CS227 Harvard Spring 2020

Perfectly Private Contact Tracing

Jeremy Dohmann

1 Introduction

COVID-19 is very bad [citation needed], especially so because people can be asymptomatic vectors for it. One of the most reliable tools we have against community transmission is contact tracing and quarantining.

There have been a number of contact tracing apps proposed, and Apple and Google have both released protocols which are based on the Low cost decentralized proximity tracing implementation of the DP3T protocol. Here is their white paper and github.

Their system relies on individuals choosing a new random key every day and calculating a sequence of ephemeral messages from that random key. From there those messages are broadcast to people nearby. When they are diagnosed, the individual uploads all the daily keys they used, and other individuals on the network are then able to calculate all the ephemeral messages those keys would've generated and see if they match any they received.

I find this approach somewhat concerning because it can allow you to pinpoint the exact time and place you got infected if you (break the protocol) and store time/location stamps for the ephermal IDs you receive. This could then allow you to determine the diagnoses of the people you know if you know who you were with at the time/location you were infected.

I make an attempt at a more secure standard which prevents other participants from learning any new information about the identity of who exposed them. There are difficulties with this security standard that I am unable to circumvent. Namely, it is difficult to ensure that Bob gains no new information regarding the identity of who exposed him without Bob providing compromising information about himself to the network.

This trade off comes because it seems impossible to make Bob unable to reconstruct the time and location of his exposure without him sending the same unique message during every exchange - but doing so runs the risk of other people being able to know that Bob was exposed, which may be a security risk to him.

2 Task requirements

Any system which performs contact tracing must accomplish one main goal:

Functional criteria

1. Notify parties whenever they have interacted with an infected individual.

There are secondary design preferences I will outline next, but first and foremost, the system is only functional if it is capable of informing individuals that they have been in the presence of someone who was infectious.

I have identified two primary security concerns relating to any contact tracing system. They are motivated primarily by the privacy of peoples' interactions (both because of the danger of revealing the underlying social network as well as the danger of revealing people's locations by mistake), and the privacy of peoples' diagnoses.

Privacy criteria

1. We don't want interactions to be revealed to anyone but the two interacting parties. That is, no party Mallory (including the central organization running the server) should be able to know that Bob and Alice interacted.
2. We don't want to reveal unnecessary information about infected peoples' diagonses. In particular, if Bob is informed that he has been exposed, the proof that he was exposed shouldn't reveal to him (or anyone else) any more information about the identity of the individual who exposed him than what he already he had.

I will now formalize the problem of Perfectly Private Contact Tracing:

> Assume the existence of a network composed of a central server S as well as N pieces of hardware interacting on the network.
>
> A protocol is a contact tracing protocol, if for every sick individual Alice, we are able to prove to everybody who interacted with Alice in the last two weeks that they were exposed.
>
> A protocol is perfectly private if a.) if Bob gets notified that he was exposed then the probability that Bob determines the identity of the infected individual who exposed him is no greater than $1/|I|$ where $|I|$ is the number of people he interacted with over the past two weeks and b.) if Mallory is snooping on the network she can't determine the identities of any two pairs of individuals who interacted unless Mallory physically saw them interact c.) if Mallory is snooping on the network, she can't determine the times and locations that Bob interacted with

other third parties, d.) if Mallory is snooping on the network, she can't determine the identities of people who were exposed.

A protocol has integrity if a third party Mallory can't inform Bob that he was exposed unless Mallory interacted with him and Mallory got a postitive diagnosis from a medical professional.

3 System design

The system will consist of three principle functions: Interaction, Notification, and Integrity Assurance. The three functions will interact with one another at run time, but I will introduce them separately as they accomplish different aspects of the goal.

I will rely on the following cryptographic primitives, which I define briefly in the appendix.

Cryptographic primitives

1. Pseudorandom number generators
2. Integer factorization
3. Commitments
4. Digital signatures

It also assumes the existence of some hardware primitives

Hardware primitives

1. Networks capable of transmiting messages between users within a certain radius of each other (but not to people not within that radius)
2. Unique, unforgeable IDs that identify where a message came from

3.1 Interaction

Say Alice and Bob interact at time t, Alice will eventually get sick by time $t' = t +$ 2 weeks, but doesn't know it yet. When she does get sick, however, she wants to be able to broadcast to Bob that they interacted, without revealing to Bob when he got exposed, where he got exposed, and who exposed him. That is, Alice must later prove to Bob that they has an interaction without Bob knowing which particular interaction it was that they had.

The process is simple and relies on the difficulty of factoring the products of large primes.

Every party following the protocol has a unique large prime (with security parameter n) which serves as their unique key that they send during every interaction, call Alice's a and Bob's b. Additionally, all parties generate transient IDs for each unique interaction, call Alice's A_t and Bob's B_t. Note that a, b, A_t, B_t are all i.i.d. large primes. Assume that Alice and Bob only engage in at most one unique exchange for every T second period they are near each other.

When Alice and Bob are within some distance of each other, their phones will engage in a simple exchange: Alice sends a to Bob and Bob sends b. Alice and Bob store $A_t \cdot b$ and $B_t \cdot a$ to their interaction ledgers, respectively.

3.2 Notification

When Alice gets sick at time t' she will upload her ledger from the last two weeks to a public server, including entry $A_t \cdot b$. Alice can only upload if she reveals her identity and a trusted medical professional verifies her diagnosis and identity. The server will then broadcast her ledger over the network.

Periodically Bob will check the broadcasts for numbers he can factor, if he encounters a number he can factor then he knows he interacted with someone who was sick.

Theorem 1. *The probability that Bob's unique private key b can factor a number in the public ledger if he wasn't exposed is negligible. i.e., there is a negligible false positive rate.*

Proof. There are two possibilities, either Bob can factor k because some other user has the same unique private key as him, or because someone generated a transient ID A_t which happens to equal his b.

If there are N users on the network, there are potentially $N^2 \times 1,200,000/T$ unique transient IDs per ledger uploaded, and potentially N ledgers uploaded over the course of the outbreak. This means there are $O(N^3)$ possible unique transient IDs generated and $O(N)$ unique private keys. If we want to ensure that there is negligible probability any transient ID or unique private key is duplicated, we can use the birthday attack to choose our parameters to ensure a low probability of any of the $N^3 \times 1,200,000/T$ IDs are duplicated. □

Theorem 2. *If Bob broadcasts b, the probability that Bob correctly guesses which interaction resulted in his exposure is $1/|I|$ where $|I|$ is the number of interactions Bob used b for.*

Proof. Bob factors k, to reveal its factors S_t, b. Assuming the security of everyone's PRG, S_t is indistinguishable from a randomly generated prime. Therefore observing S_t tells him nothing about the identity of the person who generated it or the time it was generated. All he knows is that it was generated from one of the interactions at which he exchanged b during the last two weeks.

If there were $|I|$ interactions during which he exchanged b then he can guess who infected with probability $1/|I|$. □

Theorem 3. *No third party Mallory (including the public server) can determine that Alice and Bob interacted*

Proof. It is possible that Mallory knows Bob's unique private key b from some previous interaction with Bob. This means Mallory can factor k to find A_t. Since A_t was only ever used for one interaction, Mallory cannot determine the identity of the person who produced A_t.

Assume towards the sake of contradiction, the existence of some algorithm M which has access to any set of numbers $A_1, A_2, ...$ taken from Alice's PRG except A_t. Furthermore, assume that on input A $\{A_t, S\}$, where S is a truly randomly chosen prime, M can determine whether A was produced by Alice or produced by a truly random distribution.

Clearly, if such an M existed it would violate the security of the PRG, thus there couldn't exist such an M, and therefore Mallory can't use observations of A_t alone to determine that it was Alice that Bob interacted with.

□

Theorem 4. *No third party Mallory can determine the times or locations of when Bob interacted with individuals*

Proof. If Mallory knows b she can factor any certificates of Bob's exposure to reveal the ephemeral random factor of the certificate A_t, but as discussed in theorem 2, since it is indistinguishable from a random number, it contains no information regarding the time or location it was produced. □

3.3 Integrity Assurance

We noted in the previous section that this system is only confidential w.r.t. Alice's diagnosis if Bob uses the same b for every interaction. Thus, it is crucial that we ensure the integrity of the system by only broadcasting $A_t \cdot b$ if Bob has been following the protocol.

Integrity Assurance will rely on three elements: commitment, authentication, and verification.

The intuition is a) we would like Bob to have a certifiably unique key b that he chose when he joined the network, b) we would like Alice to be able to verify that the key b that she receives from Bob is the one he committed to at the beginning, c) if Bob violates the protocol by sending a b' that wasn't the b he committed to at the beginning then he doesn't get notified in the event that Alice is sick.

In order to do so, we will need the public server to certify the authenticity of Bob's unique key *without the server ever actually seeing the key*.

Integrity Assurance procedure

- **Commitment:** At the beginning, Bob generates his unique key b and also calculates a commitment of it $C = Commit(b)$. Additionally, Bob provides his unique, unforgeable hardware ID, and concatenates it to his commitment $C' = C\|\ ID$.
- **Authentication:** Bob sends C' to the public server which uses a Digital Signature scheme to sign C' to produce $\sigma(C')$. The server sends $\sigma(C')$ to Bob and ensures it *never* gives Bob another signature for a different commitment, by making sure it only ever signs one C' per ID.
- **Verification:** Alice receives $(b, C\|ID, \sigma(C'))$ from Bob. She uses the server's public signature key to ensure that C' was authentically signed by the server and was in fact a signature of $C\|ID$. She also verifies that the ID is really Bob's ID. Finally, she asks Bob to prove that the b he sent was the b that produced the commitment C (this is called the 'reveal' step in commitment scheme parlance). If all three checks succeed, Alice adds $A_t \cdot b$ to her ledger to be uploaded in the case of her infection. Otherwise, Alice does nothing.

Theorem 5. *Alice only accepts the the interaction if Bob sends the unique b that he committed to at the beginning.*

Proof. Since S only ever signs one C' for Bob, by the security of the signature scheme he cannot possibly provide a signature for a different commitment C'' for a $b' \neq b$. Additionally, by the security of the commitment scheme he couldn't find a different $b' \neq b$ that results in the same commitment C'.

Thus Bob can't get away with sending a $b' \neq b$ without either the signature $\sigma(C')$ not matching the commitment he sends C, or the commitment he sends C not matching the b' he sends. □

4 Analysis of the protocol

This protocol accomplishes almost all of the criteria given in section 2, with some caveats.

There are also some shortcomings of the protocol and the problem definition I have given, highlighting the difficulty of designing secure contact tracing protocols.

Theorem 6. *This protocol is a contact tracing protocol*

Proof. If Alice and Bob both follow the protocol, and Alice gets sick, then Alice will upload to the server a certificate of their interaction such that Bob can factor it if he checks the public ledger. □

Theorem 7. *Bob can only determine the identity of Alice with certainty if he fixes his hardware to faithfully use his authenticated b exactly once (during his interaction with Alice)*

Proof. I have already shown that Bob can only determine the identity of the person who infected him with probability $1/|I|$. He can ensure that he can determine it was Alice that infected him if he only ever sends b to Alice, and otherwise doesn't send any messages to anybody else. □

As a corollary, if Bob wants to track who got him sick, he must sacrifice a new piece of hardware per person he interacts with.

Due to theorems 3 and 4 the conditions b.) and c.) of perfect privacy are also met. Unfortunately, condition d.) is not met, because anyone who has met Bob knows his b and can therefore check to see whether he has been exposed.

A weakened form of integrity is met: Mallory can only inform Bob that he was exposed if someone Mallory knows shares Bob's key with her, and Mallory has someone who received a positive diagnosis fraudulently upload a certificate involving Bob's key.

5 Discussion

The risk to integrity is inherent to the trade offs made to keep Alice's identity safe from Bob, it made it so that Bob always sends out exactly the same message, creating the potential for someone to upload fake certificates of Bob's exposure as long as they coordinate with someone who is actually sick.

Additionally, an individual may be able to observe the public ledger and calculate common denominators to determine the private keys of all the exposed individuals. This is also due to the fact that Bob sends the same message every time. I'm not sure the severity of this threat, as learning the identity associated with a private key requires someone affiliated with the malicious actor to be in the presence of the individual holding that key. This would require large scale mobilization of resources. Though this does seem like too big a risk to run, though it may be mitigable if you allow people to refresh their private keys every two weeks.

This issue, however, strikes at the heart of the how difficult it is to trade off privacy for the infected individual and security of other people on the network. I tried to accomplish something similar to this protocol using homomorphic hashing, but it had the property that if Bob sends a different value each time, he can still always figure out at what time he was exposed. I'm not sure whether there is a way that prevents Bob from knowing when he was exposed without revealing to others that he was exposed.