Delayed Authentication
Replay and Relay Attacks on DP-3T

Krzysztof Pietrzak

IST Austria
pietrzak@ist.ac.at

April 13, 2020

Abstract

Decentralized Privacy-Preserving Proximity Tracing (DP-3T) is an open protocol which aims to help fight the current pandemic. Their proposal is an app for mobile phones which broadcasts frequently changing pseudorandom identifiers via (low energy) Bluetooth, and at the same time, the app stores IDs broadcast by phones in its proximity. Only if a user is tested positive, their IDs of the last 14 days are published so other users can check if they have stored them locally and thus were close to an infected person.

Vaudenay [eprint 2020/399] observes that this basic scheme succumbs to relay and even replay attacks, and proposes more complex *interactive* schemes which prevent those attacks without giving up too many privacy aspects. Unfortunately interaction is problematic for this application for efficiency and security reasons.

In this note we suggest a very simple and efficient *non-interactive* solution to prevent relay and replay attacks while preserving most of the privacy guarantees. In a nutshell, we let the users broadcast their IDs together with their current time (and if we want to prevent relay attacks, also location). The time and location are authenticated using a concept we term "delayed authentication": We propose a message authentication code, where one can first check that the message matches the tag without knowing the key. The message and parts of the tag can then be deleted, and this delayed tag is independent of the message, but it's still possible to finish verification of the tag later when the key become known.

This means a receiving party can first check if the time and location match its own, but then delete those values, so this sensitive data is never locally stored. Should the sender be tested positive and his keys get released, the receiving party still can authenticate the delayed tag to detect a potential replay or relay attack.

1 Introduction

Proximity tracing aims to simplify and accelerate the process of identifying people who have been in contact with the SARS-CoV-2 virus. A concrete proposal is DP-3T [TPH+20], which stands for Decentralized Privacy-Preserving Proximity Tracing.

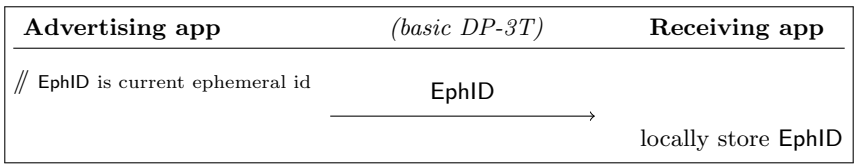

Figure 1: The basic DP-3T protocol. If at some point a user is reported sick the app will get (the keys required to recompute) its EphID's of the last 14 days, and if it has locally stored one of those EphID's it must assume it was close to an infected user.

In the DP-3T proposal users are assumed to have a mobile phone or another Bluetooth capable device with their application installed. At setup the app samples a random key $\mathtt{SK}_0$. This key is updated every day as $\mathtt{SK}_i = H(\mathtt{SK}_{i-1})$ using a cryptographic hash function H. Each key defines n Ephemeral IDentifiers (EphID) derived from $\mathtt{SK}_i$ using a pseudorandom generator and function as

$$\mathsf{EphID}_1 \| \mathsf{EphID}_2 \| \ldots \| \mathsf{EphID}_n := \mathsf{Prg}(\mathsf{Prf}(\mathtt{SK}_i, \text{“broadcast key”}))$$

Those EphID's are used in a random order during the day, each for $24 \cdot 60/n$ minutes (say 30 minutes if we set $n = 48$). The current EphID is broadcast using Bluetooth in regular intervals to potential users in its proximity.

Phones locally store the EphID's they receive as well as their own $\mathtt{SK}_i$'s of the last 14 days. If a user tests positive the health authority can upload the user's $\mathtt{SK}_i$ keys of the last 14 days to a backend server, which will distribute them to all users. The users will recompute the EphID's from the received keys and check if there's a match with any of their stored EphID's. If yes, it means they've presumably been in proximity to an infected person in the last two weeks and should self isolate (this description is oversimplifying several aspects that are not relevant for this note). Important aspects of this protocol are its simplicity, in particular the fact that the protocol is non-interactive as illustrated in Figure 1, and its privacy properties. Users only need to locally store pseudorandom EphID's, but no time or location data.

2 Replay and Relay Attacks

Vaudeny [Vau20] discusses potential attacks on this scheme including replay and relay attacks. As an illustration of a replay attack consider an adversary who

collects EphID's in an environment where it's likely infections will occur (like a hospital), and then broadcast those EphID's to users at another location, say a competing company it wants to hurt. Later, when a user from the high risk location gets tested positive, the people in the company will be instructed to self isolate.

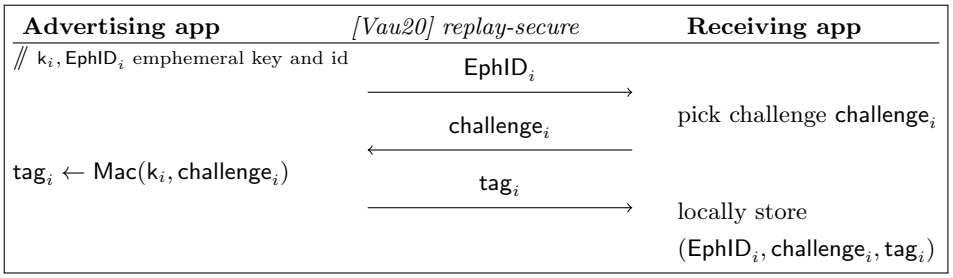

Figure 2: Vaudenay's protocol secure against replay attacks. When later the user of the receiving app gets keys of infected users it will check for every tuple $(\mathsf{k}, \mathsf{EphID})$ derived from those keys if it locally stores a triple $(\mathsf{EphID}', \mathsf{challenge}, \mathsf{tag})$ with $\mathsf{EphID} = \mathsf{EphID}'$. If yes, it will check whether $\mathsf{tag} \stackrel{?}{=} \mathsf{Mac}(\mathsf{k}, \mathsf{challenge})$ and only if this check verifies, assume he was close to an infected party.

Vaudenay suggest an extension of the basic DP-3T protocol, shown in Figure 2, which is secure against replay attacks, but it comes at the prize of using interaction. For efficiency and security reasons, the current DP-3T proposal uses Bluetooth low energy beacons, which makes interaction problematic (we refer to [TPH+20] for more details).

A relay attack is a more sophisticated attack (than a replay attack). Here the adversary relays the messages from one location (e.g. the hospital) to another (e.g. the company it wants to hurt) in real time, the protocol from Figure 2 is not secure against relay attacks. Vaudenay also suggests a protocol which thwarts relay attacks assuming both devices know their location (using GPS) and with an additional round of interactions. Our relay secure protocol also requires location data, but achieves security against relay attacks without interaction.

3 Delayed Authentication

The main tool in our protocols is a message authentication code that allows for a two step authentication process, where the first doesn't require the key, and the second is independent of the authenticated message. To construct it, apart from a standard message authentication code $\mathsf{Mac} : \mathcal{K} \times \mathcal{X} \to \mathcal{T}$ we'll use a hash function $H : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ with the following properties:

collision resistant: It must be difficult to find tuples $(x, r) \neq (x', r')$ where $H(x, r) = H(x', r')$.

randomizing: For any $x \in \mathcal{X}$ and a uniformly random $\mu \leftarrow\!\!\!\!\!\!\$\ \mathcal{R}$ the hash $H(x, \mu)$ is close to uniform given x.

Even though randomizing is not a standard assumption, we can expect a well designed cryptographic hash function to satisfy the condition when $\mathcal{R}$ is sufficiently large. The property can actually be relaxed, we just need $H(x, \mu)$ to be independent of x, not uniform, and even this weaker property must only hold computationally, not information theoretically.

Consider the following randomized message authentication code

$$\mathsf{DelayMac}(\mathsf{k}, x) \to (\ H(x,\mu)\ ,\ \mu\ ,\ \mathsf{Mac}(\mathsf{k}, H(x,\mu))\ ) \text{ where } \mu \leftarrow\!\!\!\!\!\!\$\ \mathcal{R}$$

We can verify such a message/tag pair $x, (h, \mu, \mathsf{tag})$ in two steps. First we check that $h \stackrel{?}{=} H(x, \mu)$ is the correct hash (providing h is not required as it can be computed from other known values, it's still convenient to do it), and if this is the case can forget about μ and the message x. Note that this step does not require k, we call (h, tag) the delayed tag. Later, should the key k become known to us, we can authenticate the delayed tag checking $\mathsf{tag} \stackrel{?}{=} \mathsf{Mac}(\mathsf{k}, h)$.

The security of $\mathsf{DelayMac}$ as a standard message authentication code follows easily from the security of Mac and the collision resistance of H. Moreover the randomizing property of H implies the delayed authentication tag (h, tag) is almost independent of the message x.

4 Our Protocol

Let us illustrate how we use delayed authentication in our protocol which is given if Figure 3. As in [Vau20], apart from the EphID's,

$$\mathsf{EphID}_1 \| \mathsf{EphID}_2 \| \ldots \| \mathsf{EphID}_n := \mathsf{Prg}(\mathsf{Prf}(\mathtt{SK}_i, \text{“broadcast key”}))$$

the app additionally computes an ephemeral secret key k_i with each EphID_i

$$\mathsf{k}_1 \| \mathsf{k}_2 \| \ldots \| \mathsf{k}_n = \mathsf{Prg}(\mathsf{Prf}(\mathtt{SK}_i, \text{“secret key”}))$$

The replay secure protocol know works as follows

- (broadcast) The app regularly broadcasts its current EphID together with the current time t_1 and a delayed authenticator $(h, \mu, \mathsf{tag}) \leftarrow \mathsf{DelayMac}(\mathsf{k}, t_1)$ for t_1 using the corresponding ephemeral key k.
- (receive) If the app receives a message $(t_1, (h, \mu, \mathsf{tag}), \mathsf{EphID})$ it checks if its current time t_2 is close enough to t_1 (we'll discuss what this means in §4.1), and if the hash $h \stackrel{?}{=} H(t_1, \mu)$ matches. If this doesn't hold it ignores the message assuming it's a replay attack or otherwise malformed. If the check passes, the app will locally store $(h, \mathsf{tag}, \mathsf{EphID})$.

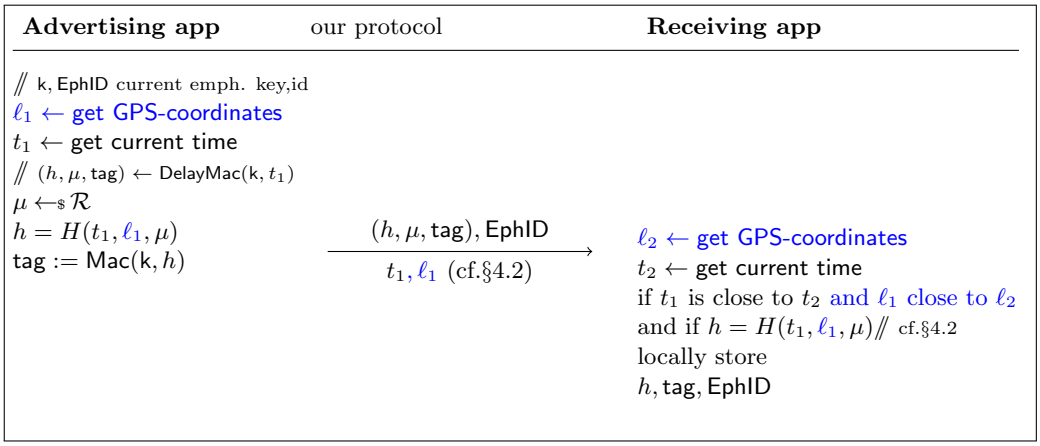

Figure 3: Our non-interactive proximity tracing protocol that is secure against relay attacks. By ignoring the blue text we get a protocol that is only secure against replay attacks but doesn't require location data, which might not always be available or due to security reasons not desired. If later a user gets sick and the app gets its $(\mathsf{k}, \mathsf{EphID}')$ pairs, it will check if it has stored a triple $(h, \mathsf{tag}, \mathsf{EphID})$ with $\mathsf{EphID} = \mathsf{EphID}'$, and if so, finish verification by checking the delayed tag $\mathsf{tag} \stackrel{?}{=} \mathsf{Mac}(\mathsf{k}, h)$ to detect potential replay or relay attacks.

- (verify) If the app learns ephemeral ID/key tuples $(\mathsf{EphID}', \mathsf{k})$ of infected parties from the backend server, it checks if it stored a tuple $(h, \mathsf{tag}, \mathsf{EphID})$ where $\mathsf{EphID} = \mathsf{EphID}'$. For every such tuple it checks if $\mathsf{Mac}(\mathsf{k}, h) \stackrel{?}{=} \mathsf{tag}$, and only if the check passes, it assumes it was close to an infected party.

To see how this approach prevents replay attacks, we note that for a successful replay attack where the receiving party finally accepts EphID receiving a broadcast message claiming a later timepoint $t' > t_1$, one needs to break the security of DelayMac, that is, either find a collision $H(t', \mu') = H(t_1, \mu)$ or forge a tag for Mac.

If we additionally authenticate the location like this, as shown in blue in Figure 3, we achieve security against relay attacks.

4.1 Closeness

What "close" means (for time and location) in the protocol must be formally specified, and depends amongst other things on what synchrony we want to assume from the clocks and what precision we can expect from the GPS. Some minor replay and relay attacks, where the time is within the clock synchrony, and the location within the GPS precision, are still possible with our protocols, but don't seem practically relevant.

4.2 Communication

The length of the broadcast messages in our protocol is larger than in the basic DP-3T scheme, where one just broadcasts the EphID. We observe that by making the broadcast timing and location data coarse enough, the advertizing app doesn't have to broadcast them at all as its time and location (t_1, ℓ_1) will likely match the (t_2, ℓ_2) of the receiving party, and even if not (as the value is right on the boundary), the receiver can check the few neighbouring values that would be considered close enough. If t_1, ℓ_1 are not sent, one must send h, which is why we added h to the tag.

4.3 Privacy Issues with Malicious Apps

If the advertizing and receiving application are honest (i.e., run the specified code) then our protocols prevent replay (or even relay) attacks while preserving the privacy of both by jointly not storing any useful information beyond the fact that the two parties have met and how many times. In particular, everything they store is independent of any time or location data, apart from coarse timing information one can already get from the EphID's in CP3T (i.e., the days of the encounters).

But a malicious receiver can produce digital evidence about the advertizing user. This is also the case for Vaudenay's protocols (and already mentioned in [Vau20]). As a concrete example, the receiver can put a hash of the entire transcript (for our protocol it's just the broadcast message) on a blockchain to timestamp it. Later, should the advertizer test positive and his keys being released, this transcript will reveal the time (and for the relay secure protocol also location) data. Timestamping is necessary here, as once the keys are public everyone can produce arbitrary transcripts.

Finally, we stress that many of the issues discussed in [TPH+20] and [Vau20] also apply to our schemes.

References

[TPH+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salath, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth Paterson, Srdjan Capkun, David Basin, Dennis Jackson, Bart Preneel, Nigel Smart, Dave Singelee, Aysajan Abidin, Seda Guerses, Michael Veale, Cas Cremers, Reuben Binns, and Thomas Wiegand. Dp3t: Decentralized privacy-preserving proximity tracing, 2020. `https://github.com/DP-3T`.

[Vau20] Serge Vaudenay. Analysis of dp3t. Cryptology ePrint Archive, Report 2020/399, 2020. `https://eprint.iacr.org/2020/399`.