# Privacy Protection: COVID-19 Contact-Tracing Applications

Elena Marzi

The 2nd June 2020

## Table of Contents

## 1. Background

The recent COVID-19 pandemic has put the entire world heavily to the test. With no vaccine available and in absence of an effective medical treatment, social distancing and contact tracing remain the most effective ways to prevent the spreading of a virus into society. Contact tracing is the act of identifying people who are at risk of contracting a contagious disease after being in contact with a known carrier and, traditionally, involves interviews with each carrier about their whereabouts and meetings. This is not just time-consuming and resource heavy, but also not fast enough to isolate new carriers of the virus.

In the battle against the virus, many countries have been adopting mobile applications for keeping track of infected individuals and their contacts. Even if those applications can help to contain the pandemic, they are not free from controversy. Many concerns are raised about privacy and how sensitive data could be abused. Mobile phones are part of people's daily life and thus they contain numerous personal information like preferences, contacts, visited locations and other sensitive data that can be learned by malicious actors and used against the users' will.

Because of the high risk of data misuse, groups of researchers, government bodies, as well as tech giants have joint efforts to develop privacy-friendly applications where the main focus is to serve as an aid in contact tracing. Among those projects are the MIT open source application "Private Kit: Safe Paths" and the European protocols: DP-3T, ROBERT and NTK.

## 2. The goal of this project

This project will explore how sensitive information is protected, or abused, in a contact tracing application. In particular, the goal of this project is to build a prototype of an application, based on the Decentralized Privacy-Preserving Proximity Tracing (DP-3T) protocol, which serves as proof of concept, able to perform contact tracing in a simulated real-world situation. Using the prototype, I will demonstrate how users, who have previously been in contact with a carrier of the SARS-CoV-2 virus, are notified by the application so that they can isolate themselves or visit the doctor. I will also show how sensitive information, like location and identity, regarding the actors involved are not collected. I will then discuss deficiencies of this protocol and how it can be misused by malicious attackers; a deanonymizing attack will be simulated using the prototype. Finally, I will discuss possible solutions based on other protocols or research in the

field. Aspects such as user experience design, legislation or medical aspects will not be taken into consideration.

## 3. Privacy Location protocols: an overview

Privacy means limiting the disclosure of sensitive information about individuals or groups. In the case of a pandemic, different interests are at stake and the line between privacy protection and the use of this information in the fight against the virus, is undoubtedly blurry. If private information cannot be disclosed in any way, there is a risk that the application will be useless to trace contacts effectively. The contrary can, however, lead to catastrophic scenarios of mass surveillance, persecutions, shaming, blackmailing and harassing both individuals and groups.

Even before the COVID-19 pandemic, research has focused on the possibility to anonymize personal data in mobile applications that use location-based services (LBS)[1].
**Privacy-Preserving Location Proximity** protocols **(PPLP)**, for example, aim to reveal if individuals are in a certain distance from each other, without revealing their exact location or exact distances, and without giving the possibility to third parties to learn that information [1]. The Inner Circle protocol is a state-of-the-art PPLP protocol which uses homomorphic encryption, for computations on encrypted data without the need of a trusted server [2].

To achieve **K-anonymity**, which aims to hide the position of the user from the LBS provider by making their location non-identifiable from at least k-1 other users, different techniques can be used [3]. In the **Dummy Locations** approach, the user sends requests for k-1 fake locations (dummy) to the LBS provider in addition to the real one, linked to the same user identifier. [3] An anchor which is a location which lies in a close distance from the actual location, can also be sent to the LBS provider which will give back a set of answers, filtered by the client side [4]. In the **Spatial Cloaking** technique, the location of a user is replaced by a region, called cloaking region, which is broader than the exact location but contains it. Different approaches are possible like using a grid or partitioning the map in predefined cells [4].

[1] LBS referrers to apps that use the location of the user to provide a service. The LBS provider provides the service in exchange of the location of the user.

A **trusted server** can also be used to grant anonymity. The server, placed between the LBS provider and the user, receives the request and anonymizes it by removing identifiers and applying data transformation before sending it to the LBS provider. The server can also group the request of multiple users in equivalence classes with collective common properties [4]. If a trusted server cannot be used, it is possible to grant anonymity by using a **Private Information Retrieval protocol** (PIR) to secure communication with the LBS provider through cryptography [4].

The use of these anonymization techniques makes possible to preserve the location privacy of the user without substantially compromising the utility of the service provided. However, given the specificity of a contact tracing application, and the consequences that a misuse of such an app can have on the society, other solutions have been explored in recent months. Some of them have already been implemented, others are in the developing phase and some are just proposals.

## 4. Requirements and functionality of a tracing app

A contact tracing application should respect at least the following requirements[2] to ensure the data protection of individuals and to avoid abuse.

Anonymity for all actors involved should be pursued. Users, authorities or private companies should not be able to learn about the position or other personal information of another user or of an infected person. Data shouldn't be extracted directly from other applications or social media without the consent of the user. Sensitive data (like locations, identity, preferences) of users shouldn't be collected in a governmental database or by big corporates without protective measures. This in order to avoid abuse of power from the authority, future mass surveillance, selling and other misuse of data.

Only cases officially reported to the authority should be relevant: the application shouldn’t permit individual assessment by users if they’re contagious or not. Moreover, the health authority should be involved in the assessment of the contagious window, which specifies from

[2] Requirements are taken from an analysis of the examined sources and various readings.

which day the carrier has been contagious. No personal assumption, without medical background, should be allowed.

The application should be useful and grant a good precision of data, not giving rise to a lot of false negatives or a lot of false positives. For example, the recommended safety social distance is around 2 meters. If a user is warned anyway even if the contact has been more than 2 meters away, this can generate false positives. False negatives can be created, for example, if the application does not warn a user which has become at risk, giving a false sense of immunity.

Finally, the application should be subjected to approval from the local or central authority, to grant its authenticity and should not be compulsory but completely free and transparent [5] [6].

## 5. Existing contact tracing applications and proposals

Applications being able to promote social distancing are henceforth made available, or will be soon, in most of the countries around the world [7]. Even if they all aim to limit the spreading of the virus, the way they are designed varies a lot.

Among the first COVID-19 apps, there is the South Korean one. It aims, mainly, at keeping track of the location of a diagnosed person, in real time, in order to control if the quarantine order is respected [8]. The Chinese version of the app tracks the GPS location of the citizen, giving colors codes to use for access to certain places or to travel. The central authority evaluates the risk of virus exposure in exchange of personal information and location of each user [9]. Other designs focus instead on tracing past contacts of an infected person during a period of time, without the need to collect personal data.

One of the first examples of such a design is the "Private Kit: Safe Paths" and has been proposed by the Massachusetts Institute of Technology (MIT). The **MIT Private Kit** has been designed with security protection in mind. It is an open-source project which uses both Bluetooth and GPS technology. Private Kit compares timestamped location trails of diagnosed carriers with the user's own paths logged in their device. The application is currently in a developing phase and aims to implement an encrypted trail match algorithm[3]: data with the encrypted location of

[3] At the moment of this report the application is being built without any encryption in place.

the diagnosed carriers will be downloaded on the device and never decrypted. The project also aims to save the location trails of all diagnosed carriers in an encrypted and inaccessible database [6].

Another initiative, originally created by a wide range of experts on different fields, researchers and engineers from all over Europe led by the Swiss Federal Institute of Technology Lausanne, is the Decentralized Privacy-Preserving Proximity Tracing or **DP-3T**[4] [10]. The protocol, proposed in three slightly different versions, is designed to use Bluetooth Low Energy technology (BLE)[5], cryptography and it is highly decentralized.

Each device generates a secret key (`SK`) every day using a cryptographic hash function, such that the previous key is hashed: $SK_t = H(SK_{t-1})$
The `SKs` are stored in the local memory of the device for a period of time, for example, 14 days. From the `SKs`, a list with 128-bits long ephemeral ids (`EphIDs`) is generated using a cryptographic algorithm:

$$EphID_1 \,||....||\, EphID_n = PRG(PRF(SK_t; \text{"broadcast key"}))$$

`PRG` is a stream cipher and `PRF` is a pseudo random function such as HMAC-SHA256. The `EphIDs` are frequently changed and broadcasted in random order via BLE to devices nearby. A device also listens to other devices' `EphIDs` which are in proximity and collect the heard `EphIDs` in their local memory together with the proximity, duration of encounter and the current day.
If a person results positive to the COVID-19 test, the health authority informs the patient and starts the contact tracing process. By using an authorization code provided by the health authority, the carrier then uploads the `SK` for the first day of being contagious together with a timestamp of that day. For this purpose, a contagious window is determined by the health authority[6]. Then after the uploading, the device generates a new random `SK`.
All devices regularly download, from the backend, new pairs (`SK`, `t`) and use that information to recreate the `EphIDs` for each day and compare these with its own list of received `EphIDs`. A risk-scoring algorithm is used to compute the risk of infection and if this is high, a message

[4] This work was born under the 'Pan-European Privacy-Preserving Proximity Tracing' (PEPP-PT) project [14] but now is an independent project, entirely open source and privacy focused.
[5] The BLE is a short-range communication technology which allows two devices to detect their close proximity with a very low power consumption This means that it can be run as a background process.
[6] A patient is contagious 1-3 days before the symptoms.

is displayed on the device of the person at risk. This is a distributed system, where the backend is not responsible for any computation but serves merely as a storage platform [10].

The ROBust and privacy-presERving proximity Tracing protocol (**ROBERT**)[7], also utilizes Bluetooth LE but, unlike DP-3T, it is based on a centralized server to which devices need to register [11]. The server creates a permanent pseudonym identifier (`ID`) for each device, a symmetric shared Key (`K`) and a timestamp, which are stored in a database. For each device, the server then generates a list containing Bluetooth identifiers (`EBID`) from the `ID` and `K` and the encrypted country code (`ECC`), using a block-cipher, such as, for example, TripleDES or Blowfish. Each device broadcasts *Hello* messages which consist of a concatenation of `ECC`s, `EBID`s and a timestamp, together with a message authentication code (MAC) computed by hashing the *Hello* message with the symmetric shared key `K` [5]. Devices nearby collect the *Hello* messages, and the time of the reception into the application's *LocalProximityList*.

When a patient has been diagnosed with the COVID-19, with their consent, a part of the *LocalProximityList* corresponding to the contagious window, is uploaded into the authority server. The server is in charge of decrypting the `EBID`s in the list, reconstructing the permanent `ID` and comparing with other permanent `ID`s and flagging the match[8]. The server, then, computes the risk of contagion and sends a warning. This solution is based on a strongly trusted backend which has the responsibility of assessing the risk of contagion and the users are kept anonymous [11].

Other similar solutions that deserve to be mentioned, are the NTK protocol[9] also based on a centralized trusted server, BLE and cryptography [5] and the **Pronto-C2** protocol designed at University of Salerno, based on a decentralized system, where the responsibility in communicating the risk of infection lays on the infected party, and not on the central server (ROBERT/NTK) or the client itself (DP-3T) [12].

[7] Coming from the PEPP-PT organization and the French research institute Inra.
[8] For more detail, refer to the source 11 and the github repository.
[9] NTK is developed by PEPP-PT and the German Robert-Koch Institute.

# 6. The prototype

### Overview of the work

Because the DP-3T is one of the most discussed proposals at the moment, it aims to respect all the requirements previously mentioned, and it has a straight-forward design, the prototype is based on a slightly simplified version of this protocol.

The following assumptions are made for the sake of the simulation:

- All users, examined by the health authority, result positive to the virus test.
- Test results are instant.
- Every carrier is always willing to upload their `SK`, which represents the first day of being contagious, together with a timestamp to the health authority database.
- The strength of each device's Bluetooth signal is always the same.
- No risk evaluation is done. Users who have been in contact with a carrier get a warning even if the encounter has been brief[10].

The application is developed in Python and includes a mechanism to generate secret keys (`SK`) and `EphIDs`, from that key, which are broadcasted over a simulated Bluetooth LE channel. It also collects `EphIDs` from other devices nearby. In addition, it downloads, at regular intervals, the list with the `SK`s of those who are declared infected by a simulated health authority, recreates `EphIDs` from those `SK`s and compares them with the collected `EphIDs`. It checks if at least one of those regenerated `EphIDs` is present in the list of collected `EphIDs` and if it is, a warning message is displayed in a terminal. The application is also responsible for uploading its own `SK`, from the first day of being contagious, together with a timestamp from that day, to the health authority database, which in the prototype is just a simple text file.

Because the goal of the project was to implement the DP-3T protocol and investigate how devices interact with each other without leaking private information, a framework has been built to simulate a real-world scenario where mobile devices could be, or not be, in proximity to each other. The framework contains different parts: server components, responsible for

[10] In the DP-3T there is a risk assessment, on the client side, which use the proximity, duration and number of encounters.

keeping track of devices' positions, and client components, which simulate Bluetooth and GPS capabilities. The health authority is also simulated, and it is part of the server side of the framework.

An outside observer can see the locations of simulated devices in real time on a map. Green markers symbolize a healthy user, red indicate declared carriers by the health authority and yellow illustrate those devices who have received a warning message. A hospital is marked as a red circle. Bluetooth LE signals are represented by a blue pulsing circle around each marker. When a green or yellow marker is placed in the hospital it becomes red (indicating that the person is found to be a carrier of the virus) and when it is placed outside the hospital it becomes green (symbolizing the recovery of the patient).

## Code Description

The code is divided into parts which represents the application on one side, and the framework on the other.

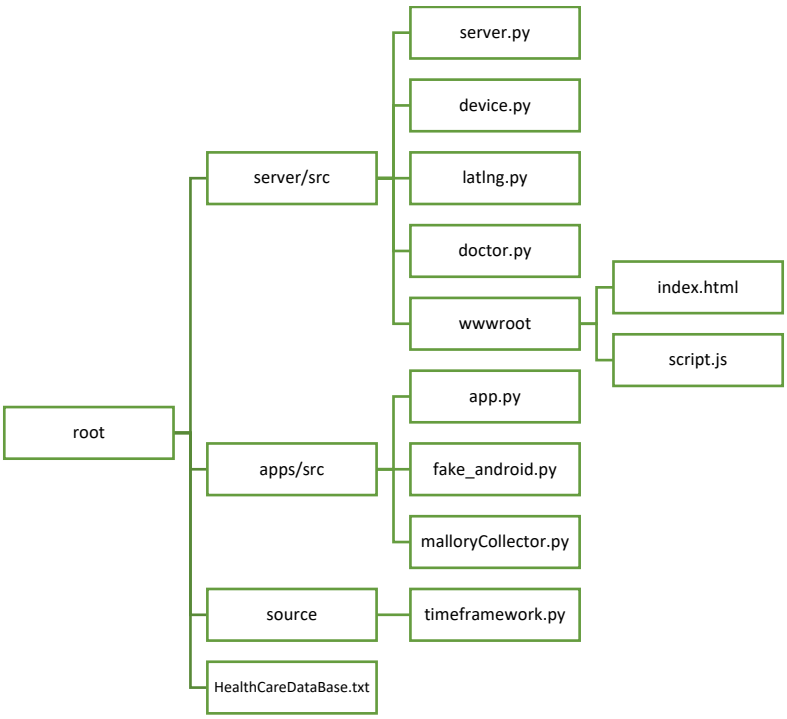


*Figure 1 File structure*

The *apps* directory contains the files: *app.py*, *fake-android.py* and *malloryCollector.py*. The *app.py* file contains the implementation of the DP-3T protocol, structured as a Python class

with three main threads. The first thread broadcasts `EphIDs` generated from a secret key (`SK`). The secret key is generated using the cryptographic hash function SHA256 in the method *generate_key* and it is stored together with a timestamp corresponding to the date of today. From the first `SK` (called $SK_0$) sequential `SK`s are generated each day in the method *get_next_key*. This implements: $SK_t = H(SK_{t-1})$ as in the protocol description.

```
def generate_key(self):
    #generate a crypthographically strong random secret key using SHA256
    if self.key0 is None:
        self.key0 = secrets.token_bytes(16)
        self.key0 = self.get_next_key(self.key0)
        self.key0_time = timeframework.get_today_index()
        self.key = self.key0
    else:
        self.key= self.get_next_key(self.key)

def get_next_key(self, key):
    sha = hashlib.sha256()
    sha.update(key)
    return sha.digest()
```

*Figure 2 Implementation of the SK*

From each daily `SK`, `EphIDs` are created in the method *generate_ephids*, one for each 15-minute period of the day, using the HMAC-SHA256 cryptographic hash function, and an AES block cipher in counter mode. The method contains the implementation of:

$$EphID_1 || .... || EphID_n = PRG(PRF(SK_t; \text{"broadcast key"}))$$

as specified by the protocol.

The `PRF` (pseudo random function) is called in two steps: first an *hmac* object is created with the secret key passed in the object and then, the string "broadcast key" is combined with the secret key producing a hash of the message, which is then used as a seed for the pseudo-random generator.

The `PRG` (pseudo random generator) is called by creating a 128-bit counter, an AES block cipher, and then encrypting a stream of blocks using the counter, which creates all `EphIDs` for one day. Finally, the `EphIDs` are split into 16-byte chunks to form a list, which is then shuffled so that `EphIDs` can be broadcasted in a random order.

```python
def generate_ephids(self, key):
    """ephids are generated by hashing the string broadcast key"""
    eph_ids = []
    #create a hmac object#
    hashed_key= hmac.new(key, digestmod='sha256')
    # create a pseudo-random number#
    hashed_key.update("broadcast key".encode("utf-8"))
    prf = hashed_key.digest()
    #create a counter of 128 bits#
    counter = Counter.new(128)
    #create a pseudo number generator#
    cipher = AES.new(prf, AES.MODE_CTR, counter= counter)
    #generate all ephids for one day#
    ephids_all = cipher.encrypt(bytes(EPHIDS_SIZE * EPOCHS_PER_DAY))
    #dived all ephids into chuncks#
    eph_ids = [
        ephids_all[i:i+EPHIDS_SIZE]
        for i in range (0, len(ephids_all), EPHIDS_SIZE)
    ]
    #shuffle the ids#
    random.shuffle(eph_ids)
    return eph_ids
```

*Figure 3 the generation of the EphIDs in several steps*

Another thread listens to incoming `EphIDs`. A file, which represents the memory of the phone, is created to hold all the collected `EphIDs` from devices nearby in the method *receive_ephid*. In the file there are no duplicates of the same `EphID`.

```python
def start_listen_to_ephids(self):
    """starting the thread that listen to other devices"""
    bluetooth_scanner = BluetoothLeScanner(
        self.receive_ephid, self.udp_client)

def receive_ephid(self, ephid):
    with open(f'Heard_EphIds{self.name}.txt', 'a') as file:
        if ephid not in self.unique_id:
            self.unique_id.add(ephid)
            file.write(ephid.hex())
            file.write('\n')
```

*Figure 4 save the collected EphIDs in a file*

The third thread downloads the `SK`s and timestamps from the healthcare database. Those `SK`s represent the first day in which a person has been declared contagious by a doctor. The download is represented by opening a file called *HealthAuthorityDataBase.txt* where those `SK`s are placed. `EphIDs` are generated in a loop from the timestamp in the file to today's date, by using the same algorithm as for generating the device's own `EphIDs`. In this way the health

care database doesn't need to collect all `EphIDs` but just one `SK` per infected. This thread is also responsible to check if there is a match between the collected `EphIDs` and the recreated `EphIDs`. If a match occurs, a warning message is displayed.

```
with open('healthCareDataBase.txt', 'r') as file:
    sk_infected_list = file.readlines()
    #create ephIds#
    for sk_and_time in sk_infected_list:
        sk = sk_and_time.split(',')[0]
        sk = bytearray.fromhex(sk)
        timestamp = sk_and_time.split(',')[1]
        timestamp = int(timestamp.strip())
        # loop from time of sk to today in simulated world
        for t in range(timestamp, timeframework.get_today_index()):
            infected_ephids = self.generate_ephids(sk)
            sk = self.get_next_key(sk)
            for infected_ephid in infected_ephids:
                # check if there is a match
                if infected_ephid in self.unique_id:
                    print("Warning. You have been in contact with a carrier of COVID-19.")
                    print("Please isolate yourself and check if you are a carrier")
                    # communicate with the simutation framework
                    urllib.request.urlopen(
                        f'http://localhost:8008/warning?name={self.name}')
                    #remove infected ephids from heard ids#
                    self.unique_id = self.unique_id - \
                        set(infected_ephids)
                    return
```

*Figure 5 recreate the EphIDs from the SKs and check if there is a match*

Finally, a method *upload_key_and_time* is responsible for adding the `SK` for the first day of being contagious, together with a timestamp to the *HealthCareDataBase.txt* file. This timestamp comes from the *Doctor* class in the simulation framework.

```
def upload_key_and_time(self, time):
    """it takes the first day of being contagious and
     it creates its Sk for that day and
       and uploads its Sk and timestamp"""
    sk = self.key0
    for k in range(self.key0_time, time):
        sk = self.get_next_key(sk)
    with open('healthCareDataBase.txt', 'a') as output:
        output.write(f'{sk.hex()} , {time}\n')
    # create a new key0 next day
    self.key0 = None
```

*Figure 6 upload SK and timestamp to the healthcare database*

**Framework overview and components**

The simulation framework is divided in several components, most of them contained in the *server* folder. Inside that folder, there is the *server.py* file which is the brain of the framework and knows everything about all devices sending Bluetooth signals; the *device.py* file simulates one device's position; the *latlng.py* file computes distances and directions; *doctor.py* represents the health authority, and finally the *index.html* and *script.js* are used to create the visualization of the simulated world in the browser.

In the *apps* folder the *fake-android.py* file is responsible for the simulation of APIs for the Bluetooth and GPS functionalities for the simulated mobile phone. In the simulation, time is sped up by a factor of 720 so that one day corresponds to 2 minutes in real time. The *timeframework* module is responsible for handling time in the simulated world and it is contained in the *source* directory.

Without going too much into detail[11], the *server.py*, *device.py* and *latlng.py* files contain code to handle positions and to create a simulation of the proximity of devices. Whenever a simulated mobile phone broadcasts a Bluetooth LE advertisement, the server receives the payload and determines which other devices are physically close enough to receive the payload, and then relays the original advertisement to the recipients. Bluetooth LE communication between devices is simulated using UDP socket communication.

The GPS position has also been simulated[12]. Location updates are sent from the simulation server to the simulated mobile phones as UDP datagrams, and the position is handled on the client side in the *fake_android.py* file. The health authority is also simulated in the *doctor.py* file. It computes an infection window: the period of time by which an individual is considered contagious and communicates this period to the device.

To create the frontend on which the devices are displayed on a map, JavaScript and HTML is added. At the beginning, devices are created, as markers, on a Google Maps view. When a

[11] The main focus of the project is not the simulation framework, so I decided not to explain it in detail because it is out of scope.
[12] The DP-3T protocol doesn't use the GPS position but it is necessary to simulate how devices interact in a simulated world.

device moves, information about the new position is sent to the web browser from the server over a WebSocket connection. The browser then renders the new position on the Google Maps view in real time. The same mechanism is also used for visualizing broadcasts made by devices as circles on the map.

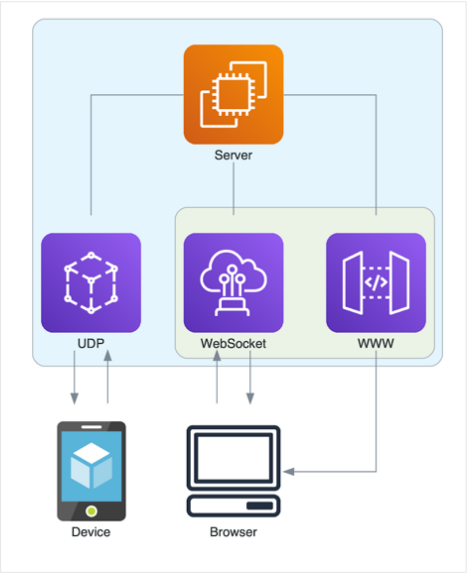


*Figure 7 overview on the simulation framework*

## 7. Attack scenarios and mitigations

The DP-3T does not need to exchange or store information regarding user's identity or location to be useful in contact tracing. However, there are some aspects which are still not clearly defined. The protocol doesn't specify how to secure different communication channels, leaving the door open to various abuse. Attacks could target society, to spread panic and overload the health care system, or individuals to harass and blackmail. Moreover, certain types of attack could use the weaknesses of the protocol to create a future mass surveillance state [12].

The DP-3T uses three main **channels of communication**: the first one (1) occurs when the application downloads the `SKs` from the backend, the second one (2) between the application and the health authority database, when the application uploads its (`SK,t`), and the third one (3) is created between peer applications when `EphIDs` are broadcasted [13]. Security should be implemented on all three channels.

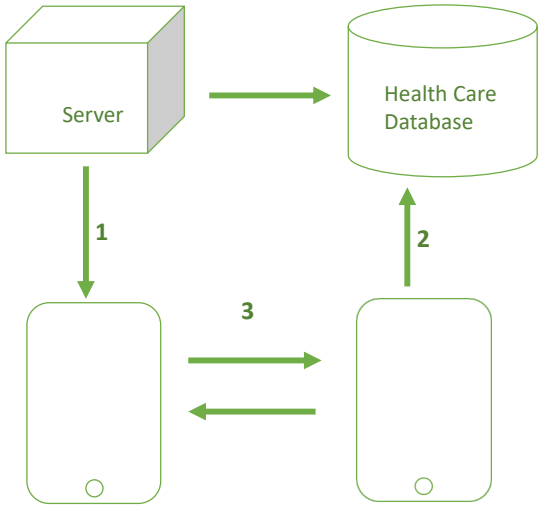


*Figure 8 Communication channels*

An attacker could, for example, exploit the **first channel** by creating a random `SK`, generating `EphIDs`, broadcasting them nearby the victim and finally impersonating the server, by for example manipulating the DNS records or the victim's router, to send the `SK` to the victim[13]. The victim then believes that they have been in contact with a declared infected, when in reality the SK comes from the impersonated backend. A variation of this attack consists in adding the `SK` directly to the server. This kind of attacks can be mitigated by using a digital signature when downloading the `SK` to verify the authenticity of the backend [13] or a decentralized blockchain could be used to keep track of the `SKs` instead of a centralized backend database, this solution has been proposed by the PRONTO-C2 protocol [12]. With a decentralized solution there is no single backend to impersonate and the attack would not be possible.

The **second channel** can be exploited also by impersonating the backend server and storing the `(SK,t)` uploaded together with the IP address. The IP address could be used to deanonymize the carrier. This attack can be mitigated by encrypting the communication from the app to the authority and using TOR or VPN to hide the IP address of the uploader [13, 12].

A malicious actor can also easily exploit the **communication channel between applications**. The attacker could derive `EphIDs` from the list of the `SKs` downloaded from the backend and broadcast them around[14]. Victims who receive `EphIDs` believe they have been in contact with a carrier and turn to the health care. A countermeasure for this kind of attack can easily be done

[13] This attack is called False Alert Injection Attack [13].
[14] Replay of Released Cases attack [13].

by creating a new SK after uploading the old one or giving a validity date to the SK. Together with the SK and timestamp an expiration date could be, for example, uploaded [13].

EphIDs can also be easily collected and massively resent to other devices creating a large number of false positives [15]. This could be done on a large scale, using many attackers, drones or Bluetooth devices with antennas and high signal strength. Unfortunately, this is much harder to defeat. A partial mitigation could be limiting the life length of the broadcasted EphIDs by tagging them with a secure hash of the EphID and a timestamp. The receiving device checks the validity of the timestamp, rejecting old EphIDs. This solution keeps the broadcasting model and makes the attack harder but still possible (the attacker needs to replay very quickly the EphIDs collected) [13].

Another solution would be to use an active handshaking instead of a broadcasting model. The EphIds would be sent to the receiver which would pick a random number (challenge) and send it back. The sender would then respond with a cryptographic hash of its SK and the challenge picked by the receiver. In this way, the receiver could verify that the EphID comes from the real sender, if the SK of the sender later is downloaded from the database. This would also require an active connection to be established between the devices before exchanging EphID. Unfortunately, this comes with much higher cost in terms of energy use (it requires a regular Bluetooth connection) and it may lead to other privacy issues [13].

The unsecure communication channel between peer devices in sending and receiving EphIDs can be exploited to deanonymize the victim. This type of scenarios is particularly dangerous: the information can be used not only against an individual but also against groups as an instrument of surveillance [12].

To expose the identity of a victim which has been declared a carrier, the adversary could collect EphIDs by following known victims like a politician or celebrity[16] [13] and later if the victim's SK is uploaded, the adversary can deduce the EphIDs from that key and learn that the person has been infected. This information can be used to blackmail the victim or publish their identity. A variation of this attack can be done using a large number of passive devices placed

[15] Replay Attack [13]
[16] This attack is called Paparazzi attack.

in fixed locations and collecting the `EphIDs`. The attacker then compare the EphIDs recreated from the downloaded list of SKs and check for a match [12].

A tech-savvy attacker could also develop an application implementing the same protocol as the DP-3T[17] and install it on their own phone[18]. This malicious application could be used on a large scale, by for example many attackers or by placing devices in strategic points, like entrances to the subway, government buildings, hospitals. This application would collect the `EphIDs` together with the location and exact time of encounter [13]. If a person is declared sick and uploads their `SK`, the attacker can recreate `EphIDs` from the downloaded `SK` and recreate the location trails of that person thanks to the extra information collected before [5].

These attacks are very difficult to mitigate without modifying the protocol radically. A partial solution to the problem is to split the `EphID` into a number of overlapping shares and broadcasting those shares. A receiver must thus collect enough of the shares to reconstruct the full `EphID`. It would be very hard for militia attackers to collect EphIDs without actively following victims closer during some time. This solution has been proposed as a variation of the DP-3T [10].

## 8. Performing the deanonymizing attack

Because the main focus of this project is to demonstrate how privacy is protected, a deanonymization attack will be performed to test this assumption.

The tech-savvy attack will be performed to show how a malicious actor can easily develop their own application based on the protocol and use it to collect approximate locations of the encountered devices. The attacker will store its own location together with timestamp and `EphIDs` for each encounter in order to recreate the location trail of a person who upload their `SK` to the health care database.

[17] The DP-3T is open source.
[18] This attack is called nerd attack.

In order to perform the attack, a file has been added to the *apps* folder. The *malloryCollector.py* file contains a modified version of the DP-3T protocol. In the file there are two classes: *MalloryCollector* and *MalloryBoss*. The first one represents a malicious collector of `EphIDs`, locations and timestamps. This collector could be a person with a mobile phone, a drone or a device placed in a strategic place.

```
def receive_ephid(self, ephid):
    """store ephId, lat, lng, time in a map"""
    if ephid not in stolen_pos:
       stolen_pos[ephid] = list()
    if self.lat is None:
        return
    stolen_pos[ephid].append((self.lat, self.lng, time.time()))

def start_listen_to_gps(self):
    """subscribe to location updates from the device's gps"""
    self.location = LocationManager(self.udp_client, self.collect_location)

def collect_location(self, lat, lng):
    """when new location is known, store it"""
    self.lat = lat
    self.lng = lng
```

*Figure 9 Receiving EphIds and collected with the position and timestamp*

The class *MalloryBoss* represents an external agent which downloads `SKs` from the health authority database and uses them together with the collected information by *MalloryCollector*. The *MalloryBoss* checks if the collectors have had any encounters with the declared infected and builds a location trail which shows the path of the victim from the points collected.

```
#loop from time of sk to today in simulated world
for t in range(timestamp, timeframework.get_today_index()):
    infected_ephids = self.generate_ephids(sk)
    sk = self.get_next_key(sk)
    for infected_ephid in infected_ephids:
        #check if there is a match
        if infected_ephid in stolen_pos:
            #loop over tuples
            for (lat, lng, timestamp) in stolen_pos[infected_ephid]:
                location_trail.append((round(lat, 6), round(lng, 6), int(timestamp)))
#check if list is not empty, meaning that there is a match
if location_trail:
   #convert location_trail to json and ecode it into bytearray to send through udp
   location_trail_json= json.dumps(location_trail)
```

*Figure 10 use the information on location*

On the frontend side, a purple Mallory marker has been added and the possibility to draw a location trail on the map with the information sent from *MalloryBoss* through the simulation server to the browser.

```
case 'location_trail':
  //show the trail of infected
  let coordinates = trail.map(([ lat, lng ]) => ({
    lat: lat,
    lng: lng
  }));
  let polyline = new google.maps.Polyline({
    path: coordinates,
    strokeColor: 'red',
    strokeWeight: 10
  });
  polyline.setMap(map);
  break;
```

*Figure 11 JavaScript code to render the trail of the map*

When the attack is performed a number of normal users and "Mallorys" are created. They walk around randomly collecting `EphIDs` in shared memory which symbolizes a common database.

## 9. Results

Three simulations have been performed: two in order to demonstrate how the protocol would work in a real scenario and one which simulates a large scale deanonymization attack.

### The simulation of the DP-3T

The first simulation of the DP-3T protocol is made by using just four users. They can roam around and `EphIDs` are exchanged. In the simulation, five terminal windows appear (which are processes): one for the server, and one for each of the four devices. In addition, a browser window opens, and four green markers are rendered on a Google Maps view. The markers start moving randomly, sending simulated Bluetooth signals, displayed as blue circles. When two devices are in contact with each other, information on which devices are nearby is displayed on the server terminal.

The markers can be dragged and dropped near each other and dragged to the big red circle in the middle which represents the hospital. Once in the hospital, the marker becomes red and the contact tracing starts. Markers previously in the Bluetooth proximity of the declared carrier (red marker), become yellow and receive a warning message in their terminal window.

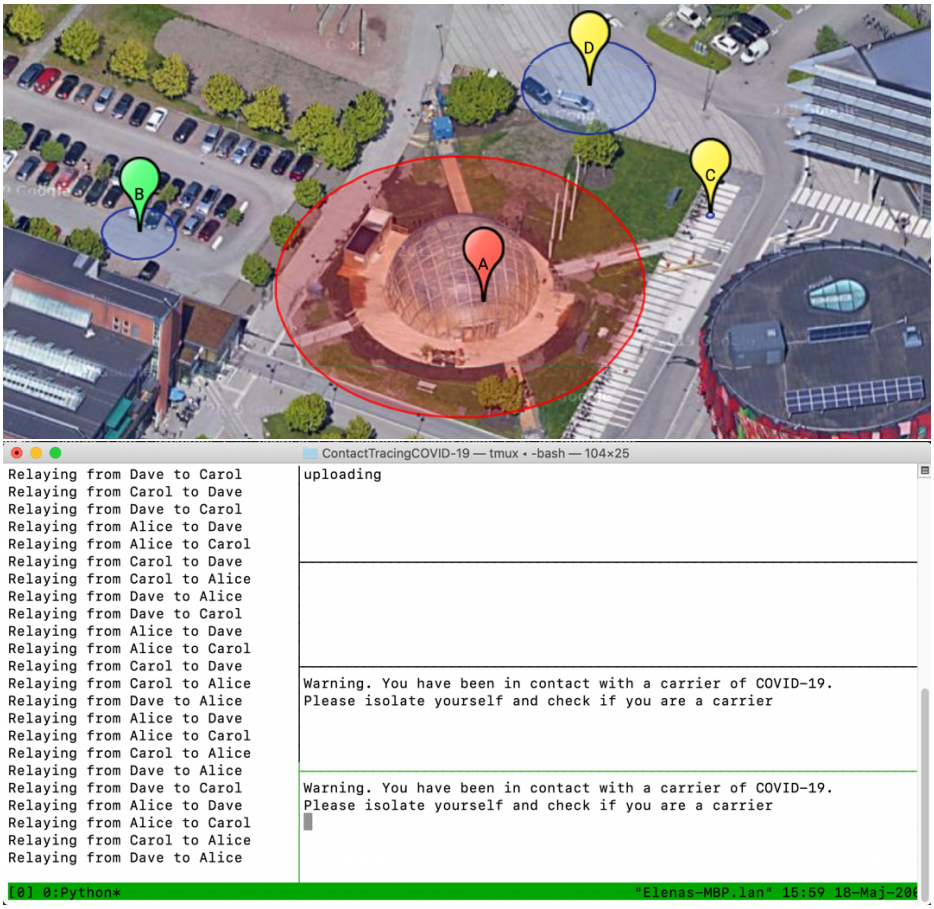


*Figure 12 The implementation of the protocol DP-3T.*

The other simulation is performed using 40 devices and the same procedure is applied. In this case only two terminals are activated: one for the server and one for all the devices.

It is possible now to play around with markers and see what happens. To make a marker healthy again, is it possible to drag it outside the hospital. In the code it is also possible to observe the *HealthCareDataBase.txt* file which fills up with `SK`s and the *Heard_EphIds.txt* file for each marker.


*Figure 13 The simulation running with 40 clients.*

### The attack

When the attack is run, 20 malicious actors are added in the scene. They are represented as purple markers and move around without sending any Bluetooth signals but just collecting them. After a couple of minutes, it is assumed that the malicious actors have collected enough data. When a standard user is then dragged to the hospital, a red path appears on the map showing the location trail for that user as recreated by *MalloryBoss*.

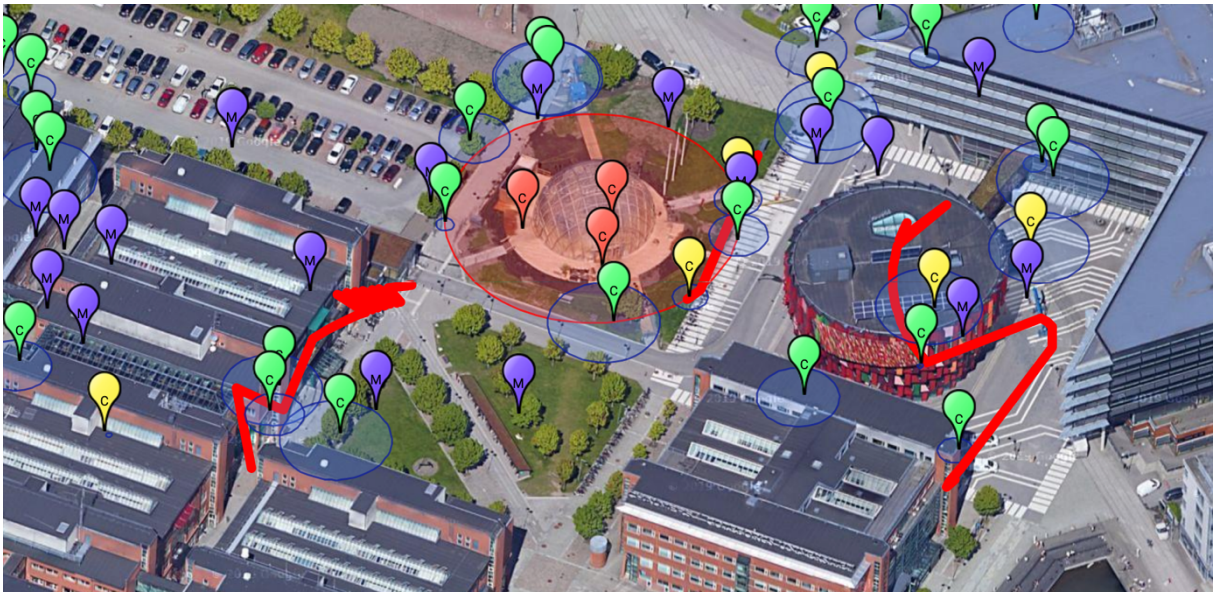

*Figure 14 Attack's result: 3 people in the hospital and 3 location trails for those people as recreated*

## 10. Conclusions

By building this prototype, I have demonstrated how the DP-3T contact tracing protocol could be used as an aid in the fight against the COVID-19 pandemic. I also showed that, even if location or personal information are not accessible to the authority or other users, they can be deduced by associating stolen Bluetooth beacons with a location.

The DP-3T is based on a decentralized design, which prevents a central authority from having access to personal information. It also uses Bluetooth Low Energy, which grants the application to run in the background and consume little energy, while having a good proximity precision. However, these choices come with a price: attackers could easily use deficiencies of the protocol and unsecure channels to deanonymize a large part of the population, and cause mass surveillance and privacy abuse.

Possible alternative solutions as well as countermeasures have been proposed by the researching community. Much is still to discover. The perfect protocol is far away from existing and in designing contact tracing applications, difficult trade-offs need to be made, between privacy and utility, simplicity and complexity.

## Acknowledgment

Thank you to Professor Andrei Sabelfeld at Chalmers University of Technology, for the supervision in this project.

## Sources

[1] S. Stirbys, O. Abu Nabah, P. Hallgren and A. Sabelfeld, "Privacy-Preserving Location-Proximity for Mobile Apps".

[2] P. Hallgren, M. Ochoa and A. Sabelfeld, "InnerCircle: A Parallelizable Decentralized Privacy-Preserving Location Proximity Protocol".

[3] S. Shaham, S. Rafieian, M. Ding, M. Shirvanimoghaddam and Z. Lin, "On the Importance of Location Privacy for Users of Location Based Applications," November 2019.

[4] M. Terrovitis, "Privacy preservation in the dissemination of location data," *SIGKDD Explorations,* vol. 13, July 2011.

[5] A. Fraunhofer, ”Pandemic Contact Tracing apps: DP-3T, PEPP-PT NKT and ROBERT froma a Privacy Perspective,” 27 April 2020.

[6] R. Raskar, I. Schunemann, R. Barbar, K. Vilcans, J. Gray, P. Vepakomma, S. Kapa, A. Nuzzo, R. Gupta, A. Berke, D. Greenwood, C. K. Keegan, S. Kanaparti, R. Beaudry och D. Stansbury, ”Apps Gone Rogue: Maintaining Personal Privacy in an Epidemic,” 19 March 2020.

[7] "Wikipedia," 18 May 2020. [Online]. Available: https://en.wikipedia.org/wiki/COVID-19_apps.

[8] M. S. Kim, "MIT Technology Review," 6 march 2020. [Online]. Available: https://www.technologyreview.com/2020/03/06/905459/coronavirus-south-korea-smartphone-app-quarantine/.

[9] A. JiJi, ”The Japan Times NEWS,” 13 May 2020. [Online]. Available: https://www.japantimes.co.jp/news/2020/05/13/asia-pacific/china-coronavirus-app/#.XsJXBxMzby9.

[10] EPFL, ETHZ, KU Leuven, TU Delft, University College London, CISPA, University of Oxford och University of Torino, ”github,” 12 April 2020. [Online]. Available: https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf.

[11] F. A. Inria, ”github,” 18 April 2020. [Online]. Available: https://github.com/ROBERT-proximity-tracing/documents.

[12] G. Avitabile, V. Botta, V. Iovinio and I. Visconti, "Toward Defeating Mass Surveillance and SARS-CoV-2: The Pronto-C2 Fully Decentralized Automatic Contact Tracing System," 6 May 2020.

[13] S. Vaudenay, ”Analysis of DP3T Between Scylla and Charybdis,” 8 April 2020.

[14] ”PEPP-PT,” May 2020. [Online]. Available: https://www.pepp-pt.org/.