Whitepaper

ShareTrace: A Smart Privacy-Preserving Contact Tracing Solution by Architectural Design During an Epidemic

Authors (alphabetically listed)

Erman Ayday*, Case Western Reserve University
Fred Collopy, Case Western Reserve University
Taehyun Hwang, Cleveland Clinic,
Glenn Parry, University of Surrey
Justo Elias Karell, Pratt and Whitney
James Kingston, HAT-LAB
Irene Ng, Dataswift / University of Warwick
Aiden Owens, University School
Brian Ray, Cleveland State University
Shirley Reynolds, Case Western Reserve University
Jenny Tran, Century Business Solutions
Shawn Yeager, Dataswift
Youngjin Yoo*, Case Western Reserve University
Gretchen Young, Century Business Solutions

Version 1.0
May 3, 2020

Acknowledgement: ShareTrace is an open source project, based on Health Traffic Light project from #HackfromHome (April 4 - 5, 2020). This material is in part based upon work supported by the National Science Foundation (Grant No. 1447670). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

* Corresponding authors: Erman Ayday: exa@case.edu, Youngjin Yoo: yxy23@case.edu

1. Introduction

Contact tracing has emerged as an integral tool for containment during an epidemic such as the current novel coronavirus (COVID-19) crisis. As demonstrated in countries like South Korea, China and Singapore, smartphone-based automated contact tracing solutions have emerged as a powerful tool to assist government and healthcare authorities to rapidly respond to the public health crisis. As many countries are getting ready to reopen their economies, rapid and effective contact tracing together with timely and accurate testing will become an indispensable part of the collective efforts of restoring the economy and the society.

The potential benefits of smartphone-based contact tracing have already been discussed extensively elsewhere (Raskar et al. 2020, Troncoso et al. 2020, PEPP-PT, PACT, Google and Apple, Chan et al. 2020). While the benefits of such digital contact tracing are clear, the potential deployment of ubiquitous surveillance tools by state authorities or even by private firms raise serious concerns of privacy and civil liberties. In many of the tools that have been deployed or are being developed, what data is being collected, where and how they are stored, who will have access to them, and how they will be managed and purged are not clear.

Recognizing these concerns, several decentralized approaches are being proposed, most notably by the joint efforts between Apple and Google. Several academic institutions have also introduced similar solutions. One common feature in these privacy-preserving contact tracing solutions is the use of anonymized contact lists for collecting Bluetooth radio signals on a user's device. Most applications broadcast a frequently rotating Ephemeral ID (EphID) and receive those same IDs from other users who come within a designated distance of the user. Once a user tests positive for the Covid-19 virus, that user's EphID is broadcast to the network. If the broadcasted EphID matches with one of the EphIDs in the local proximity contact list in their device, the user is notified.

While more efficient and scalable than traditional manual contact tracing, these proximity-based decentralized contact tracing solutions suffer potential shortfalls both in their effectiveness and privacy protections. Particularly, the functionality of the proximity-based decentralized contact tracing is limited by the inherent architectural design choice that these solutions use to protect user privacy. As these approaches rely only direct contact history, they can be too slow in a highly viral epidemic like Covid-19.

In this white paper, we propose an alternative architecture that can provide more effective and timely solutions while providing better protection of user privacy. Specifically, we propose a distributed architecture that constitutes a network of anonymized digital twins of users using decentralized Personal Data Accounts (PDAs). PDAs are implemented using the HAT Microserver in the cloud. Using the HAT Microserver, we compute hyperlocal interaction graphs using anonymized digital twins. As we will demonstrate below, the use of hyperlocal interaction graphs offers our solution, ShareTrace, several advantages over other proximity-based solutions:

- ShareTrace provides much quicker and more effective notifications to individual users compared to other proximity-based solutions.

- ShareTrace’s distributed PDA architecture better protects privacy by eliminating the need to broadcast even anonymized user IDs and related information, which can be used to re-identify individual users in some circumstances.
- By using hyperlocal networks, ShareTrace allows users to “roam” to different locations, and immediately receive appropriate guidance even when traveling.
- By using PDAs, ShareTrace provides a *personalized* risk assessment for each user that is based on multiple data sources, not only potential contact with infected individuals.
- Second order information is available to users more promptly, so that they can alter their behaviors days earlier.

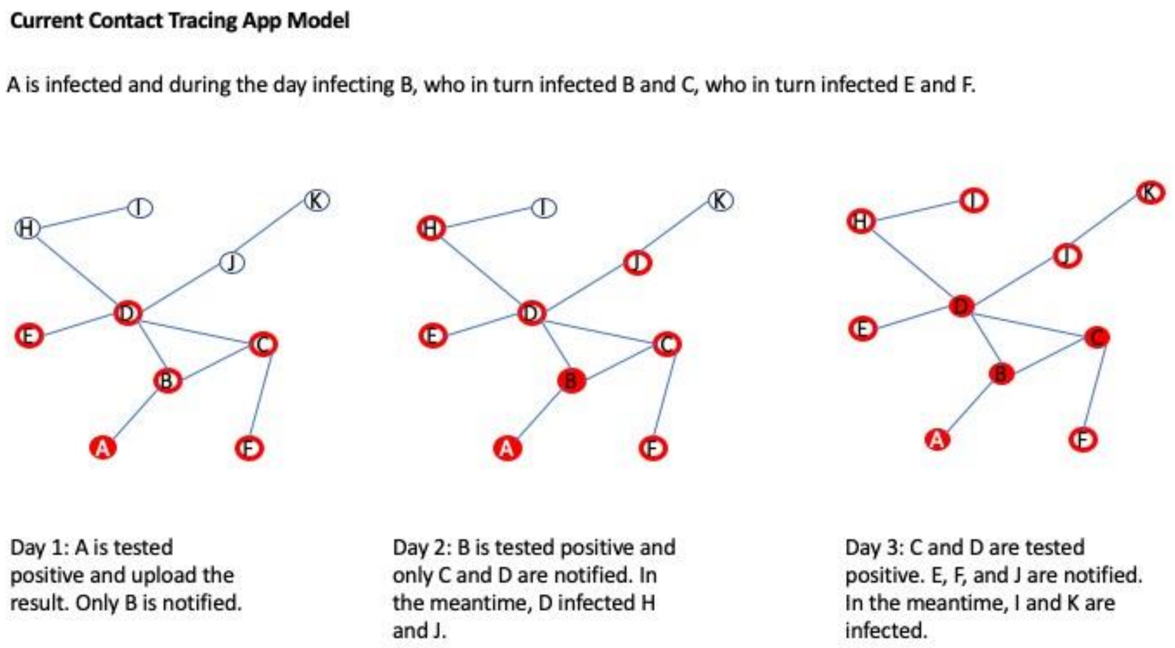
In what follows, we provide a high-level summary of our approach and a comparison with other proximity-based approaches. We then offer our approach to privacy protection, and potential risks that our approach faces and how we address them.

2. Motivation

2.1. Proximity-based Contact Tracing

Currently, most privacy-preserving contact tracing apps use a decentralized proximity-list generated and stored in the user’s smartphone. This approach *enables each phone to locally calculate whether the smartphone owner is at risk of having been exposed to a virus because they were in close proximity to an infected individual.* These apps digitize the traditional manual contact tracing, and claim that an “interaction graph” is not needed for analyzing the spread of COVID-19. These apps work similarly to traditional contact tracing in two ways. They focus on sending warnings to individual users and potentially report to a central authority based on direct interaction of individuals with confirmed infected patients.

This approach is only partially effective in quickly and efficiently identifying individuals at high risk of infection, particularly with highly viral epidemic situations, where many carriers of a virus are asymptomatic. The decentralized proximity lists at best offers a retroactive containment solution. In an illustrative example shown in Figure 1, even with a contact tracing app alerting users to potential exposure and with the availability of testing within the same day, we see the entire community is infected by day 3.



Current Contact Tracing App Model

A is infected and during the day infecting B, who in turn infected B and C, who in turn infected E and F.

Day 1: A is tested positive and upload the result. Only B is notified.

Day 2: B is tested positive and only C and D are notified. In the meantime, D infected H and J.

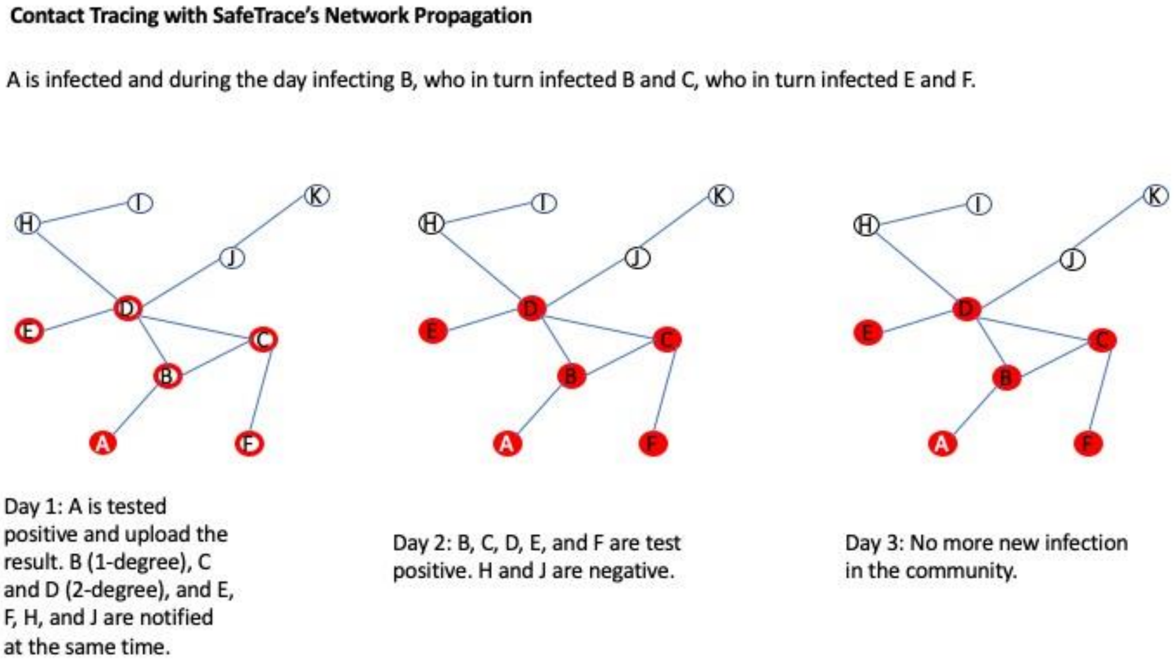
Day 3: C and D are tested positive. E, F, and J are notified. In the meantime, I and K are infected.

Figure 1. An Illustrative Example of Contact Tracing with Proximity-based Approach

2.2. Contact Tracing using an Interaction Graph

As discussed, existing contact tracing apps claim that an interaction graph is not needed for analyzing the spread of Covid-19. We believe that using an interaction graph for contact tracing is essential for timely and effective containment. In a hyperlocal interaction graph, one can capture direct and indirect interactions among users. This is essential as many users carry the virus asymptomatically from an earlier exposure to virus carriers.

It is also important that the system is able to pick up early warning signals based on the combination of interactions with others and symptoms. If a user has been exposed directly or indirectly to infected users, and the user is reporting symptoms, the system must be able to update the potential risk that the user is posing to others, even before the user is definitely tested positive. Figure 2 shows the same illustrative example with a contact tracing solution using an interaction graph. In this example, a virus flare-up is contained by day 2.



Contact Tracing with SafeTrace's Network Propagation

A is infected and during the day infecting B, who in turn infected B and C, who in turn infected E and F.

Day 1: A is tested positive and upload the result. B (1-degree), C and D (2-degree), and E, F, H, and J are notified at the same time.

Day 2: B, C, D, E, and F are test positive. H and J are negative.

Day 3: No more new infection in the community.

Figure 2. An Illustrative Example of Contact Tracing with an Interaction Graph

In summary, we need a solution that offers faster identification through a combination of (a) the consideration of both direct and indirect encounters; and (b) the ability to update the potential risk of a user to others based on their symptoms and their encounter history. A community-level interaction graph can be used to perform network propagations in order to calculate the risk that each individual node faces.

However, despite the obvious advantage of an interaction graph-based contact tracing approach, such an approach requires an aggregation of symptoms and encounter history in order to build an interaction graph at the community level. Protecting privacy with such an approach poses a huge challenge. This is precisely why most privacy preserving contact tracing apps avoid this approach, resorting on a less-effective proximity-based approach.

We offer an alternative to the current situation by considering a different underlying data architecture that is designed to protect the user's privacy through the separation of the real user and the virtual representation of the user. Our solution enables us to provide a community-level interaction graph to users while preserving their privacy.

Our approach is interoperable with other apps using proximity-based approaches, as long as these apps adhere to cross-platform standards (see Section 6 for a detailed discussion). ShareTrace therefore offers the same basic contract tracing function that other contact tracing apps provide, and more. The interoperability of ShareTrace allows its users (in-network users) share direct encounter information and the possible positive test results with the users of other apps (out-of-network users). In addition, ShareTrace users gain additional personalized risk assessments based on their symptoms and existing conditions.

In the future implementation, ShareTrace will offer in-app opt-in options for users to share their risks with government agencies, healthcare providers, their employer, and research organizations. Using HAT Microserver, users can legally and functionally control how their data is shared with these organizations.

3. Our Approach: Privacy by Architectural Design

We implement a smart contact tracing solution using a hyperlocal interaction graph while preserving privacy of users. We do this by using Personal Data Accounts (PDAs) to create anonymized digital twins of individual users. PDAs are implemented using HAT Microserver architecture. A schematic architecture of our design is also shown in Figure 3.

A central element to our approach to privacy by architectural design is Personal Data Accounts. A PDA is different from a cloud-based file storage (such as Google Drive or Dropbox) or a AWS/Firebase backend as it decentralizes user's data storage and computation to the "edge". It is powered by a new technology called the HAT Microserver (the HAT). The HAT is a personal data server where the user is the legal and functional owner of the data and confers data rights to users. The HAT Microserver sits within the HATDeX platform operated by Dataswift, and the platform enables data storage, computation, and contracting to different applications, ShareTrace being one of them. A PDA is, therefore, a cloud-based technology for data rights, mobility and control for individuals, making it fully interoperable with all other applications

Individual users can also give websites and applications permission to read and write data into their own specific app folders in the PDA. PDAs are usually issued to individuals by apps and websites when they register with the app/website, replacing the need for websites and apps to have centralized data stores where they hold users' personal and private information. PDAs also enable individuals to install pre-trained tools (supplied by organisations or data scientists) to generate "edge" analytics and private AI insights where the output data is returned back into the PDA and websites and apps can ask their users to share those insights with their websites and applications.

Using PDAs, we create anonymized digital twins. In our architecture, each user is represented as a physical and virtual identity. The virtual identity of each user is their own PDA, and the virtual identity of the user provides anonymity to the user. **A user's PDA acts as a proxy between the real identity of the user and the node of an interaction graph on the ShareTrace server.** Since the user has full control of her PDA, the user's PDA behaves as a trusted entity on behalf of the user on the ShareTrace server.

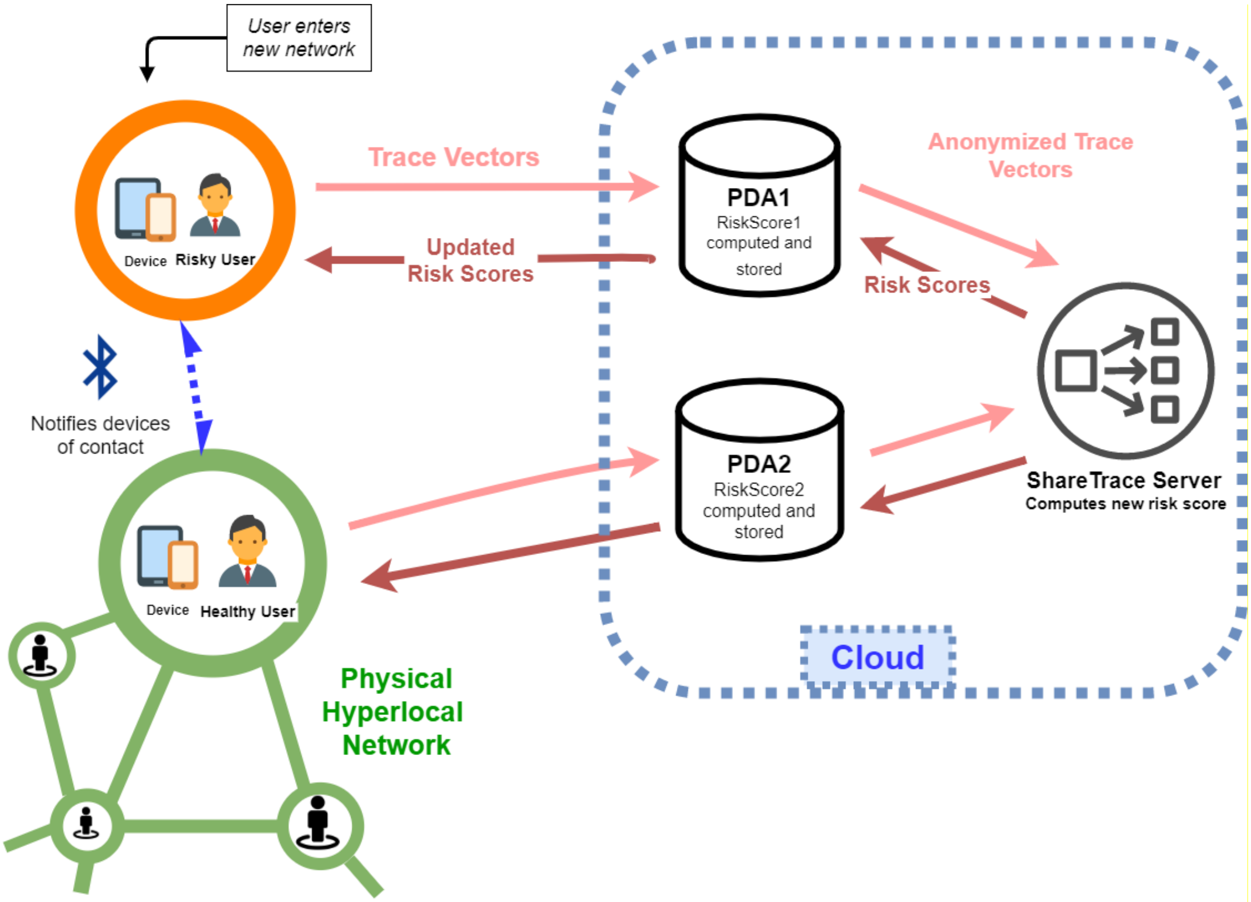


Figure 3. A schematic architecture of ShareTrace with Digital Twins and PDAs

When a user registers, the system generates a randomized unique ID (UID - anonymous user identity) and registers its UID to ShareTrace server. Thus, the ShareTrace server knows the user only through a UID which is randomized. UID, email, and any identifying information is all kept in the PDA. Thus, a UID is used for the user's PDA to communicate with the ShareTrace server and it will be updated periodically.

Each user also has a Bluetooth ID, which is the ID that the user transmits to other devices during a contact. This has to be updated regularly to avoid tracing a user based on their Bluetooth ID (i.e., to prevent location tracking via Bluetooth IDs).

In the first implementation, each user keeps two vectors in their PDA: (i) a vector for their symptoms and diagnosis; and (ii) a vector keeping their contact history (collected using Bluetooth). Each user also keeps her GPS history in her PDA. We collect GPS information from the users in a differentially-private way to provide some privacy guarantees for the users in the same way that Apple and Google collect usage data from users' phones and browsers. We also store user-reported symptoms and user-reported existing conditions in the PDA. Verified results of COVID-19 test and antibody test will also be stored in PDAs.

In a future implementation, each digital twin can include machine-generated biomarkers of symptoms such as temperature, coughing, heartbeat rate, SpO_2, etc. Users can also voluntarily share their GPS history with the ShareTrace server (after anonymization and obfuscation) in the future.

Periodically, each user's PDA updates the stored information to the ShareTrace server, but without revealing any identifying information about the user. Collected data is processed at the ShareTrace server to (i) update the risk factor of each user, and (ii) generate models calculating the spread of a virus (or maybe to identify the risky areas or the ones that are under risk in the future).

Through our approach to privacy by architectural design using PDAs, we go beyond traditional contact tracing. We follow symptoms and dynamically update personal risks by combining dynamically changing interaction patterns (direct and indirect) and changing symptoms of individual users. Each user's risk is updated when there are changes in (i) their own symptoms, (ii) symptoms of other users in the networks, (iii) diagnosis of other users (not necessarily direct contacts), and (iv) changes in the interaction network (not just direct contacts of the users).

In a future implementation, we can provide an option to combine the accurate existing health condition of the user by integrating their electronic health records into the user's own PDA. This will allow us to provide personalized, dynamic recommendations to the user.

4. Main functions of the App

Assigning keys and IDs to the users (e.g., cryptographic keys, UID, Bluetooth ID)

Each user will be assigned with cryptographic keys, UID (an anonymous identity between user's PDA and ShareTrace server), Bluetooth IDs, and functions (to derive new keys and IDs).

Access control

Make sure that the user only registers to the ShareTrace server once (anonymously). Without an access control mechanism, potentially, a user can register millions of times and feed the system with fake information.

We use blind signatures for access control. Note that using blind signatures, even if the signature authority and ShareTrace server collude, they cannot deanonymize the user

Managing and updating Bluetooth IDs

Each user i creates a daily key (K_i^d) for each day d and using this key, the user generates some number of daily Bluetooth IDs using a one-way hash function.

At any given time, user keeps its keys for the next “14” days.

When the user’s PDA sends information to the ShareTrace server it also sends the corresponding daily key. Using this key, the ShareTrace server can reconstruct all Bluetooth IDs of the user (that are used on that day).

Note that unlike similar apps, we do not let the server derive other keys of the user from K_i^d. This is to provide unlinkability between interaction graphs that are formed each day.

Updating the local contact vector of a user

The app logs close proximity between two users, exchanging their Bluetooth IDs and updating their local contact vectors.

User i provides to user j: Its Bluetooth ID for day d and also randomly generated Bluetooth IDs for the next 13 days: $[B_{i,j}^d, B_{i,j}^{d+1}, B_{i,j}^{d+13}]$ (cost of this is more data exchange during interaction).

Users keep the day (d) on which the interaction happened on their contact vector.

User j updates its local contact vector everyday with the corresponding Bluetooth ID of user i.

Local contact vectors will be kept in users’ PDAs and they will be updated after each contact.

Collecting GPS information from a user

The app provides users with the possibility to voluntarily share GPS data with epidemiologists and research groups to enable these groups to further analyze the spread of the virus (using location, demographics, etc.)

Updating the symptom and diagnosis information of the user

Periodically (once or multiple times a day), users update their symptoms to the app. In the first implementation, we will use 10 main symptoms identified by the NHS and we will ask the user to enter them in binary format (yes or no).

We may also ask the user about the progress of the symptoms (for the ones the user responded as “yes”). For each “yes” answer, we can let the user enter “better”, “worse”, or “same” compared to the last update.

Once the symptom vector is updated by the user, it will be stored in the user's PDA. Then, it will be uploaded to the ShareTrace server.

Storage of collected data in user’s PDA

Both the local contact vector and symptom vector of the user will be stored at the user's PDA and it will be updated over time (as the user has new interactions or as the user updates their symptoms).

Sharing symptoms, diagnosis, and contact information with ShareTrace server

Periodically (a few times a day), each user's PDA will share the local contact and symptom vectors with the ShareTrace server.

User i's PDA sends to the ShareTrace server on day d: $[S_i^d, C_i^d]$

S_i^d is the risk score of user i computed based on their symptoms on day d

C_i^d is the local contact vector of the user including the user's contacts during the last 14 days.

$\underline{C_i^d}$ includes pairs in the form $(B_{i,j}^d, t)$, where the first entry is the Bluetooth ID of user j for day d and the second entry is the time (e.g., day) on which i and j had a contact.

Data processing at the ShareTrace server

ShareTrace server updates the risk scores of users by (i) creating the interaction graph and (ii) applying the graph propagation algorithm (see the next section for the details of the algorithm) on the constructed graph.

ShareTrace server does not store data permanently. It deletes the collected data and parameters after each processing. We require this to minimize the risk of linkability between two interaction graphs that are generated on two different days (and hence to minimize the risk of tracing a user).

Sending the new risk score to the user's PDA

ShareTrace server pushes the updated risk score of each user to the user's PDA.

Or each user's PDA fetches the scores from the server periodically (by following a schedule).

Illustrating the risk score to the user

User receives the updated score from their PDA and we illustrate the risk to the user.

Roaming (from a local network to a remote network)

Each network will be hyperlocal, such as ZIP, or maximum county. When a user travels a long distance, ShareTrace app will register to the local (geographically speaking) network on the server.

Integration with other PDAs

Potentially, user’s other PDAs can be integrated with the ShareTrace app (locally at the user’s PDA). For instance, user’s electronic health records or demographics (in user’s health folder of his PDA) can be integrated with the user’s risk score to provide a more accurate and personalized risk score.

After warning

The user will decide what to do after getting the warning. In future implementations we may provide advice based on the score.

5. Hyperlocal Interaction Network using Label Propagation Algorithm

We generate an encounter graph based on each user’s contact (Fig. 4). A user reports COVID19 status (e.g., label information), then the label information will be propagated through the encounter network. The higher value indicates the higher risk of exposure to COVID-19.

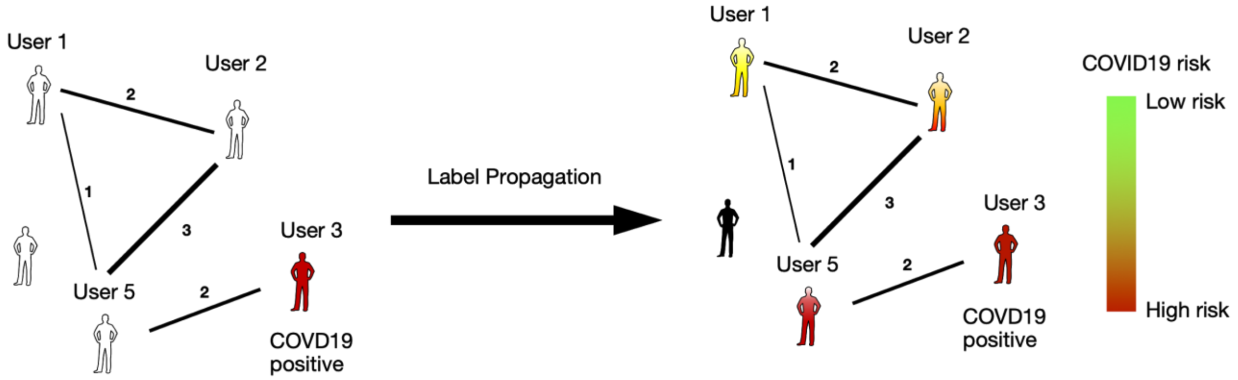


Figure 4. An illustration of Graph-based label propagation algorithm

5.1 Encounter graph

Encounter graphs are generated based on each user’s contact information. A node represents a user, and an edge represents an encounter between two users. The edge can be weighted based on a number of encounters at different time points. Initial label information (e.g., COVID-19 status or symptoms) will be assigned by the user. The encounter graph, along with label information, will be collected and generated multiple times per day.

5.2. Label propagation

In the first implementation, label propagation will be performed based on the algorithm proposed in Zhou 2020. We will develop and apply an algorithm taking into account real-time updates of label and encounter information.

6. Interoperability

Currently, the dominant design of privacy-preserving contact tracing apps locally generate frequently-changing IDs and broadcast them via Bluetooth. Other smartphones observe these IDs and store them together with the duration and a coarse indication of time. Then, when tested positive, users instruct their phones to upload to the backend a compact representation of their IDs for the infectious period. The backend stores these compact representations. Other smartphones periodically query the backend for this information and reconstruct the corresponding IDs of infected patients locally. Therefore, the main functionality of other existing contact tracing apps is to warn the users about their previous contacts with an already infected individual.

Since multiple contact tracing apps are being developed, it is critical to ensure interoperability among these apps. Without interoperability, individual contact tracing apps will never be able to provide enough coverage of the population, which is essential for those apps to be effective. Of course, an alternative approach is for a central government or public health authority to make such an app mandatory as being done in other countries like South Korea, Singapore and India. Such a centralized approach is unlikely to be effective in pluralistic and democratic societies. Therefore, it is essential for these apps to provide interoperability.

In order to offer interoperability, we need two cross-platform standards. First, all contact tracing apps must use the same Bluetooth ID framework to generate contact vectors. WIth it, users of different contact tracing apps can still generate the local contact list. Currently, Apple and Google are developing a cross-platform framework to provide such a standard. Second, all contact tracing apps must agree to exchange keys of the Bluetooth IDs of infected users to each other. Such exchange of Bluetooth IDs can be done with minimum threats via API calls between servers. At the moment, there are no such standard protocols being developed.

ShareTrace is designed to provide such interoperability with other contact tracing apps. When a cross-platform Bluetooth ID broadcasting framework emerges, ShareTrace users (in-network users) can collect Bluetooth IDs of other app users (out-of-network users), and vice versa. Unlike other apps, ShareTrace will store the list of other users’ Bluetooth ID in their HAT, not on their local device. ShareTrace server is designed so that it can exchange keys of the Bluetooth IDs of confirmed test positive users. In the future implementation, ShareTrace will provide APIs for out-of-network users.

In the near term, in the absence of cross-platform standards, ShareTrace can utilize the warnings provided by other apps (from out-of-network users) for our in-network users (assuming that out in-network user also has the other apps on their device). When an in-network user receives a warning from another app, it means that the user had contact with a diagnosed individual (that uses the other app). This is something we cannot detect since here, the diagnosed user does not use our app. But, using the received warning, we update the risk of our in-network user accordingly and propagate that updated risk to all of our other in-network users.

7. Roaming

Since ShareTrace is based on hyperlocal interaction networks, each user belongs to a home network. Therefore, we must consider the situation when users travel outside of their own hypernetwork. To do this, we introduce “roaming” capability.

ShareTrace allows users to “roam” to different locations, and immediately receive appropriate guidance even when traveling. Initially, a user registers to one or more hyperlocal networks and then it is possible to change this choice (e.gl, when the user travels). During the day, when the user uploads its contact and symptom vectors (along with its diagnosis), it uploads this to all hyperlocal networks that it is actively registered. Most users will register to only one network at a time, but considering users with long commutes, it is possible that a user may be actively registered to multiple hyperlocal networks at a time.

Thus it is possible that a user may have multiple risk scores computed over different hyperlocal networks. These risk scores will be aggregated inside the user's PDA and the user will report its aggregated risk score when it sends its data to its hyperlocal networks for the next interaction. Note that when a user registers to a new hyperlocal network for the first time, it only carries its risk score from its previous hyperlocal network.

8. Design parameters

Unique master key of the user (M_i): using a random number generator

Size and generation of UID: UID of the user is generated using user’s master key, an identifier of the user, and a counter:

$UID_i = f(M_i, ID_i, c_i)$, where f is a one way function, ID_i can be the email of the user, and c_i is the counter (it is incremented by one whenever the UID is updated)

Size of user’s daily key and how to generate it locally: The daily key is the hash of master key concatenated with the date and the time:

$$K_i^d = H(M_i|month|day|year|hour|minute|second)$$

H is a cryptographic hash function (MD5 or SHA)

Size of Bluetooth ID: equal to the size of the Bluetooth beacon

How often to update the Bluetooth ID and which function to use for the update:

Bluetooth ID of a user i at a given epoch (epoch ID) on a given day d is computed as:

$B_i^d(epochID) = f(K_i^d, epochID)$

Number of epoch IDs per day is a design parameter and f is a one way function

Identification of an interaction as a contact: (i) maximum contact distance (dist) and (ii) minimum contact duration (dur).

How often does the ShareTrace server get information from user's PDAs: Once or twice a day

How often (how many times a day) does the ShareTrace server update the risks of the users and in which schedule: Once or twice a day

Illustration of the risk to the users: Color coded risk levels

RSA blind signature parameters: Standard RSA parameters recommended by NIST (2048 bits key size as of today)

9. Privacy analysis

We show how we provide privacy against:

(i) a malicious user (trying to infer location information, symptoms, or diagnosis belonging to another target user)

User i trying to learn the symptoms or diagnosis of a user j: The symptom vector never leaves the user's PDA, only the risk score of the user is transferred to the ShareTrace server. Thus, the only way for user i to learn this information (or infer something about the risk score of user j) is to arrange its local contact vector in such a way that it only includes user j during a day. By doing so and analyzing the change in its overall risk score, user i may infer some information about the risk score of user j.

But, this attack is not scalable, and the same problem (modification of contact histories) exists in all other similar contact tracing apps. One way to mitigate this is to use digital signatures. Furthermore, in our scheme, since the contacts are symmetric, such an attack (i.e., modification of contact vectors) will be detected at the ShareTrace server and such a user will not be included in the risk calculation.

(ii) ShareTrace server (trying to deanonymize collected data)

- Deanonymization of the interaction graph: The graph includes no location information about the users. Also, it is computationally infeasible to deanonymize large scale graph data, especially when there is no auxiliary graph to help.

- Linking different interaction graphs to trace the users (avoided using new UIDs each time)

(iii) ShareTrace server colluding with a malicious user (trying to infer location information, symptoms, or diagnosis belonging to another target user)

- If a user i colludes with the ShareTrace server, they can infer the diagnosis or risk score of a user j as follows:

User i simultaneously monitors its contact with user j and the update in its local contact vector. By doing so, user i can learn the Bluetooth ID of user j. Then, colluding with the ShareTrace server, user i sends the Bluetooth ID of user j to the server to learn its diagnosis and risk score. Since the server knows the Bluetooth ID, diagnosis, and risk score of user j, it provides this information to user i. Note that this is a common problem of all existing contact tracing apps (not unique for our solution). In traditional contact tracing apps, it is possible to mitigate this using private set intersection (Demirag and Ayday 2020). However, as discussed, our functionality is much richer than existing contact tracing apps.

Thus, compared to existing contact tracing apps, the above risk is the same in our scheme. Thus, the potential privacy leak of the proposed scheme is equivalent to existing contact tracing apps in terms of deanonymization (inferring the diagnosis or risk of a target user).

10. Scalability

The complexity of operations we will conduct at the ShareTrace server a few times a day:

- Generation of the interaction graph: This is done at the ShareTrace server and this has a low complexity.

- Running the graph propagation algorithm: $O(n^2)$ where n is a number of users. There are variations of label propagation algorithms which can scale to millions and billions nodes in the graph.

Complexity at the user’s side: (i) sending and receiving small amount of data, (ii) updating local contact and symptom vectors, (iii) calculating the symptom risk