Decentralized Privacy-Preserving Proximity Tracing

Based on the document

https://github.com/DP-3T/documents

Elena Kirshanova

Course “Information and Network Security” 12 апреля 2020 г.

Goal

How to identify people who have been in contact with an infected person?

Academics issued a proposal of a design of an app aiming at proximiti tracing – determining who has been in close physical proximity to an infected person.

Main purpose: quick notification of contact people

Ideally the app should ensure security and privacy for users.
This mini-lecture: overview of the design proposal

Please visit `https://github.com/DP-3T/documents` for details

High level idea

High level idea

- the app generates a pseudorandom ID (using a PRG) that represents a user
- This ID is continually broadcasted via BlueTooth
- the app records all IDs observed in the proximity
- if $ID^\star$ is diagnosed for the virus, the user $ID^\star$ uploads all its recorded IDs to a central server
- the app once in a while downloads data from the severs and locally checks if any of the recorded IDs has been diagnosed for the virus

High level idea

- the app generates a pseudorandom ID (using a PRG) that represents a user
- This ID is continually broadcasted via BlueTooth
- the app records all IDs observed in the proximity
- if $ID^\star$ is diagnosed for the virus, the user $ID^\star$ uploads all its recorded IDs to a central server
- the app once in a while downloads data from the severs and locally checks if any of the recorded IDs has been diagnosed for the virus

Privacy considerations:

- the central server only stores IDs of infected people (no proximity data is uploaded)
- Nobody can track non-infected users seeing only their IDs
- No entity should be able to use the known data for malicious purposes (e.g.,)

More details I

1. Secret Key Generation:
   For the day t, the smartphone generates a random key S_t:

   $$S_t \leftarrow \mathsf{KeyGen}(t)$$

   using, for example, a PRG (see lecture 2).

   For the next days, the secret keys S_{t+i} are generated using a cryptographic hash function $\mathcal{H}$:

   $$S_{t+i} = \mathcal{H}(S_{t+i-1})$$

More details I

1. Secret Key Generation:
   For the day t, the smartphone generates a random key S_t:

   $$S_t \leftarrow \mathsf{KeyGen}(t)$$

   using, for example, a PRG (see lecture 2).

   For the next days, the secret keys S_{t+i} are generated using a cryptographic hash function $\mathcal{H}$:

   $$S_{t+i} = \mathcal{H}(S_{t+i-1})$$

2. Ephemeral ID generation: Each user changes his `EphID` n times per day (tunable parameter).
   At the beginning of day t, the app generates n ephemeral IDs `EphID` as

   $$\texttt{EphID}_1 || \ldots || \texttt{EphID}_n = \mathsf{PRG}(\mathcal{H}(S_t, \text{“}\texttt{broadcast key}\text{”})),$$

More details I

1. Secret Key Generation:
   For the day t, the smartphone generates a random key S_t:

   $$S_t \leftarrow \mathsf{KeyGen}(t)$$

   using, for example, a PRG (see lecture 2).

   For the next days, the secret keys S_{t+i} are generated using a cryptographic hash function $\mathcal{H}$:

   $$S_{t+i} = \mathcal{H}(S_{t+i-1})$$

2. Ephemeral ID generation: Each user changes his $\texttt{EphID}$ n times per day (tunable parameter).
   At the beginning of day t, the app generates n ephemeral IDs $\texttt{EphID}$ as

   $$\texttt{EphID}_1 || \ldots || \texttt{EphID}_n = \mathsf{PRG}(\mathcal{H}(S_t, \text{“}\texttt{broadcast key}\text{”})),$$

   Each $|\texttt{EphID}_i|$ is 16 bytes (Bluetooth Low Energy beacons payload)

More details II

3. Local storage of observed data: “Seeing” any `EphID` in proximity, each smartphone stores
   - `EphID`
   - duration of contact
   - proximity
   - time indication (April, 13)

   This whole data occupies approx. 32 bytes.

More details II

3. Local storage of observed data: "Seeing" any `EphID` in proximity, each smartphone stores
   - `EphID`
   - duration of contact
   - proximity
   - time indication (April, 13)

   This whole data occupies approx. 32 bytes.

4. If a user is infected: on the day t:
   - The user sends S_t to the central server
   - The sever computes all $\texttt{EphID}_i$ from the day t up until today (knowing S_t enable to do that)
   - The server additionally checks if any of the $\texttt{EphID}_i$ has been recorded *before* S_t has been published
   - The infected user chooses a completely new secret key

More details II

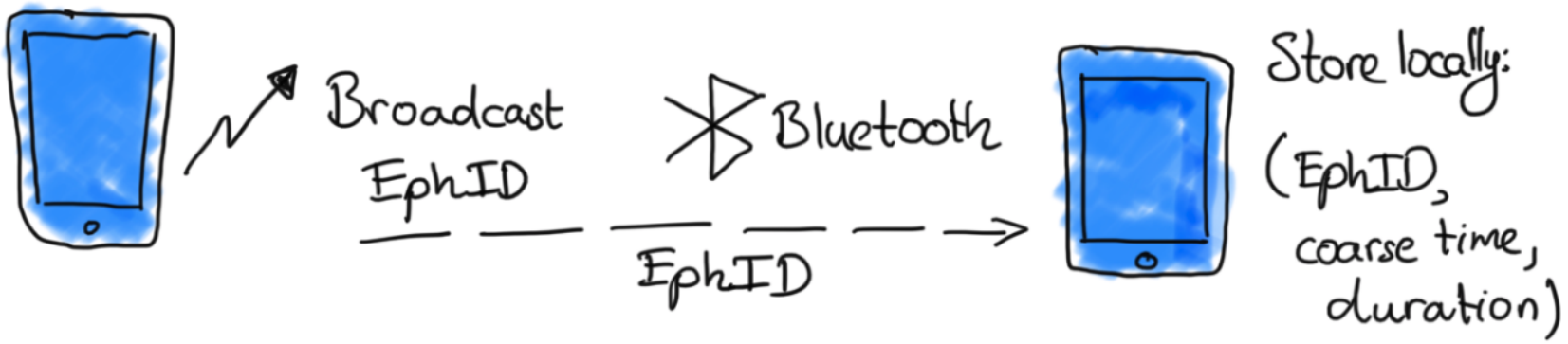
3. Local storage of observed data: "Seeing" any `EphID` in proximity, each smartphone stores
   - `EphID`
   - duration of contact
   - proximity
   - time indication (April, 13)

   This whole data occupies approx. 32 bytes.

4. If a user is infected: on the day t:
   - The user sends S_t to the central server
   - The sever computes all $\texttt{EphID}_i$ from the day t up until today (knowing S_t enable to do that)
   - The server additionally checks if any of the $\texttt{EphID}_i$ has been recorded *before* S_t has been published
   - The infected user chooses a completely new secret key

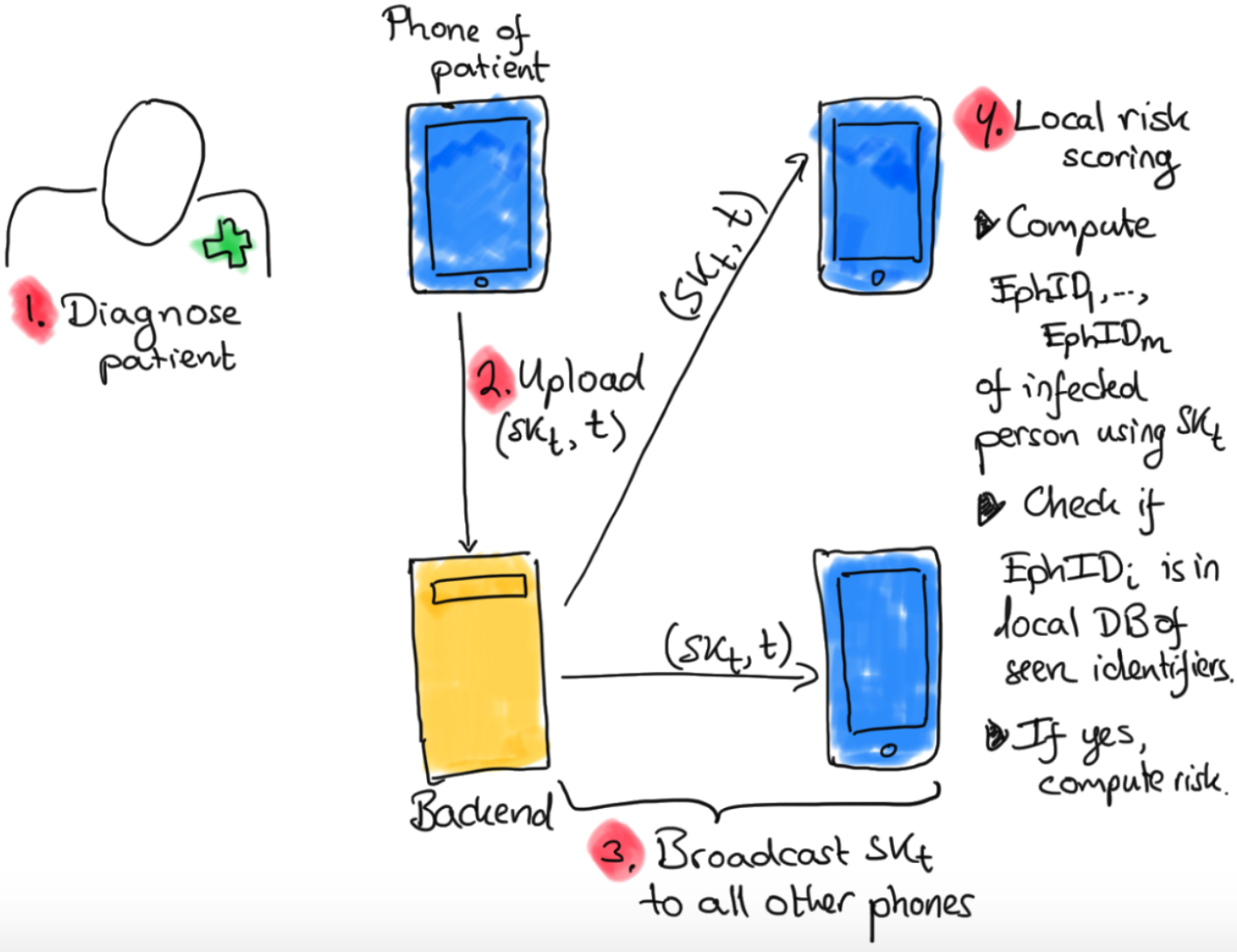
5. Notification of risk:
   - The server broadcasts secret keys (S_t, t) of infected users
   - Each user loads (S_t, t) and recomputes the corresponding $\texttt{EphID}'$s
   - And checks these $\texttt{EphID}'$ against the locally stored ones.

In picture I



The picture is taken from DP3T White Paper.pdf
`https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf`

In picture II



The picture is taken from DP3T White Paper.pdf
https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf