Decentralized Privacy-Preserving Proximity Tracing

based on version: 3rd April 2020 – non authoritative version
Contact the first author for the latest version.

EPFL : Prof. Carmela Troncoso, Prof. Mathias Payer, Prof. Jean-Pierre Hubaux, Prof. Marcel Salathé, Prof. James Larus, Prof. Edouard Bugnion, Dr. Wouter Lueks, Theresa Stadler, Dr. Apostolos Pyrgelis, Dr. Daniele Antonioli, Ludovic Barman, Sylvain Chatel

ETHZ : Prof. Kenneth Paterson, Prof. Srdjan Capkun, Prof. David Basin, Dennis Jackson

KU Leuven : Prof. Bart Preneel, Prof. Nigel Smart, Dr. Dave Singelee, Dr. Aysajan Abidin

TU Delft : Prof. Seda Guerses

University College London : Dr. Michael Veale

CISPA : Prof. Cas Cremers

University of Oxford : Dr. Reuben Binns

TU Berlin / **Fraunhofer HHI** : Prof. Thomas Wiegand

Disclaimer/Provenance

This document is a manually created, LaTeX version, of the original document at https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf as as captured on 2020-04-08. As this document was only available as a PDF it was hard to collaborate in typical Open Source style.

This *non-authoritative, derived*, version is to facilitate easier edits/contributions and collaboration.

See https://github.com/DP-3T/ for the correct versions.

Executive Summary

This document proposes a system for secure and privacy-preserving proximity tracing (aka contact tracing) at large scale. Its goal is to simplify and accelerate the process of identifying people who have been in contact with an infected person, thus providing a technological foundation to help slow the spread of the SARS-CoV-2 virus. The system aims to minimize privacy and security risks for individuals and communities and guarantee the highest level of data protection.

We are publishing this document to seek feedback from a broad audience on the high-level design, its security and privacy properties, and the functionality it offers; so that further protection mechanisms can be added if weaknesses are identified. This document is accompanied by an overview of the Data Protection compliance of the design.

The goal of proximity tracing is to determine who has been in close physical proximity to an infected person, without revealing the contact's identity or where this contact occurred. To achieve this goal, users continually run a smartphone app that broadcasts an ephemeral, pseudo-random ID representing the user and also record pseudo-random IDs observed from smartphones in close proximity. Whenever a patient is diagnosed, he or she can upload some anonymous data from their phone to a central server. This step should only be done with the approval of a health authority and the consent of the individual. Before, no data is communicated to any entity. Other instances of the app can use the anonymous data from the server to locally compute whether the app's user was in physical proximity and potentially infected by an infected person and to inform the user of the risk. Additionally, the system enables users to voluntarily provide information to epidemiologists, in a privacy-preserving manner, to enable studies of the evolution of disease and to assist in finding better policies to prevent further infections.

The system provides the following security and privacy protections:

- **Ensures data minimization**. The central server only observes anonymous identifiers of infected people without any proximity information; health authorities learn no information (beyond when a user manually reaches out to them after being notified); and the epidemiologists obtain an anonymized proximity graph with minimal information.
- **Prevents abuse of data**. As the different entities in the system receive the minimum amount of information tailored to their requirements, none of them can abuse the data for other purposes, nor can they be coerced or subpoenaed to make other data available.
- **Prevents tracking of non-infected users**. No entity, including the backend, can track non-infected users based on broadcasted ephemeral identifiers.
- **Graceful dismantling**. The system will organically dismantle itself after the end

of the epidemic. Infected patients will stop uploading their data to the central server, and people will stop using the app. Data on the server is removed after 14 days.

Goals and Requirements

System Goals

1) Enable quick notification of contact persons at risk and give guidance on next steps
In Switzerland, the proximity tracing (also known as contact tracing) process is legally anchored in the Infection Protection Act and is carried out by the health authorities. Other countries have similar laws. The multi-stage procedure is ***time-consuming*** and requires a ***large number of trained personnel***. Under the current process, an employee of the health authorities conducts an in-person interview with an infected person to trace her or his contact history and identify other people who are likely to have contracted a disease. However, this process is slow and the ***results incomplete*** as usually patients are ***often unable to recall without gaps all contacts over a period of days***. Furthermore, ***random contacts*** (e.g., seat neighbours in public transport) ***cannot be identified*** and alerted.

Fortunately, most adults carry smartphones throughout the day, which opens the possibility of an app that can aid health authorities in their efforts to quickly and precisely notify all individuals who have been in close proximity to an infected person during an infectious period. We call the process that enables the health authority to learn whom to notify proximity tracing.

2) Enable epidemiologists to analyse the spread of SARS-CoV-2
Currently, there is a ***lack of detailed data*** on the spread of SARS-CoV-2. Epidemiologists are trying to understand the key factors in the spread of the virus. More precise and timely data would enable epidemiologists to improve their recommendations to policy makers and health authorities about the most important and effective measures during the containment phase of this and future pandemics.

The application should provide users with the possibility to ***voluntarily share data with epidemiologists*** and research groups to enable these groups to reconstruct the interaction graph among infected and at-risk users (referred to as a ***proximity graph***).

System requirements

1) Functional requirements
To achieve the system goals outlined above, the application must fulfill these functional requirements:

- **Completeness**: The contact history is **comprehensive** regarding contact events.
- **Precision**: Reported contact events **must reflect** actual physical proximity

- **Integrity**: Contact events corresponding to at-risk parties are **authentic** , i.e., users cannot fake contact events.
- **Confidentiality**: A malicious actor cannot access the contact history of a user
- **Notification**: At-risk individuals can be informed

2) Respect and preserve digital right to privacy of individuals
It is of paramount importance that any digital solution to enhance proximity tracing **respects the privacy of individual users and communities** and **complies with relevant data protection guidelines** such as the European General Data Protection Regulation (see EDPB Statement on GDPR and COVID-19) . The GDPR does not stop the use of data for public health, particularly in times of crisis, but it still imposes a binding obligation to ensure that 'only personal data which are necessary for each specific purpose of the processing are processed' (art 25). It is therefore a legal requirement to consider, particularly in the creation of systems with major implications for rights and freedoms, whether such a system could be technically designed to use and retain less data while achieving the same effect. To this end, an application must minimize the amount of data collected and processed to avoid risks for individuals and communities, and it should reveal only the minimum information truly needed to each authorized entity.

Furthermore, a common concern with systems like these is that the data and infrastructure might be used beyond its originally intended purpose. Data protection law supports the overarching principle of 'purpose limitation' — precluding the widening of purposes after the crisis through technical limitations. Such assurances will likely be important to achieve the necessary level of adoption in each country and across Europe, by providing citizens with the confidence and trust that their personal data is protected and used appropriately and carefully. Only applications that do not violate a user's privacy **by design** will be widely accepted.

The system should provide the following guarantees:

- **Data use**: Data collection and use should be limited to the purpose of the data collection: proximity tracing and proximity graph reconstruction. This implies that the design should avoid collecting and using any data, such as for example geolocation data, that is not directly related to the task of detecting a close contact between two individuals.
- **Controlled inference**: Inferences about individuals and communities, such as information about social interactions or medical diagnosis, should be controlled to avoid unintended information leakage. Each authorised entity should only be able learn the information strictly necessary to fulfill the functional requirement.
- **Protect identities**: The system should collect, store, and use anonymous or pseudonymous data that is not directly linkable to an individual's identity where possible.

- **Erasure**: The system should respect best practices in terms of data retention periods and delete any data that is not relevant.

3) Fulfill the scalability requirements posed by a global pandemic
SARS-CoV-2 is rapidly spreading across the globe due to the free movement of people across national borders and continents. As a core principle of free democracies, after the current confinement measures end, free movement should resume. Proximity tracing must support free movement across borders and scale to the world's population.

The system should give the following guarantees:

- **Scalability**: The system scales to billions of users.
- **Interoperability**: The system works across borders and health authorities.

4) Feasibility under current technical constraints
There is an urgency to not only design but ***implement*** a digital system that simplifies and accelerates proximity tracing in the near future. This mandates a system design that is ***mindful of the technical constraints*** posed by currently available technologies.

- **No reliance on new breakthroughs**: The system should, as far as possible, only use techniques and methods readily available at the time of development and avoid relying on new breakthroughs in areas such as GPS localisation or Bluetooth distance measurements.
- **Widely available hardware** : The goal of high adoption of proximity tracing can only be achieved if both server- and client-side applications can run on widely available smartphones and server hardware

Decentralized proximity tracing

We propose a privacy-friendly, decentralized solution that reveals minimal information to the backend server. To facilitate proximity tracing, smartphones locally generate ephemeral bluetooth identifiers (EphIDs) and broadcast them. Other smartphones observe these EphIDs and store them together with the duration and a coarse indication of time (e.g., "The morning of April 2"). See Figure 1.

The proximity tracing process is supported by a backend server that shares infection information with the app running on each phone. This backend server is trusted to not add or remove information shared by the users and to be available. However, it is untrusted with regards to privacy (*i.e., collecting and processing of personal data*). In other words, the privacy of the users in the system does not depend on the actions of this server. Even if the server is compromised or seized, privacy remains intact.

When patients are diagnosed with SARS-CoV-2, they will be authorized by national health authorities to publish information. Then, they will instruct their phones to upload to the backend a compact representation of their EphIDs for the infectious period. The backend stores these *compact representations*. Other smartphones periodically query the backend for this information and reconstruct the corresponding EphIDs of infected patients locally. If the smartphone has stored a record of any of these infected EphIDs, then the smartphone's user has been in contact with an infected person and the smartphone computes the owner's risk score. If this score is above the threshold the smartphone initiates a notification process.

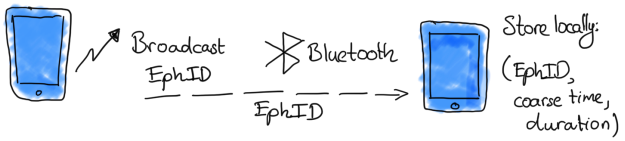

Figure 1: Processing and storing of observed EphIDs.

Setup

Generating a key. Smartphones generate a random initial daily key $\mathtt{SK}_0$. As we explain below, smartphones use hash chaining to compute daily secret keys $\mathtt{SK}_t$. From each day key, they generate a set of EphIDs to broadcast during that day. Smartphones store the last 14 keys $\mathtt{SK}_t$ (corresponding to an infectious window of 14 days, ultimately, the length of this window is a parameter that should be determined by the health authorities).

Creating ephemeral IDs (EphIDs)

EphID Constraints. Given the completeness requirement, it is necessary that smartphones can observe and record as many EphIDs as possible. This precludes the use of connection-based communication between smartphones, as establishing connections limits the amount of exchanges of EphIDs. Instead, we rely on Bluetooth Low Energy beacons. These beacons' payload is 16 bytes, which technically limits the size of our system's EphIDs.

EphID Generation. Smartphones generate a stream of secret day keys $\mathtt{SK}_t$, by computing

$$\mathtt{SK}_t = \mathtt{H}(\mathtt{SK}_t - 1),$$

where H is a cryptographic hash function. The smartphone will use the secret key $\mathtt{SK}_t$ during day t to generate EphIDs.

Smartphones locally generate $\mathtt{EphID}_i$s to use during day t as follows. Let t be the current day, and n the number of distinct EphIDs we must generate for that day. Then the smartphone computes

$$\mathtt{EphID}_1 \,||\, \ldots \,||\, \mathtt{EphID}_n \;=\; \mathtt{PRG}(\,\mathtt{PRF}(\mathtt{SK}_t,\, \backslash\mathtt{broadcast\ key"})\,)$$

where PRF is a pseudo-random function (e.g., HMAC-SHA256), "broadcast key" is a fixed and public string, and PRG is a stream cipher (e.g., AES in counter mode, or Salsa20) producing $\mathtt{n} * 16$ bytes, which we split into 16-byte chunks to obtain the n ephemeral Bluetooth identifiers EphID of the day.

Smartphones pick **a random order** in which to broadcast the EphID during the day. Each EphID is broadcast for $(24 * 60)/\mathtt{n}$ minutes

Local storage of observed EphIDs

Smartphones locally store each observed EphID together with the corresponding proximity, duration, and other auxiliary data, and a coarse time indication (e.g., "The morning of April 2"). See Figure 1.

Decentralized proximity tracing

The decentralized proximity tracing process requires the participation of infected patient's smartphones, all other smartphones, the backend, and the health authority. The backend acts **solely** as a communication platform and does not perform any processing.

Once the health authority triggers proximity tracing for an individual (Figure 2, step 1), the patient instructs their phone to send to the backend the key $\texttt{SK}_t$ corresponding to the first day in which the app user was considered to be infectious (Figure 2, step 2). Note that given the key $\texttt{SK}_t$, everyone can compute all ephemeral identifiers EphID used by the infected patient starting from epoch t by repeating the process described in "EphID generation" above.

After reporting their current $\texttt{SK}_t$, the smartphone of the infected patient picks a new completely random key

Periodically, the backend sends the $\texttt{SK}_t$ of infected patients to all other smartphones in the system (Figure 2, step 3). If an update is needed, it can also send to the smartphones the latest risk-scoring algorithm parameters provided by the health authority. Each smartphone uses this key to reconstruct the list of EphIDs of an infected person and checks if it has observed any of these EphIDs in the past (i.e., before the corresponding key $\texttt{SK}_t$ was published). If so, the smartphone owner may be at risk. The smartphone uses the risk-scoring algorithm with its local records corresponding to the infectious EphID , to determine the owner's risk score. See Figure 2 step 4.

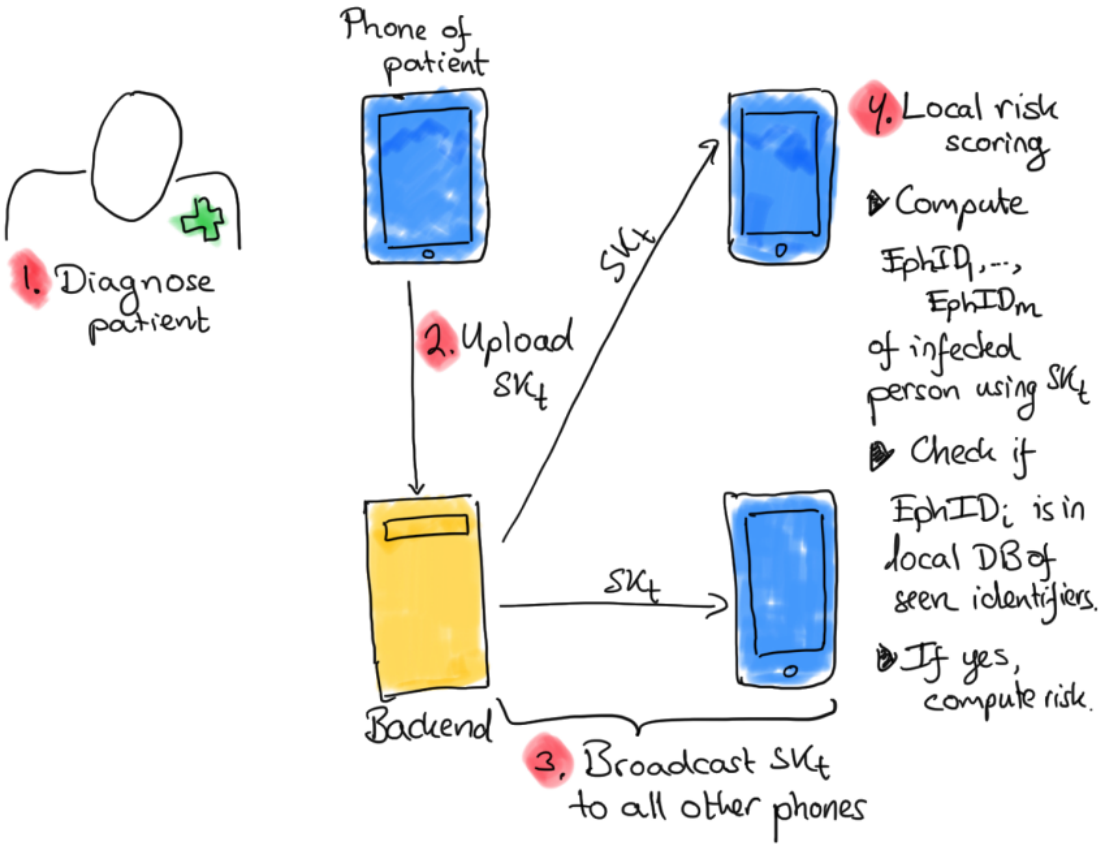

Figure 2: proximity tracing process.

Notification of Risk

If the risk score computed in the previous step is below a threshold determined by the health authority, the smartphone does nothing. Otherwise, the smartphone displays a notification that the user has been in close proximity to an infected patient. The notification advises the user on what to do and where to find more information.

Interoperability

To enable interoperability between countries, the smartphone records the countries that a user has visited. This can either happen automatically, in the background, based on GPS location data or through a manual entry by the user (e.g., if GPS was not available or inaccurate). To learn if a user has been in contact with an infected patient, the phone regularly requests data (containing $\mathtt{SK}_t$'s from infected users) from the backend services of all visited countries. The addresses of these backends can be hardcoded in the app. In case of a positive diagnosis, the smartphone uploads its own key $\mathtt{SK}_t$ to

the backends of all visited countries.

When the smartphone determines its owner has a high risk score, the smartphone contacts its local health authority as if it was not roaming.

Sharing data with epidemiologists

When installing the app, users will be asked if they want to opt-in to sharing data with epidemiologists in case they have been in contact with an infected person. If users opt in, the app will regularly (e.g., every day) upload a dummy package to the epidemiologists to defeat traffic analysis. No *real data* is uploaded at this time.

After a patient receives a notification that they are now at risk, the app again asks users to confirm that they still want to opt-in to sharing data with the epidemiologists. If so, at the next transmission time, the app will send *anonymous* data about all contact events the user had with each *infected individual* over the past period to a selected research center. In our scheme, *infected individuals* are represented by their keys $\mathtt{SK}_t$. The data shared includes a tag that indicates whether the user herself has been infected and information about the duration of encounters or the distance between the two individuals. However, any location or precise timing information about contact events will never be shared. The data submitted enables epidemiologists to study the *proximity graph* around an *infected individual* and understand which circumstances and encounters led to an infection. However, it does not reveal any information about other encounters the user has had with non-infected people.

Scalability

The decentralized design scales very well. For each infected user, the backend needs to store a 32 bytes key for the duration of the infectious window. Storage cost at the backend is therefore not a problem. Throughout the day, smartphones download the 32 byte keys of newly diagnosed patients. This data is static, and can therefore be effectively served through a content delivery network.

Smartphones download a small amount of data every day. For 40.000 new infections per day, smartphones download 1.25 MB each day. They require a few seconds of computation time to regenerate the ephemeral keys `EphID`, and to check if they are included in the local list of observed `EphIDs`.

Security and privacy considerations

Threat model

Regular user. A typical user of the system that is assumed to be able to install and use the application by navigating its user interface (UI). They will exclusively look at information available via the app UI to infer private information about other users.

Tech-savvy user (Blackhat/Whitehat hacker, NGOs, Academic researchers, etc.) . This user has access to the system via the mobile App. Can set up (BT, WiFi, and Mobile) antennas to eavesdrop locally. Can decompile/modify the app. Can have access to the backend source code.

- (Whitehat hacker) Will investigate the App code, the information in the phone, and will look at what information is exchanged with the server (using an antenna or software installed on the phone, e.g., Lumen) or broadcast via Bluetooth (passive).
- (Malicious) Can DOS the system (targeted or system-wide), deviate from protocols, and actively broadcast Bluetooth identifiers.

Eavesdropper (Internet Service Provider, Local System administrators, Bluetooth sniffer). Can observer network communication (i.e., source and destination of packages, payload, time) and/or Bluetooth BLE broadcast messages.

- (Network adversary) Can use observed network traffic to determine the state of a user (e.g., whether they are at-risk, infected, etc.)
- (Local Bluetooth BLE Sniffer) Can observe local Bluetooth broadcasts (possibly with a big antenna to cover a wider area) and try to trace people.

It should be noted however that in many instances, for individuals or companies to use data in this way, such as to collect data about passers-by to try and estimate their infection status based on the announced identifiers, will fall foul of a range of existing national and European laws around data protection, ePrivacy and computer misuse.

Health authority. Receives information about infected people as part of their normal operations to diagnose patients. The health authority learn information about at-risk people only when these at-risk people themselves reach out to the health authority (e.g., after receiving a notification from their app).

Epidemiologists. Receive and analyse data about interactions between infected and at-risk users. Epidemiologists and related research groups are not in direct contact with app users and data is shared with them on a voluntary basis. They are mainly

interested in learning the proximity graph around an infected user but can combine background knowledge about individual users with the inferred graph data to learn more about infected and at-risk users.

Backend. Can access all data stored at the servers. Moreover, the backend can query data from the mobile app in the same way that it would do during normal operations (in our system, it can only send). They could also change the code of their backend software and the code of the mobile apps. We assume they will not modify the mobile app because doing so would be detectable. They can combine and correlate information, request information from apps, combine with other public information to learn (co-)location information of individuals.

State-level adversary (Law enforcement, intelligence agencies). Has the combined capabilities of the tech-savvy user and the eavesdropper. In addition, a state-level adversary can obtain subpoenas that give them the capabilities of the health authority, epidemiologists or the backend. Their goal is to obtain information about the population or to target particular individuals. They may be interested in past information, already stored in the system, or future information that will enable them to trace target individuals based on observed `EphIDs`.

Privacy

Privacy Concerns

Global interaction graph. The global interaction graph reflects the social relationships of all users in the system. A labelled edge indicates an interaction between two adjacent users at a specific time. Knowledge of this graph is not necessary for proximity tracing nor for analyzing the spread of SARS-CoV-2. Therefore, *no party* needs to learn the global interaction graph. Only the relevant subset of this graph, such as for example the proximity graph (below), should be revealed to the authorised entities.

Proximity graph. The proximity graph is a subset of the global interaction graph. It encodes contacts between infected users and others individuals. Every edge is adjacent to at least one infected patient. Edges can be labeled with coarse-grained timing information (e.g., “contact on the 4th of April”). This information is *essential for epidemiologists* but not for any other party in the system.

Infected individuals. Only the individuals themselves, and the health authority, need to know that they are infected with SARS-CoV-2. No other parties in the system need to learn this information. In particular, infected users do not need to know which of the individuals they have been in contact with are infected.

At-risk individuals. At-risk individuals are people that have recently been in contact with somebody who has been infected with SARS-CoV-2. At-risk individuals need to

know that they are at risk so that they can take appropriate measures. The health authority needs to know who is at risk so that they can notify them. No other parties in the system need to know this information.

Location traceability. To perform proximity tracing or to analyze the virus' spread, location traces are not required, e.g. GPS coordinates. Therefore, no party in the system needs to have access to them, or be able to easily trace individuals based on the Bluetooth signals that the apps broadcast.

Privacy Analysis

Interaction graph. The system does not reveal any information about the interaction between two users to any entity other than the two users themselves. The $\texttt{EphIDs}$ revealed by infected users do not allow any inference about the people they have been in contact with. The system thus prevents any one party from learning the interaction graph.

Proximity graph. By construction, the proximity graph (containing interactions between infected patients and other users) is only revealed to epidemiologists, with the specific, separate consent of a user, and not to any other party.

Location traceability. The $\texttt{EphIDs}$ of all users are perfectly unlinkable, and only the smartphone that generated them knows the corresponding keys $\texttt{SK}_\texttt{t}$. When the phone's owner is diagnosed with SARS-Cov-2, the phone publishes to the backend the key $\texttt{SK}_\texttt{t}$ corresponding to the first infectious day. After disclosing this information, the phone will generate a new key at random. Given this key $\texttt{SK}_\texttt{t}$, all corresponding $\texttt{EphIDs}$ of the infected person are linkable during the infectious period.

As a result, tech-savvy users, eavesdroppers, and state-level adversaries with strategically placed Bluetooth receivers and recording devices can track infected patients during the (earlier) window in which the identifiers broadcasted via Bluetooth are linkable. The app's Bluetooth broadcasts of non-infected people and infected people outside the infectious window remain unlinkable.

At-risk individuals. The keys revealed to the server by infected people are independent of their contacts, i.e., the people they interacted with. They therefore do not give any information about people at risk.

Smartphones locally compute at-risk status. The smartphone then relays this at-risk status to the health authority so that the health authority can contact the phone's user. The health authority therefore learns who is at risk.

If the user consents, the smartphone uploads anonymized data about the user's encounters with infected individuals to epidemiologists. However, the information shared does not include any pseudonyms or anonymous identifiers and does not allow linking

contact events to identities. Epidemiologists therefore cannot learn who is at risk, but they can study the spread of the virus through the proximity graph.

Infected individuals. Any proximity tracing mechanism that informs a user that he/she has been in contact with an infected person inherently reveals a piece of information to the person at risk: one of the people they interacted with has been infected. For the purpose of this analysis we will divide these people in three categories:

- *Close individuals*: Family, friends, or colleagues with whom the at-risk individual spends long periods of time. If these people are infected, they will inform the at-risk user personally about their infection if they have spent time together. It is common practice that the authorities ask COVID-19 patients to notify any contact person at risk they remember.
- *Routine-sharing individuals*: People who share an activity with the at-risk individual such as riding a bus every day, supermarket tellers, etc. Infected individuals in this group will likely not remember having been in contact with them and therefore will not (and can not) notify the at-risk individual.
- *Anonymous individuals*: People that the user sees sporadically. These people cannot be re-identified as their identities are unknown to the at-risk individual. They are therefore out of the scope for this analysis.

As close individuals will reveal themselves, there is no extra information that an at-risk individual can gain about the infection status of this group by exploiting the app. Anonymous infected persons cannot be identified, so their privacy is also not at stake. In the remainder of this analysis we thus focus on routine-sharing individuals.

First, and foremost, the app in normal operation does not reveal to users anything about the `EphIDs` downloaded from the backend. Therefore, a curious user, who only uses the standard interface of the app, cannot learn who is infected. As this user does not perform further actions, they cannot obtain any further information about who is infected other than the information provided to them through app notifications, health officials, or by infected individuals themselves.

On the other end, a proactive tech-savvy person can abuse ***any*** proximity tracing mechanism to narrow down the group of individuals they have been in contact with to infected individuals. To do so they must, 1) they keep a detailed log of who they saw when. 2) they register many accounts in the proximity tracing system, and use each account for proximity tracing during a short time window. When one of these accounts is notified, the attacker can link the account identifier back to the time-window in which the contact with an infected individual occurred. The attacker can correlate this information with the detailed log to narrow down who in their list of contacts is now infected. This attack ***is inherent to any proximity-based system notification system***, as the adversary only uses the fact that they are notified together with additional information gathered by their phone or other means.

In the decentralized system, tech-savvy adversaries can make this inference without having to create multiple accounts. To determine when they interacted with an infected individual, they **proactively modify the app** to store detailed logs of which EphID they received and when, and cross reference this list with the EphIDs they computed for each infected person. They then correlate these infection times with their log of who they saw when as before.

Tech-savvy users can also attempt a retroactive attack in which they attempt reidentification based on linkage, without the need to collect additional information in advance. The retroactive attacker only uses information stored by the app and auxiliary knowledge about the whereabouts of routine-sharing individuals during the infectious period. The data stored in the app provides coarse timing information when a specific EphID has been observed (e.g., up to 8 hour time windows). A tech-savy at-risk user could leverage this information to single out an infected individual based on matching observed EphIDs to the background knowledge about whom the at-risk user was with during this time window. A combination of multiple time windows might be enough to uniquely identify whom the infected EphIDs belong to. However, since smartphones broadcast the daily set of EphIDs in random order, the attacker cannot use the published keys SK t to narrow down this coarse time-window. This decreases the likelihood that she will be able to successfully identify the infected individual in her contact list.

Furthermore, an adversary operating its own BLE equipment from a single location can collect EphIDs within 20-100m range, depending on the phone output power and environment. When combining this list with the EphIDs that can be computed from the SK downloaded to the phone, an adversary could estimate the percentage of infected people in a small radius of 50m. If in addition, the adversary has a camera, he can capture images and potentially re-identify those people.

As for at-risk users, the pattern associated with the upload of identifiers to the server would reveal infected users to network eavesdroppers (ISP, curious WiFi provider) and Tech-savvy adversary. If these adversaries can bind the observed IP to a more stable identifier such as an ISP subscription number, then they can de-anonymize the infected user.

Summary. For an adversarial at-risk user to learn which infected individuals they have been in contact with, the following conditions must all be met:

- An adversary has to have access to a fine-grained log of who they have seen when and where.
- An adversary has to either modify the application to store fine-grained time measurements alongside each observed EphID or rely on the coarse time window given by the unmodified implementation
- The adversary and the infected individual must be alone for a long enough duration.
- If there are other people around, whether they are running the app or not, the

adversary cannot be certain of who is the infector unless the adversary sees this infector in a large enough number of epochs.

Note that if, in addition to these conditions, the adversary can create multiple accounts, then this adversary can identify the infector **regardless of the design of the proximity tracing system**.

We stress that in any case, having been close to an infected person is not a proof of causality regarding an infection. Moreover it is worth noting that reidentifying individuals and inferring their health status as a private entity without their permission would likely violate data protection law and, potentially, computer misuse law, which would further increase the cost and risk of undertaking this attack.

Mitigations. In the current setting, tech-savvy users can download and analyze the data of infected ids.
As one mitigation, we propose to allow infected individuals to control the periods of time that they consider sensitive and to not disclose the `EphID`s that their device generated during those sensitive periods. This can alleviate concerns that might arise e.g., in close-knit and smaller communities where in person notifications are better tools for informing people about infections and risks.
Another possible mitigation is the use of local trusted execution environments (TEEs) that, on each phone download and decrypt the list of infected EphID s and compute the local risk scores by cross referencing the list of infected `EphID`s with the collected beacons, and returning a risk-score to the app. Instead on the phone, the user could also delegate this functionality to a TEE that she explicitly trusts. Modern phones are equipped with TEE functionality and TEEs are already used to harden smartphone kernels against attacks and to store cryptographic keys. TEEs require buy-in from mobile platform providers (Apple, Google) and, for Android, the device manufacturers (Samsung, Huawei, etc.). The TEEs are well protected and hard to attack even for tech-savvy users. While it is not impossible to leak this information, it is unlikely. We think such a mitigation is worthwhile in a later version of the proximity tracing system to further increase privacy guarantees. Other mitigation techniques could include the use of Private Information Retrieval and Private Set Intersection techniques.

Such technical measures as well as non-technical measures (e.g., banning modified applications from the market) could be introduced in case that the identification of infected individuals would become a threat to the system operation and to the involved users. The introduction of such measures depends on the overall risk assessment.

Furthermore, the transmit power of BLE beacons needs to be reduced in other to limit exposure and risks related to eavesdropping attacks.

Finally, we note that if a small, extremely cautious portion of the population is concerned with these attacks and decides not to participate, this will not greatly impair the effectiveness of the deployment. As long as a large portion of the population runs the app, the number of at-risk identifications will be large enough to significantly reduce

the rate of infections.

Security

Security Concerns

Fake contact events. A fake contact event could make a person believe that they are at-risk, even though they have never been in contact with an infected person. Attackers could try to generate such fake contact events.

Suppressing at-risk contacts: There is a risk that either an infected person or the backend server could prevent other individuals from learning they are at risk, e.g., by modifying the app's local storage. This violates the integrity of the system and would lead to an increased health risk for contact persons who rely on the system to alert them.

Prevent contact discovery: A malicious actor could disrupt the system, e.g. by jamming Bluetooth signals, and prevent contact discovery.

Security Analysis

Fake contact events. Fake contact events cannot be completely avoided. A malicious tech-savvy user can always use a large antenna to artificially increase their broadcast range. If the broadcasted identifiers belong to a (later) infected person, then any recipient will conclude that they are now at risk.

In the decentralized system, smartphones compute risk scores purely based on the $\texttt{EphID}$s they have seen and the $\texttt{EphID}$s of infected people. Therefore, to create a fake contact event, an attacker must modify one of these two. The former can only be changed by using Bluetooth broadcasts. The latter can only be changed by (1) being infected with SARS-Cov-2, and then (2) reporting somebody else's key $\texttt{SK}_t$ so that that key is treated as infected.

Suppressing at-risk contacts. Hiding at-risk contacts is possible in any proximity tracing system. Infected users can choose to not participate at all; to temporarily not broadcast Bluetooth identifiers, or not to upload their data once diagnosed.

Prevent contact discovery. Any proximity tracing system based on Bluetooth low energy is susceptible to jamming attacks by active adversaries. Such jamming attacks will cause the normal recording of $\texttt{EphID}$s to stop working, hence preventing contact discovery. This is an inherent problem of this approach.

Comparison with a centralized approach

A centralized, on-demand trace upload

In a centralized approach, instead of the smartphones gathering information to learn whether the owner is at risk, it is the central server who identifies at-risk users and notifies those users' smartphones.

As the central server needs means to identify users, it must hold a long-term pseudo-identifier and must generate the ephemeral pseudo-identities (`EphID`s) to be pushed to the smartphones. As in the decentralized design, these `EphID`s can be produced, and sent to the smartphones, in epochs.

The smartphones broadcast the received `EphID`s and receive `EphID`s sent by nearby smartphones. Smartphones locally store all observed `EphID`s together with the corresponding proximity, duration, and auxiliary data. See Figure 3.

As in the decentralized design, when patients have been diagnosed and they are authorized, they can instruct their smartphone to send the recorded list of observations to the server to enable proximity tracing.

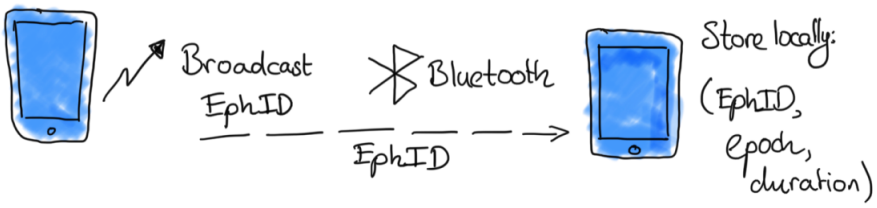

Figure 3: Processing and storing of observed `EphID`s.

Proximity tracing

The proximity tracing process is executed by the backend after a diagnosed patient has made available to the backend their list of observations [(`EphID`, epoch, duration)] for the relevant period of time. The backend recovers the long-term pseudo-identifiers of the at-risk users from the reported observed `EphID`s and triggers a process to identify them. See Figure 4.

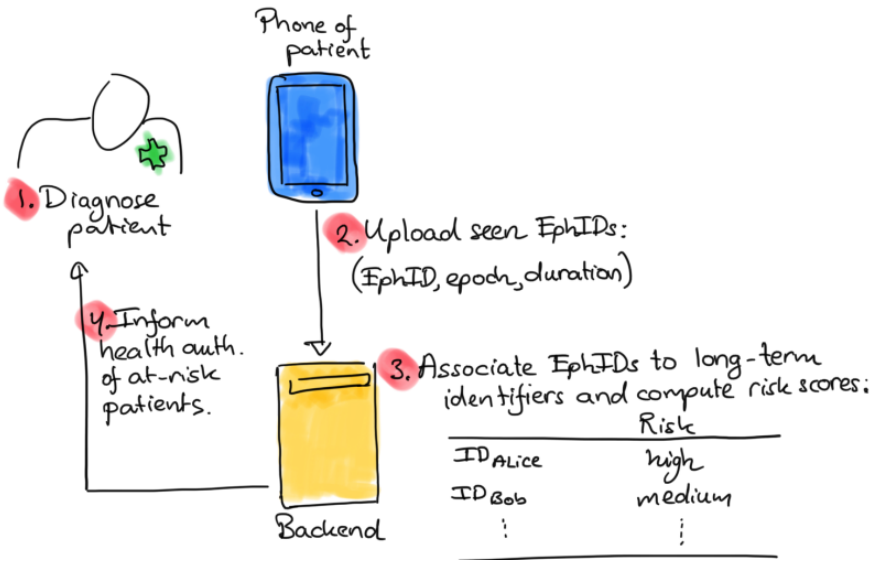

Figure 4: proximity tracing for centralized design

Interoperability

Because EphIDs are generated by the backend, if there exist more than one backend (e.g., one backend per country), each backend can only recover their own long term pseudo-identifiers. Therefore, if infected people have spent time in another country during the infectious period, the backend will receive observed EphIDs collected in that country that it cannot interpret. This means that the centralized design requires a "routing" mechanism to ensure that EphIDs arrive at the backends that can interpret them. Options to achieve this goal can be to include a country code in the EphID , or to broadcast non-interpretable EphIDs to all other backends to make sure that it reaches home.

Sharing data with epidemiologists

In the centralized design, the smartphones never learn which contact events correspond to contact events with infected individuals, and therefore cannot provide the relevant information to epidemiologists. To provide this service, the central server must keep all relevant information and share it with researchers.

Privacy Comparison

Interaction graph. The centralized system reveals the interaction graph of each infected user to the backend server. This is by design, as the server maps each time-

stamped ephemeral identifier back to a permanent pseudonym to enable contact tracing. The subset of the full interaction graph learned by the server grows quickly as every newly infected user uploads their entire contact history, which can be linked to existing nodes in the graph. Even though the nodes in the graph are pseudonymous, this is a serious privacy concern because graph data is easy to reidentify.

Proximity graph. While in the decentralized design, the proximity graph is only selectively revealed to researchers, the centralised design allows the backend server to learn this information as well. This violates the privacy requirement about limiting inference to minimum necessary and to authorised entities only.

Location traceability. The decentralized design limits the potential for location tracking to infected users over the course of the infectious period. In the centralised system, access to server-side keys (e.g., the backend itself or law enforcement) enables linking ephemeral `EphID`s to the corresponding permanent app identifier and thus tracing/identifying people based on `EphID`s observed in the past, as well as tracing people's future movements.

At-risk individuals. In the centralized design, by design the backend recovers the identity of the at-risk individuals to be able to notify the health provider about the fact that they are at risk. The health authority will naturally also learn their identities as they need to be contacted. The epidemiologists learn the same information than in the decentralized approach, and so does the eavesdropper that can monitor the exchange of the phone during at-risk notification.

Infected individuals. The centralised and decentralised contact tracing systems share the inherent privacy limitation that they can be exploited by an eavesdropper to learn whether an individual user got infected and by a tech-savvy user to reveal which individuals in their contact list might be infected now. However, the centralised design does not allow proactive and retroactive linkage attacks by tech-savvy users to learn which contacts are infected because the server never reveals the `EphID`s of infected users.

Security Comparison

Fake contact events. Creating fake at-risk events is easy in the centralised design and can be done retroactively by any tech-savvy infected patient. It does not require broadcasting. It suffices with adding the target `EphID`s to the list of observed events prior to uploading it to the backend.

Suppressing at-risk contacts. Hiding at-risk contacts is possible in any proximity tracing system.

Prevent contact discovery. Any proximity tracing system based on Bluetooth low energy is susceptible to jamming attacks by active adversaries.

	Decentralised	**centralised**
Privacy Concerns (who can learn what)		
Interaction graph	-	Backend / State-level
Proximity graph	Epidemiologist	Epidemiologist / Backend / State-level
Location tracking	Tech-savvy user	Backend / State-level
Of infected users	During infectious period	Always
Location tracking	Backend/state-level	Backend / State-level
Of non-infected users		Always
At-risk individuals	Tech-savvy user / Eavesdropper	Eavesdropper / Backend / State-level
Infected individuals	Tech-savvy user / Eavesdropper	Tech-savvy user / Eavesdropper
Percentage infected individuals	Tech-savvy external with antenna	State-level
Security Concerns		
Fake contact events	Yes Physical proximity + amplified broadcast (with knowledge of infected EphID)	Yes Infected tech-savvy user / Backend / State-level
Suppressing at-risk contacts	Yes Tech-savvy user (own contacts only)	Yes Tech-savvy user / Backend / State-level
Prevent contact discovery	Yes Tech-savvy user + broadcast	Yes Tech-savvy user / Backend / State-level

Conclusion

In this whitepaper, we have outlined a reference design for a decentralized approach to proximity tracing, and we have described evaluation criteria to assess the level of privacy provided by any proximity tracing solution. A key requirement driving our design is to minimize exposure of private data, limiting privacy leakage.

Our proposed decentralized design relies on smartphones to locally compute the risk for an individual user to have contracted the virus based on exposure to infected persons. Data about specific contact events, i.e. interactions between individuals, always remains on users' phones and risk calculation happens locally, according to the guidelines set by the health authorities. In addition, users may voluntarily and privately share data about interactions with infected persons (but never contact events itself) with epidemiologists to aid the investigation into the spread of SARS-CoV-2. The decentralised design gives users fine-grained control over the information they share and all data sharing happens under the user's explicit permission.

We have presented criteria for the evaluation of security and privacy aspects of proximity tracing and have thoroughly evaluated our decentralized design. Our design scales to a large number of users with minimal local computation and minimal centralization. Compared to a central design in which the backend would compute risks and inform users, our design protects interaction graphs from the backend, and only a determined tech-savvy adversary can learn any extra information besides the one made visible by the app. The centralized system, in comparison, leaks a lot of unnecessary information about contacts to the backend, and requires large amounts of trust in a central entity.

We strongly urge governments, health authorities, and researchers that any deployment of proximity tracing follows a decentralized design similar to our system to avoid the creation of centralized systems that have the potential to become surveillance infrastructures. We are currently working on a reference implementation of the decentralized design which will be released openly during the next weeks.