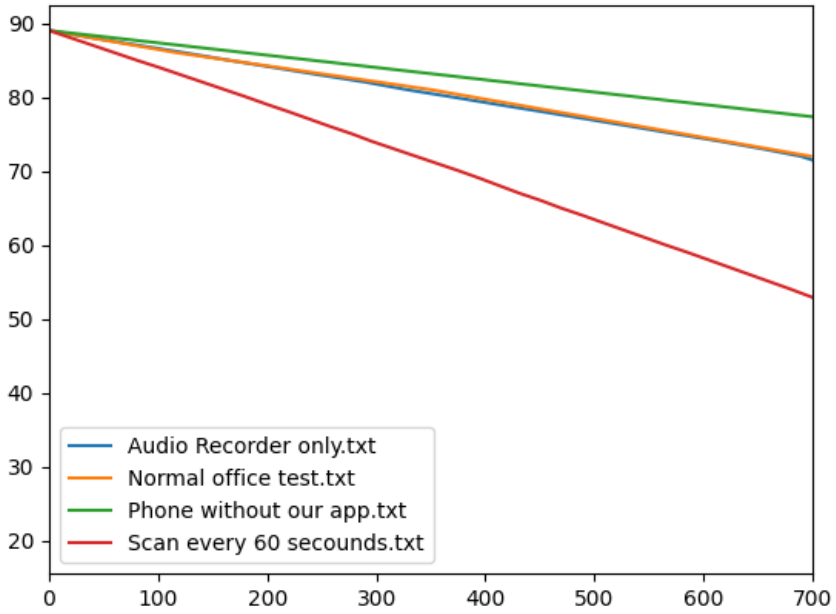

Figure 6.1: Log of the phones battery level over x Minutes.

6.2.2 Running Times

We have two components that have a dynamic runtime. These are the Acoustic Scene Classification and the database access. The audio recorder records constantly and thus has no dynamic run time. Also, the motion detection is provided by the OS and therefore not measurable for us. We start with the Acoustic Scene Classification. Here, the entire process takes place in 4.1 seconds. The database access is finished in 1.2 seconds. The runtime does not change either with the screen on or off. This is shown in the two profiler recordings we made 6.2 6.3.
These results mean that we can run our implementation every 5 seconds if necessary. But with the recordings that were needed, another twenty seconds were added. Furthermore, a case might occur that the database edit will need more time with lager databases. Therefore, we decided to set the cooldown time to 45 seconds. This way, the case that two scans may intertwine cannot occur.

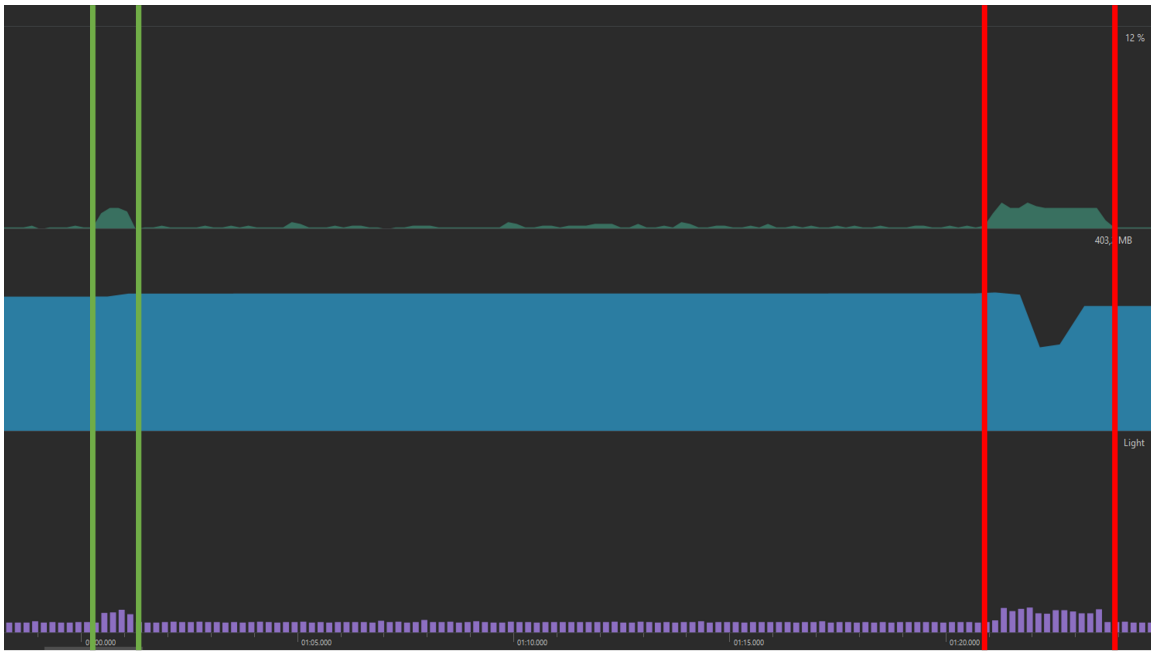

Figure 6.2: Android Studio Profiler with screen off. Between the green lines is the database read/write and between the red lines the Acoustic Scene Classification.

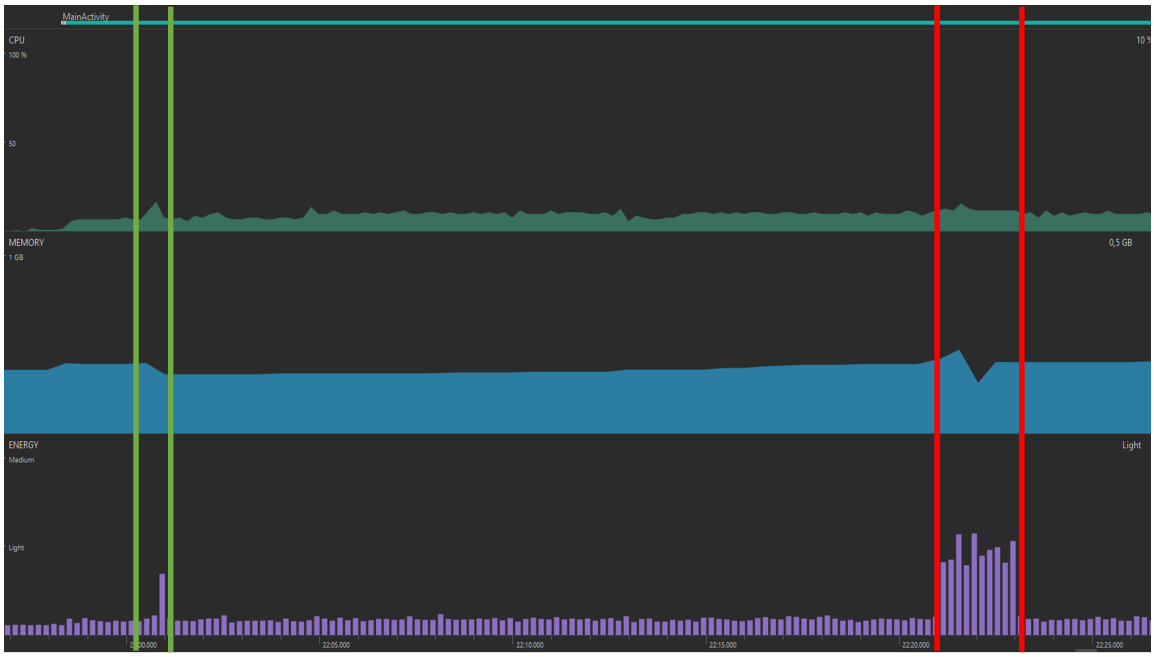

Figure 6.3: Android Studio Profiler with screen on. Between the green lines is the database read/write and between the red lines the Acoustic Scene Classification.

7 Limitations

This section discusses the limitation of our implementation.

7.1 Motion Detection

The detection is limited to the sensors of the smartphone. If the user has a habit of wiggling with his foot while sitting, this could activate the motion detection with no real change of location. We counter this with the Database-Contact Detection, but it could still lead to some edge cases where it will trigger a new classification without a location change.

7.2 The Car/Train Scenario

In a scenario where two or more people are together within a close radius, the key for every device changes every ten minutes. This creates changes in the database that are detected. With the motion detection detecting car or train motion the Acoustic Scene Classification will be at least triggered every ten minutes. This is a compromise that is taken here to avoid wrong labels.

7.3 Audio Recorder

As mentioned before our implementation of an audio recorder as a foreground service is intended and can be realized with the permissions of Android. If we want to start a new audio recording when the screen is locked, this is prohibited by the Android system. It is not possible to initialize a new audio recorder if the app is not in the foreground.

8 Conclusion

The Corona-Warn-App can be a powerful tool but needs more features to understand the context of contacts. With our solution we showed that it is possible to add this feature.
We used the Exposure Notification Database as our base and showed that the functionality of the Corona-Warn-App is fully functional when run side-by-side with our app. This means that it can be used without losing any protection offered by the official implementations.
Also, we grant the user the permission to remove the extended database at will. The database is stored in the cache folder and can be cleared by the user in the Android settings.
Our app is modular and can be used as a basis implementation for further projects that want to access the Exposure Notification Database and add their data to it.
The automatic activation that we supply is energy efficient. We showed in our evaluation that our app energy consumption is noticeable but is way better than any time-based solution. Also, in a time span of over 11 hours we showed that it is possible to use the implementation in a real live scenario without the necessity for additional recharges.
Our audio recorder implementation shows that it is possible to run the task in the background, which is why it can also be used for further implementations to access other sensors for further information gathering.
Overall we reached our set goal and created an energy efficient app that can edit the database and provides an Acoustic Scene Classification label.

9 Future Work

There are many ways how this work can be expanded and used as a base implementation. Some ideas and active research topics are listed here.

9.1 Advance Activity Recognition

The general goal of this work is to improve the CWA using human activity recognition. This could be used to improve Johannes Acoustic Scene Classifier, e.g. by better recognizing scene transitions and using those to control the classifier, or by concretely linking the prediction of the scene to the activity. An example for the latter case would be a combination of the activity "Walking" and the scene "In the car" which would be a contradiction.

9.2 Privacy

This work has the goal of learning representations of environments and corresponding labels in a secure and privacy preserving way based on acoustic data distributed among several clients.

9.3 Backend Communication

At this stage of implementation, we show that we can add information to the database without any loss of the original Exposure Notification data structure. This will make it quite easy to extend the Exposure Notification API database.

9.4 Audio Recorder

There could be a way to avoid the restrictions made by Android without recording constantly. So this module could be replaced with a better solution in the future.

9.5 Database Maintenance

The Exposure Notification removes every contact that is older than 14 days. The current implementation does not have anything like this now. In a further work, this feature can be added to clean up the database and subsequently create a smaller database.

References

[1] Stephan Broszio. https://www.telekom.com/de/konzern/details/corona-vorhersage-telekom-unterstuetzt-rki-596772. 2020. Last Accessed in March 2022.

[2] culture4life GmbH. https://www.luca-app.de/. 2022. Last Accessed in March 2022.

[3] Patrick Dickinson, Gregorz Cielniak, Olivier Szymanezyk, and Mike Mannion. Indoor positioning of shoppers using a network of bluetooth low energy beacons. pages 1–8, 2016.

[4] John P DiMoia. Contact tracing and covid-19: The south korean context for public health enforcement. *East Asian Science, Technology and Society: An International Journal*, 14(4):657–665, 2020.

[5] Bundeszentrale für gesundheitliche Aufklärung (BZgA). https://www.infektionsschutz.de/leichte-sprache/informationen-zum-corona-virus/wie-schutze-ich-mich-und-andere-menschen/die-corona-warn-app/. Last Accessed in March 2022.

[6] Google. https://developer.android.com/guide/components/services. Last Accessed in March 2022.

[7] Google. https://developer.android.com/reference/android/media/audiorecord. Last Accessed in March 2022.

[8] Google. https://developer.android.com/studio. Last Accessed in March 2022.

[9] Google. https://developers.google.com/android/images. Last Accessed in March 2022.

[10] Google. https://developers.google.com/android/reference/com/google/android/gms/location/activityrecognitionclient. Last Accessed in March 2022.

[11] Google. https://developers.google.com/location-context/activity-recognition. Last Accessed in March 2022.

[12] Google. https://developers.google.com/protocol-buffers. Last Accessed in March 2022.

[13] Google. https://github.com/android/location-samples/tree/main/activityrecognition. Last Accessed in March 2022.

[14] Google. https://github.com/google/exposure-notifications-android. Last Accessed in March 2022.

[15] Google. https://www.blog.google/documents/66/overview_of_covid-19_contact_tracing_using_ble_1.pdf/. Last Accessed in March 2022.

[16] Google. https://web.archive.org/web/20110820001028/http://leveldb.google-code.com/svn/trunk/doc/benchmark.html. 2011. Last Accessed in March 2022.

[17] Google. https://developer.android.com/studio/command-line/adb. 2015. Last Accessed in March 2022.

[18] Google. https://web.archive.org/web/20150402103057/http://tools.android.com/recent/androidstudio12preview1. 2015. Last Accessed in March 2022.

[19] Google. https://static.googleusercontent.com/media/www.google.com/de//-covid19/exposurenotifications/pdfs/exposure-notification-faq-v1.2.pdf. 2020. Last Accessed in March 2022.

[20] Google. https://github.com/google/leveldb. 2021. Last Accessed in March 2022.

[21] Gary F Hatke, Monica Montanari, Swaroop Appadwedula, Michael Wentz, John Meklenburg, Louise Ivers, Jennifer Watson, and Paul Fiore. Using bluetooth low energy (ble) signal strength estimation to facilitate contact tracing for covid-19. *arXiv preprint arXiv:2006.15711*, 2020.

[22] Michael Huebler. https://github.com/mh-/corona-warn-companion-android. 2022. Last Accessed in March 2022.

[23] Michael Huebler. https://github.com/mh-/corona-warn-companion-android/blob/master/corona-warn-companion/src/main/java/org/tosl/coronawarncompanion/gmsreadout/contactrecords.proto. 2022. Last Accessed in March 2022.

[24] Johannes Liebenow. https://mohammadi.eu/dateien/contact_contexts.pdf. Last Accessed in March 2022.

[25] lineage. https://lineage.microg.org/. 2022. Last Accessed in March 2022.

[26] Yuxiang Luo, Cheng Zhang, Yunqi Zhang, Chaoshun Zuo, Dong Xuan, Zhiqiang Lin, Adam C Champion, and Ness Shroff. Acoustic-turf: acoustic-based privacy-preserving covid-19 contact tracing. *arXiv preprint arXiv:2006.13362*, 2020.

[27] microG Team. https://microg.org/. 2017. Last Accessed in March 2022.

[28] microG Team. https://github.com/microg/gmscore. 2021. Last Accessed in March 2022.

[29] Khuong An Nguyen, Zhiyuan Luo, and Chris Watkins. Epidemic contact tracing with smartphone sensors. *Journal of Location Based Services*, 14(2):92–128, 2020.

[30] nift4. https://github.com/nift4/microg_installer_revived. 2022. Last Accessed in March 2022.

[31] nordicsemi. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp. 2022. Last Accessed in March 2022.

[32] The LineageOS Project. https://lineageos.org/. 2021. Last Accessed in March 2022.

[33] Amee Trivedi, Camellia Zakaria, Rajesh Balan, Ann Becker, George Corey, and Prashant Shenoy. Wifitrace: Network-based contact tracing for infectious diseases using passive wifi sensing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(1):1–26, 2021.

[34] whew. https://github.com/whew-inc/fakegapps. 2022. Last Accessed in March 2022.

[35] wondershare. https://drfone.wondershare.de/. Last Accessed in March 2022.

[36] Jaida Wu. https://github.com/magisk-modules-repo/riru_edxposed. Last Accessed in March 2022.

[37] John Wu. https://github.com/magisk-modules-repo. 2022. Last Accessed in March 2022.

[38] John Wu. https://github.com/topjohnwu/magisk. 2022. Last Accessed in March 2022.

[39] yujincheng. https://github.com/lsposed/lsposed. 2022. Last Accessed in March 2022.

[40] yujincheng. https://github.com/rikkaapps/riru/releases. 2022. Last Accessed in March 2022.