SafeTrace: Simple Infection Tracking Model

Justin Lewis

safetraceapi.org

April 4 2020

1 Model Assumptions

Assume we have n users $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. Each user owns written data d_i. Data d_i is divided into public data $d_{(P,i)}$ and private data $d_{(p,i)}$. Let $\mathcal{D}_P$ be the set of all public user data, $\mathcal{D}_p$ the set of all private user data, and $\mathcal{D}$ is the union $\mathcal{D}_P \cup \mathcal{D}_p$. Each user also possesses identifying information I_i, which includes private information which is not considered in written form for this analysis. I_i could include someone's name, home address, physical identity, etc. (we may need to define this more narrowly). Assume further that we have m servers $\mathcal{S} = S_1, S_2, ..., S_m$, where each server S_i is operated by a distinct non-colluding entity E_i. We will assume that each entity E_i is *honest, but curious*. This means that each E_i follows the protocol prescribed to it; however, each will try to learn as much as possible about private user data $\mathcal{D}_p$ in running the protocol. Please see Figure 1 for a schematic of this abstraction.

2 Private Infection Tracking Problem

Now, under the assumptions of Section 1, users $\mathcal{U}$ wish to track the spread of a disease at the level of individuals by interacting with an *Infection Tracking System* (ITS). This ITS will have the following components:

a. A standardized *user data structure* and *data collection mechanism* $\mathcal{M}_t(u_i)$. This mechanism generates data for user u_i at time t.

b. *Infection tracking program* $P(\mathcal{D}, u_i)$. This program is dependent on all user data $\mathcal{D}$ and input user u_i. A query to the program updates u_i's estimated probability of being infected, $\hat{p}_i$.

c. Decentralized group of servers $\mathcal{S}$ (as defined previously) with the following components:

   (i.) standardized database structure for collectively storing $\mathcal{D}$ as it is updated.

(ii.) *Computation protocol* Π by which the servers collectively compute $P(\cdot)$.

(iii.) *Communication protocol* π by which users interact with servers $\mathcal{S}$.

Given all this, our problem is to design an ITS which is *sufficient*. That is, an ITS which satisfies the following criteria (each point clarified in following subsections):

a. Computation protocol Π is *private*.

b. Tracking program $P(\cdot)$ is *sound*.

c. ITS is *scalable*

d. ITS guarantees some reasonable level of *identification privacy*.

e. (Bonus:) choice of program $P(\cdot)$ guarantees some level of *differential privacy*

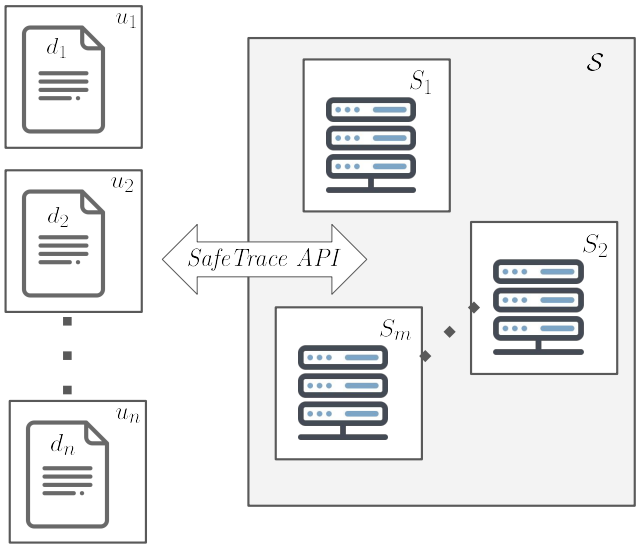

Figure 1: Diagram demonstrating the relationship between users, their data, our *private* ITS (*SafeTrace API*), and a set of decentralized servers.

2.a *Private* Computation Protocol

Here we define the notion of a *private* decentralized protocol Π which is used to collectively run $P(\cdot)$ on servers $\mathcal{S}$. Such a protocol is *private* if: (i) in running

the protocol, servers $\mathcal{S}$ gain *zero-knowledge* of private user data $\mathcal{D}_p$. (ii) From the perspective of users, the protocol is *secure*. That is, no user u_i obtains any more knowledge about private data $\{d_{(p,j)} \; \forall \; j \neq i\}$ than could be obtained if all computation were done through a trusted third party .

2.b *Sound* Tracking Program

Here we define the notion of a tracking program $P(\cdot)$ which is *sound*. This property is satisfied if probability estimate $\hat{p}_i$, as informed by running $P(\mathcal{D})$, closely matches true infection probability $p_i \; \forall \; i$. (could define this more clearly, but maybe not so important).

2.c ITS Scalability

The exact notion of scalability is not something we will likely analyze theoretically; however, it's something we will hope to demonstrate empirically. Let $\mathbb{E}[\Delta t_i]$ be the expected time between infection probability updates for user u_i. In our context, let's say that an ITS is scalable if: (i) $|\mathcal{U}| \sim$ millions/billions (ii) $\mathbb{E}[\Delta t_i] \sim$ hours/one day $\; \forall \; i$. Until we can test an ITS empirically, one clear measure of scaleability in the MPC protcol model would be the complexity of the boolean circuit to be run.

2.d Identification Privacy

Here, an ITS preserves *identification privacy* if a user u_i's identifying information I_i is not leaked (or minimally leaked) in their interaction with the system. This criterion is left loosely defined. This is natural as there are many side channels by which a computational server $S_k \in \mathcal{S}$ or another user could learn about I_i. Note the distinction between *identification privacy* and other notions of privacy listed: *identification privacy* is privacy w.r.t non-written information I_i while all other privacy notions defined previously are defined w.r.t written data $\mathcal{D}_p$.

2.e Differential Privacy

Loosely speaking, our program $P(\mathcal{D}, u_i)$ is deferentially private if the output of $P(\mathcal{D}, u_i)$ gives user u_i little information about the private data of another user. As an example, consider the case that over the course of a week user u_i only interacts with user u_j who is infected. In giving user u_i information about their probability of infection $\hat{p}_i$ via program $P(\mathcal{D}, u_i)$, u_i can potentially infer knowledge of $\hat{p}_j$. This data $\hat{p}_j$ is assumed to be private for u_j. Thus, we hope to design the program $P(\cdot)$ to make a trade off between: (i) knowledge u_i gains of $\hat{p}_i$ in running $P(\cdot)$ (ii) knowledge u_i gains of other users' private data $\{d_{p,j} \; \forall \; j \neq i\}$ in running $P(\cdot)$. These two things seem to be fundamentally at odds, thus a tradeoff is necessary.

3 Simple Model: BC-ITS

Here, we outline a very simple ITS using bluetooth as a sensing modality. Let this model be denoted as BC-ITS, i.e. "bluetooth collision"-ITS. The components of the system are spelled out in Section 3.a and then we analyze this system in Section 3.b.

3.a BC-ITS Components

3.a.1 User Data Structure

Let each user u_i possess a unique public identifier U_i. Let C_i be a public piece of contact information (which is only shared with servers $\mathcal{S}$). Let $(T_i, \tilde{T}_i)$ be an estimated time range during which u_i was/is believed to be infectious. Finally, let $L_i = \{\cdots\}$ be a list of location/interaction data. Given this, we have that $d_{(p,i)} = \{C_i, \hat{p}_i, (T_i, \tilde{T}_i), L_i\}$ and $d_{(P,i)} = \{U_i\}$.

3.a.2 Data Collection Mechanism

A user u_A estimates the distance $\Delta(u_A, u_B)$ to another user u_B via bluetooth signal strength. If the estimate $\hat{\Delta}(u_A, u_B) \leq \delta$ for a time period $\Delta t \geq \tau$ (given appropriate constants δ, τ), then the users log this (δ, τ)-encounter. That is, user u_A appends the tuple (U_B, t_0) to their location data list L_A while u_B appends (U_A, t_0) to L_B. Note that t_0 denotes the timestamp when users A and B first crossed the distance threshold δ.

3.a.3 Infection Tracking Program

Here we define how a query to $P(\mathcal{D}, u_i)$ updates $\hat{p}_i$. Consider the following simple rule: if u_i has ever had a (δ, τ)-encounter with someone who was infectious at that time, then u_i is considered to be infected. To update $\hat{p}_i$ for user u_i based on this rule, $P(\mathcal{D}, u_i)$ must loop over each interaction (U_k, t_k) within interaction list L_i and do a simple *collision* check. A *collision* occurs between u_i and u_k if $\hat{p}_k == 1$ and $t_k \in (T_k, \tilde{T}_k)$. If one or more of these *collision* checks is True, then $\hat{p}_i$ should be set to 1. Otherwise, it remains 0.

3.a.4 Storage and Computation Protocols

The general framework for storage and computation of the ITS will be MPC-based using Shamir's Secret Sharing as a primitive. See main whitepaper for more details. Here, we will describe storage and computation at a high level.

i. MPC Storage:
To interact with the ITS, a user u_i will take each element of their private data $d_{(p,i)}$, break this element into m cryptographic shares, and send a single share to each of the m servers. Let each server $S_k \in \mathcal{S}$ store these shares onto its database D_k using the same data structure from Section

3.a.1. So instead of holding $d_{(p,i)}$ for user u_i, database D_k holds the cryptographic share $s(d_{(p,i)})[k]$ (see Appendix Section 5.a for more details). Note that the size of interaction list L_i must be of a fixed size to guarantee privacy. For simplicity, let's consider each database D_k to be a dictionary with (U_i, $s(d_{(p,i)})[k]$) as a (key, value) pair.

NOTE: the size of list L_i for each user u_i must be of a fixed length to guarantee the highest level of data privacy. Otherwise, if this list was of variable length, a server could gain some information of $\hat{p}_i$ (i.e. the more interactions a person has, the higher their risk of being sick). Therefore, if u_i has only a few interactions, they will have to pad the rest of the list with "dummy" interactions. These "dummy" interactions could be indicated to the computational circuit by an additional bit, say $b_(i, j)$. Where $b_(i, j) = 1$ if the jth interaction of L_i should be ignored and 0 otherwise. Furthermore the user ID and time of the jth interaction can be chosen arbitrarily if it is of type "dummy".

ii. MPC Computation:
For this introduction, we will omit the majority of the MPC protocol details. Instead, we will consider the boolean circuit that will be run and how data will be accessed.

To update $\hat{p}_i$ in a way which is fully *private*, note that each server only owns a share of each interaction of L_i. Thus, every server has *zero-knowledge* of the user IDs of user u_i's logged interactions L_i, that is, no knowledge of $\tilde{\mathcal{U}} = \{U_j \ \forall \ j \mid u_j \in \text{users}(L_i)\}$. Because of this, in order to check for a *collision* between u_i and an unknown user $u_k \in \tilde{\mathcal{U}}$, the MPC implementation of $P(\cdot)$ must consider all user entries in the database. More specifically, the computational circuit must take the probability estimates $\{\hat{p}_j \ \forall \ j\}$, user IDs $\{U_j \ \forall \ j\}$, and estimated infection timeframes $\{(T_j, \tilde{T}_j) \ \forall \ j\}$ as input. For each user ID U_j, the computational circuit must compute a *collision* check bit which is only passed on if U_j matches U_k. This pairwise *collision* check is repeated for every interaction in L_i and the corresponding output bits are OR'd to form the new estimate $\hat{p}'_i$. This may have been confusing, so for illustration this naive computational circuit be seen in Figure ??. Note that location obscurity of the relevant data seems to be the most costly factor in the full computation. In Section 4 , we will consider modifications to the above construction which strike a tradeoff between privacy and computational scalability. It seems this tradeoff may be essential for the BC-ITS to scale.

3.a.5 User-Server Communication protocol

Once the MPC computation is completed by the servers $\mathcal{S}$, each server S_k sends a share of the new infection probability estimate $\hat{p}'_i$ back to user u_i. Then, u_i will reconstruct $\hat{p}'_i$ locally. Then, u_i must update $(T_i, \tilde{T}_i)$. After updating, u_i

must eventually share this private data update back to the servers $\mathcal{S}$ by further secret sharing. NOTE: in this scheme, users must periodically send an update to their data stored in servers $\mathcal{S}$ in order to maintain their privacy. Otherwise, the servers can potentially infer some information of $\hat{p}_i$. E.g. if someone only updates their data when they have a (δ, τ)-encounter, then clearly information of $|L_i|$ is leaked. This can be remedied by having users choose random time intervals between updates or updating on a periodic schedule. This is another example of how privacy does not come for free, in this case the system pays in communication overhead.

A question which has yet to be answered: how does each pair of server S_k and user u_i establish a communication channel $CC(i,k)$ and do so in a way which does not leak identifying information I_i? (i) each channel $CC(i,k)$ must be encrypted with a unique set of keys (ii) the channel must be akin to onion routing which aims to prevent the communication endpoint to gain information of the senders geophysical location. The information necessary for each server S_k to use this channel to u_i is stored inside of u_i's public contact information $C_i[k]$ which is shared with the kth server upon first connection.

3.b BC-ITS Analysis

In this section, we will consider the *sufficiency* of the BC-ITS construction outlined in Section 3.a. Recall that an ITS is *sufficient* if it satisfies the criteria listed in Section 2.

3.b.1 *Private* Computation Protocol

Given the assumptions of Section 1 are true, the BC-ITS construction satisfies the criterion of being *private*. Secret sharing plus using MPC for the computation ensures this.

3.b.2 *Sound* Tracking Program

The level of sophistication that the BC-ITS affords is similar to that of manual contact tracing; however, it operates in a way which is automated. One drawback this framework shares with manual contact tracing is that a user must be labeled as infected before any infection information can propagate to other users (confirmation by test or by obvious symptoms). Note also that the BC-ITS tracking program has coarse granularity in assessing interactions. The context of the interaction could be very useful information this model does not leverage (e.g. an interaction inside a closed space is much more significant).

3.b.3 Scalability

In considering the wide range of possible models for contact tracing, this initial BC-ITS construction is quite simple. The core computational components for

one update of $\hat{p}_i$ are: comparisons between small numbers (for timestamp component), a simple one bit equality check, and a for loop with $|L_i|$ iterations. If the problem of data location obscurity can be alleviated, this approach seems promising in terms of scaleability. It is clear that scaling to millions of users would be infeasible if $P(\cdot)$ has to touch every user entry in the MPC shared database. See Section 4 for ideas to improve the scalability of this construction.

3.b.4 Identification Privacy

By using a secure communication protocol π, and an anonymizing ID U_i, it seems that user u_i is not leaking any identifying information I_i in interacting with the BC-ITS. In this type of system, side cahnnels abound, so there may be something clever a malicious user could do to link U_i and I_i.

3.b.5 Differential Privacy

At this point, the BC-ITS as written does not address differential privacy directly. For example, a user u_i can still infer a lot of $\hat{p}_j$ if the two have a (δ, τ)-encounter and $|L_i|$ is small.

4 Modified Model: BC-ITS+

4.a MPC Computation Mods

In Section 3.a.4, it was noted that the most private way to implement an MPC-style computation for a *collision* check between two users (u_i, u_k) must consider every user in the database for input. This is clearly terrible for scaling purposes. There are a number of possible mods that can remedy this problem and each comes with various levels of reduced privacy.

4.a.1 User IDs "in the clear"

One plausible idea to consider is sharing user IDs as plaintext for each (δ, τ)-encounter. Normally, each ID is secret shared. Secret sharing prevents a server from knowing that there is a connection between two user IDs, say U_A and U_B. Because U_A and U_B are anonymized, allowing knowledge of this connection is completely acceptable *in theory*. Furthermore, the infection status of individuals remains private. The question is if the anonymity of user IDs is sufficient in practice. In general, there is a time varying graph $\mathcal{G}_t$ on $\{U_1, U_2, ..., U_n\}$ which describes all of the ID encounters through time. By sending user IDs "in the clear" this allows a server to know this graph exactly. By using this method, the MPC computation to update $\hat{p}_i$ only needs to operate on users within users(L_i) rather than all users $\mathcal{U}$. This complexity reduction is quite significant.

4.a.2 User IDs Partially "in the clear"

Another more plausible idea is to give partial knowledge of user IDs as plaintext for each (δ, τ)-encounter. For example, let's say that each ID U_i is assigned to u_i by the servers $\mathcal{S}$ in a way which is random and unique. Specifically, U_i is chosen by sampling uniformly at random from $\{0,1\}^s$ until a sample which was not previously assigned is found (so long as s is long enough, we can guarantee uniqueness for all users). Now, let's say that u_A and u_B have a (δ, τ)-encounter. Instead of secret sharing U_B to servers $\mathcal{S}$, u_A sends the first b bits of U_B as plaintext and the rest can be secret shared. The same split is done by u_B as well, but with U_A. In doing so, u_A and u_B only give partial information of their interaction. The servers now know that ID U_A interacted with an ID in set $S_1 = \{U_k \mid U_k[1:b] = U_B[1:b]\}$ and that ID U_B interacted with an ID in set $S_2 = \{U_k \mid U_k[1:b] = U_A[1:b]\}$. Note that the sizes of sets S_1 and S_2 must be large enough for a significant number of cross interactions to occur w.h.p. Now, if b is chosen appropriately, the number of users which need to be considered for MPC computation could potentially be a constant number which does not grow with the number of users n. This is a very desirable scaling property. The reason why a constant may be sufficient is that estimating $\mathcal{G}_t$ is quite difficult due to its sparsity. It seems that gaining any useful knowledge from $\mathcal{G}_t$ would require almost perfect knowledge of the interactions.

4.a.3 Spatiotemporal "in the clear"

This idea is similar in spirit to the above idea. Again, let's say that u_A and u_B have a (δ, τ)-encounter. Instead of giving partial information of U_B, u_A would share some plaintext information of the time and location of the encounter. This information will be provided at a granularity which is deemed acceptable to the user. More formally, let's say there is a partition P_L of (lat, lon) global location coordinates and a partition P_t in time. Now, the user only shares which element of each partition the true (lat, lon) and timestamp triple fall into. By changing the granularity shared of the encounter, users tradeoff spatiotemporal privacy with $\mathcal{G}_t$ interaction link privacy. In this tradeoff, it should be noted that with more interaction link privacy comes a higher price in MPC computation cost. For example, if the spatiotemporal zone that is sent "in the clear" is too large, the number of users that also had a (δ, τ)-encounter in this zone is also large.

NOTE: The above mentioned partitions can be designed arbitrarily. For example, perhaps the location partition can be non-uniform to account for the density of urban areas.

5 Appendix

5.a Secret Sharing Notes