Hashomer – A Proposal for a Privacy-Preserving Bluetooth Based Contact Tracing Scheme for Hamagen

Benny Pinkas* Eyal Ronen†

Version 0.93 – April 27, 2020

Abstract

In recent weeks multiple proposals for schemes allowing contact tracing for combating the spread of COVID-19 have been published. Many of those proposals try to implement this functionality in a decentralized and privacy-preserving manner using Bluetooth Low Energy (BLE). In this whitepaper, we present "Hashomer", our proposal for a contact tracing scheme tailored for the Israeli Ministry of Health's (MoH) "Hamagen" application. A prototype implementation of this design can be found at https://github.com/eyalr0/HashomerCryptoRef. The design has the following properties:

- Similarly to many other solutions (e.g., DP-3T, Google and Apple, and PACT), our scheme is completely decentralized, but with specific tradeoffs between privacy, security, and explainability. To guarantee privacy, **no messages are ever sent from the application to any server, except for voluntary messages by COVID-19 positive persons or by users who were notified (by the application) about an exposure to a different user who had been diagnosed as COVID-19 positive** .
- To further protect privacy, different BLE messages of the same person cannot be linked to each other. The only exception is for messages of persons who were found to be COVID-19 positive and agreed to notify their contacts. Messages of such a person, which were sent within a short time period, for example one hour, can be linked to each other.
- For performance, the BLE message size is limited to 16 bytes. The application uses only symmetric key cryptography (AES and HMAC). Contact updates from the MoH are of limited size.
- For explainability (namely, convincing users that they were exposed to a COVID-19 patient), we let users who were exposed to a patient learn the time of contact. This also implies that users can learn, using the phone's GPS information, the location of the exposure to the patient.
- The design also prevents relay attacks, namely attacks where a malicious receiver relays BLE transmissions from one location to other locations. As the geo-location is already known to the receiver, we prevent these attacks by encrypting and MACing the geo-location information. To clarify, the relevant checks are done locally and **no location information is ever uploaded** to the MoH server. (The relay prevention feature can removed if needed, and in that case the application can run without any location data.)
- The design enables both the user and the MoH to redact contact tracing information for specific time intervals. This is useful both for privacy (for example, letting users redact private meetings which they want to hide), and for efficiency (e.g., redacting times when it is known that the user was alone). To facilitate this, we use a tree structure for key derivation, allowing the MoH to broadcast only a subset of the keys.
- To prevent false reports about exposure to infected persons, the application can provide users with a short string proving that they had a contact with a COVID-19 positive person.

Readers who are familiar with contact tracing, can initially focus on Sections 1 and 3, which describe the main features that our design aims to achieve, and describe a technical overview of the design.

*Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel. Email: benny@pinkas.net

†School of Computer Science, Tel Aviv University, Tel Aviv 76100, Israel. Member of CPIIS. Email: er@eyalro.net

1 Executive Summary

(This section is intended for readers who are familiar with the concept of BLE contact tracing.)

Our design follows the same structure as other designs for *decentralized* contact tracing. We would first like to emphasize several privacy features that our design supports: (1) The ephemeral IDs sent in BLE messages are computed from keys which are generated in the device, and look random to any other entity. (2) The decision whether to reveal contact tracing information is made by the user, and is completely voluntary. Similarly, a notification about exposure to a COVID-19 positive person is shown in the application, and is kept hidden from the MoH unless the user wishes to report it. (3) To ensure these features the application never sends any information to any server (except when requested by the user). (4) The application does not reveal to the MoH any location information, not even of COVID-19 positive persons.

While the basic idea and design are similar to that of other designs, and in particular DP-3T, our design has some privacy and security features which are not present in most other contact tracing designs. We list here the main differences with respect to the DP-3T design (as of April 20, 2020).

Tradeoff Between Privacy and Explainability The DP-3T design prioritizes the privacy of the patient and therefore provides to exposed users only a coarse time window about the time of the exposure (e.g., the morning of April 9). We think that explainability is important in order to convince user to quarantine themselves. Namely, that the application must provide users with a more fine grained time frame about the time of exposure (a few minutes).

Prevention of Relay Attacks The fact that the application saves fine-grained timing information implicitly leaks the location of the contact (we assume that the receiving phone stores GPS information). We therefore chose to incorporate the location data of the place of contact into the protocol, in order to prevent relay attacks (attacks which relay BLE messages from one location, for example an emergency room, to other locations).

The BLE message therefore includes an encryption of coarse geo-location information (e.g., accuracy of ± 500 meters) and MACs it to verify its authenticity. The encryption and signing keys are only revealed voluntarily by confirmed infected persons. *This data is never sent to the server.* This information is used to verify that both sender and receiver were in close physical proximity.

Allowing Partial Disclosure Our design lets the user or MoH redact contact tracing information for specific periods of time. This is important for privacy, for accuracy (for example, for periods where the patient was sitting alone in a car in traffic jam next to many other cars), and for efficiency – in order to reduce the size of the update broadcast message.

Enabling users to prove an exposure to a COVID-19 patient The design enables users who are identified by the application about an exposure to a COVID-19 patient, to prove this fact to the server. This prevents users from faking such exposures (for example, in order to get a sick leave). On the other hand, sending this proof discloses to the server the specific patient to whom a user was exposed.

2 Introduction

2.1 Basic Architecture

The basic architecture includes copies of the application that run on clients' phones, and a central server run by the MoH. Following are the basic properties of the operation of the system:

1. When the application is installed on a phone it generates a personal random master key. This key defines daily master keys, and subsequently all messages sent by the application. This key is unrecoverably deleted after installation.

2. The application periodically transmits ephemeral IDs using BLE messages. The ephemeral IDs include a 16 byte pseudo-random value that is generated by the application. (The length of messages is 16 bytes due to constraints of the communication layer.) These message are received by phones running the application which are within a short range.

3. When the application receives such an ephemeral ID, it saves it together with the current location, and the time when it was received (both potentially rounded to be in a more coarse resolution).

4. The application never sends anything over any channel other than BLE. The only exception is by an *explicit request of the user*, and only when the user was identified as a COVID infected person or as being exposed to a COVID infected person.

5. A user who was identified as a COVID patient can ask the application to send a message to the MoH, which will enable the identification of all messages sent by the user in the last 14 days. The overall system should provide a mechanism to ensure that only infected persons are able to upload keys, based on confirmed test results.

   Note that the information uploaded to the server *does not reveal* to the MoH the locations visited by the user, but should only enable other users who were exposed to the user to identify this fact.

6. The server broadcasts to all users the required to identify the ephemeral IDs sent by new infected persons. The application will run an algorithm which decides, based on these messages, whether the exposure to the patient makes the user at risk, and in that case suggests to the user to self isolate or contact the MoH.

   The broadcast must protect the privacy of the infected person (so that it is hard to link different exposures to the same infected person), must support a large number of new infected persons, and must be of reasonable size (say, 5 MByte per day). The broadcast is expected to be sent to millions of users.

2.2 Privacy and Security Requirements

Privacy requirements

1. No information is revealed to the MoH without the user's explicit consent.

   To demonstrate this property to users, no information is sent from the user's application unless the user chooses to do so. (The only exception is BLE communication, which is short range and pseudo-random.)

2. If the application learns that the user was exposed to a COVID patient, this information is only revealed to the user and is not sent to any other party (unless the user chooses to update the MoH).

3. Data collection and usage must be limited to the purpose of proximity tracing.

4. The information sent by a COVID patient to the MoH must not identify the users who were in close proximity to the infected persons.

5. The information sent from the MoH to users, must not include any information related to the identify infected persons. Note that other users who were in close physical proximity to the infected person might still be able to use this information to learn the infected person's identity. E.g., if a user is notified of a contact in a time she was in close proximity to only one person, she will know that person was infected.

6. Both the application and the server *must delete all information older than 14 days.* This includes both secret keys and stored beacons.

7. The application must *allow the user to "snooze"* the sending of beacons in privacy critical situations (e.g., a journalist meeting with a confidential source).

8. The application must *allow a user to delete keys* associated with specific time intervals. These keys will not be reported if the user is identified as a COVID-19 patient. This is required to retroactively protect the user's privacy in critical situations when she didn't snooze the application.

9. The application must allow users *to delete their history of exposure alerts*, and the ephemeral IDs that caused the alerts. This feature is important in order to prevent other parties, such as employers, from coercing users to demonstrate that they have not received any exposure alerts. By enabling users to erase their exposure history, others cannot identify users who received such alerts.

Security requirements

1. Security against relay attacks: an attacker might receive messages sent by users of the application in one location, for example, an emergency room in a hospital, and transmit them in different locations (say, busy train stations). This might causes unnecessary warnings showed to users. The application should prevent this type of attacks.

2. The scheme will provide COVID-19 positive users with the ability to send a commitment to their keys to the MoH. This allows the MoH to verify the origin of uploaded keys, and that these keys have not been changed or replaced since the commitment.

3. Prevention of "fake" exposures – The design should prevent users from claiming that they were in contact with an infected person (for example, by modifying the local database of the application). Some users might opt to do so, for instance, if they want to get 14 days of sick leave. This attack can be partly prevented by including a cryptographic verification value in the ephemeral ID that is sent over BLE. Note that a user who had a real contact with an infected person can always transfer this information to other users, and then these users will be able to fake a contact. However, the server can become suspicious and do more checks if too many people use the same verification key. Note that this proof will reveal to the server the identity of the user and the fact that he was in contact with that specific infected person.

4. Cloning attacks: A sophisticated attacker might clone multiple phones to use the same master key. Then if any of the users of these phones is revealed as a patient, warnings are sent to anyone who was exposed to any of these phones. Note that the current scheme does not prevent this attack.

5. Copying attacks: A sophisticated attacker might identify that a user is a likely patient, and then break into this user's phone, copy data, and send in many locations BLE messages that seem to originate from that phone. Note that the current scheme does not prevent this attack.

2.3 Operational Constraints

1. **Short messages:** BLE communication enables each phone to send a relatively short message. We assume this message to be 16 bytes long.[1]

2. **Efficiency:** the application running on the phone should use limited CPU resources. We therefore limit the application to using only symmetric key cryptography (such as AES encryption, and computing hash functions), rather than using public key cryptography. Calculations which are run in near real time (i.e. computing the ephemeral ID) use only AES. The more computationally demanding HMAC operations are only run every once in a while.

[1] Some of the other projects use the same message length, for example the Privacy-Preserving Contact Tracing project of Apple and Google.

3. **Explainability:** In order to convince users that they were indeed exposed to a COVID patient, they should be given relatively specific information about the time of exposure.

4. **Limited download channel:** A server run by the MoH will publish the set of BT ephemeral IDs that were sent in the last 14 days by newly identified infected persons. Each copy of the application must download this data, but we can only expect users to download a few megabytes of data per day. We work under the following assumptions: (1) The system should support up to 1000 new infected persons per day; (2) For each patient we need to report an ephemeral ID for each 5 minute window in the last 14 days. These assumptions require reporting $1000 \times 12 \times 24 \times 14 \approx 4 \cdot 10^6$ values every day. Even reporting 5 bytes per value requires the application to retrieve 20MB every day, which is too much (our design reports 8 bytes per ephemeral ID). In order to reduce the size of the downloaded data, sequences of ℓ consecutive ephemeral IDs will be generated from the same seed (specifically, we can set $\ell = 12$ to correspond to one hour of ephemeral ID messages). The download file will only include these seeds.

3 Technical Overview

The basic straw man design for a contact tracing scheme can have each instance of the application send short random messages (ephemeral IDs) over BLE. The application also records all messages that it receives BLE. When a user is identified as being COVID-19 positive, the user can provide the ministry of Health with all the ephemeral IDs that were sent by its copy of the application in the last 14 days. The MoH periodically distributes these values, received from all new COVID-19 positive persons, to all users, where for each time period the MoH sends all corresponding ephemeral IDs in random order. Then any copy of the application can locally check if it previously received any of these ephemeral IDs over BLE, and use a local algorithm to alert the user to the risk of contact with the corona virus (for example, alert the user if ephemeral IDs were consecutively received in any period of 15 minutes.) The user is then able to inform the MoH that he or she were in contact with COVID-19 positive person.

This initial design provides privacy for all receiving users since no information is sent from their phones, except for short *random* BLE messages. The BLE messages can also be generated in each personal copy of the application based on a secret random seed, and look pseudo-random to all other users. The privacy of COVID-19 positive persons is preserved in the sense that it is impossible to link different BLE messages of the same person.

This basic scheme has a scalabily issue: Suppose that users change their ephemeral IDs every 5 minutes. Then every user sends more than 4000 ephemeral IDs in the 14 day period. In this case an update message for about 1000 or 10000 new COVID-19 patients will be too large. (This is true irrespectively of the compression method that is used, such as a Bloom filter or a Cuckoo filter, as long as we want to keep the false positive probability low.)

Daily seeds: An appealing option is for the client to use a separate seed per day, and use it to derive all ephemeral IDs of that day. In this case the MoH needs to send only 14 seeds per COVID-19 positive person. However, this approach suffers from some drawbacks: The privacy of the COVID-19 positive persons is affected since it is now possible to link different ephemeral IDs sent by the same person in the same day. The client application needs to generate all ephemeral IDs from the seeds, and this task might be computationally heavy. In addition, it is impossible to redact less than an entire day of the BLE message history of COVID-19 patients, namely have the MoH send an update broadcast which does not include some of the ephemeral IDs sent by a COVID-19 positive person. (The redaction feature is important for privacy, efficiency, and accuracy.)

A hybrid design: We use a hybrid approach in our design. An initial master key is generated when the application is installed, and a chain of daily master keys is generated from it. The application can keep the master day key of two weeks ago, and erases all prior keys. Each day is divided to epochs (e.g., hours), and each epoch is divided to time units (e.g., 5 minutes). Every epoch has a different epoch key which is generated from the daily key. The ephemeral ID is changed each time unit, and is derived from the current

epoch key. When it is needed to report the ephemeral IDs sent by a COVID-19 positive person, only relevant epoch keys that were used by the person are broadcast to all users. This prevents linking BLE messages sent by the same person in a period longer than a single epoch, and also enables to limit the report to epochs which need not be kept private, and where it is known that the person was not alone. If the number of new patients is very large, and the overhead of broadcasting epoch keys is too large, then it is possible to broadcast the daily keys of some or all of the new patients and thus improve bandwidth at the cost of reducing privacy.

A tradeoff between privacy and explainability: We assume that users who are informed that they were exposed to a patient would require a rather detailed information in order to trust this report. (This is issue arguably depends on the local culture.) We therefore have the application store the location and time of each reception of an ephemeral ID. This information is presented to the user when an exposure is reported to him/her.

Proving exposure to COVID-19: In some cases users might falsely report that they were exposed to a COVID-19 positive person, in order to gain benefits such as priority in COVID-19 testing or a paid sick leave. We therefore add an option for users to prove this exposure. This is done by defining part of the ephemeral ID as a function of an epoch verification key. When a user is identified as COVID-19 positive he or she sends a master verification key to the MoH. This key and the derived verification values are not broadcast to all users. Users are still able to identify ephemeral IDs sent by COVID-19 positive persons. They can prove the exposure to the MoH by providing the part of the ephemeral ID which depends on the verification key. (The challenge is supporting this feature while keeping the size of the ephemeral IDs to be only 16 bytes)

Preventing Relay attacks: A potential attack scenario is where an attacker relays messages from one location where COVID-19 positive persons are likely to appear to many other busy locations, thus causing many false reports of exposure to patients. In order to overcome this attack we have the sender of an ephemeral ID include in it an encrypted and MACed GeoHash of its current location. The receiver stores the location in which it received this message. When it receives the update that is broadcast by the MoH and identifies a match in part of the ephemeral ID, the receiver is able to decrypt and verify the GeoHash encrypted in the ephemeral ID and compare it to the stored location. This means that an ephemeral ID is only valid for recipients who are in the same location as the sender, and therefore relay attacks become useless. The encryption is done such that the MoH server, which received the epoch keys but not the ephemeral IDs, is not able to learn the location data. (Here, too, the challenge is supporting this property while limiting the size of the ephemeral IDs to 16 bytes.)

Note that in some cases, location information will not be available. The sender can indicate this in the ephemeral ID (under the same encryption and MAC). If for a specific ephemeral ID there is no location information for either the sender or the receiver, we do not preform the check for a relay. As long as a large majority of the users have location information, this mitigation will still be effective.

Committing to a device: We want to prevent adversaries from having multiple devices send the same ephemeral IDs. Therefore each copy of the application must generate at installation an ID commitment key, which must be unique for each device and affects all derivations of ephemeral IDs. When COVID-19 patients report their keys, the MoH server verifies that they reported different ID commitment keys. Two users who report the same ID commitment keys must raise a suspicion and require further checks.

4 Definitions

We follow the definitions (partly taken from from the Apple and Google solution):

- **Concatenation** We use the symbol || to denote concatenation.
- **HMAC** HMAC designates the HMAC function as defined by IETF RFC 2104, using the SHA-256 hash function:

$$\text{Output} = \text{HMAC}(\text{Key}, \text{Data})$$

Symbol	Meaning	Length (Bytes)
$AES(k, m)$	encryption	$\|k\| = \|m\| = \|AES(K, M)\| = 16$
K_{master}	master key	16
$K_{\text{DAYmaster}}$	day master key	16
$K_{\text{masterVER}}$	master verification key	16
K_{day}	day key	16
K_{epoch}	epoch key	16
K'_{epochENC}	epoch encryption key	16
K'_{epochMAC}	epoch MAC key	16
K'_{epochVER}	epoch verification key	16
K_{com}	identity commitment key	16
EphID	ephemeral ID	16

Table 1: Notation.

- **AES** AES designates the AES block cipher encryption using 128-bit keys as defined by

$$\text{Output} = \text{AES}(\text{Key}, \text{Data})$$

- **Truncation** Truncate defines a truncation function that returns the first L bytes of the data (the input data size being greater or equal to L is a precondition):

$$\text{Output} = \text{Truncate}(\text{Data}, \text{L})$$

- **CHAR** CHAR is ASCII-encoding of latin characters as a bytelist.

- **DayNumber** Provides a number for each 24-hour window. These time windows are based on Unix (Posix) time.

$$DayNumber = \frac{\text{Unix Time in Seconds}}{60 \cdot 60 \cdot 24}$$

  DayNumber is encoded as a 32-bit (`uint32_t`) unsigned little-endian value.

- **EpochNumber** Provides a number for each key epoch or T_E-minute window in a 24-hour window as defined by DayNumber. This value will be in the interval $[0, 24 \cdot 60/T_E]$

$$EpochNumber = \frac{\text{Unix Time in Seconds}\%60 \cdot 60 \cdot 24}{60 \cdot T_E}$$

  EpochNumber is encoded as a 8-bit (`uint8_t`) value. Note that EpochNumber should not be confused with the Unix Epoch.

- **TimeUnit** Provides a number for each T_U-minute window inside a key epoch. Each TimeUnit will have a different EphId. This value will be in the interval $[0, T_E/T_U]$

$$TimeUnit = \frac{\text{Unix Time in Seconds}\%60 \cdot T_E}{T_U}$$

  TimeUnit is encoded as a 8-bit (`uint8_t`) value.

5 Cryptographic Design

We use notation that is summarized in Table 1.

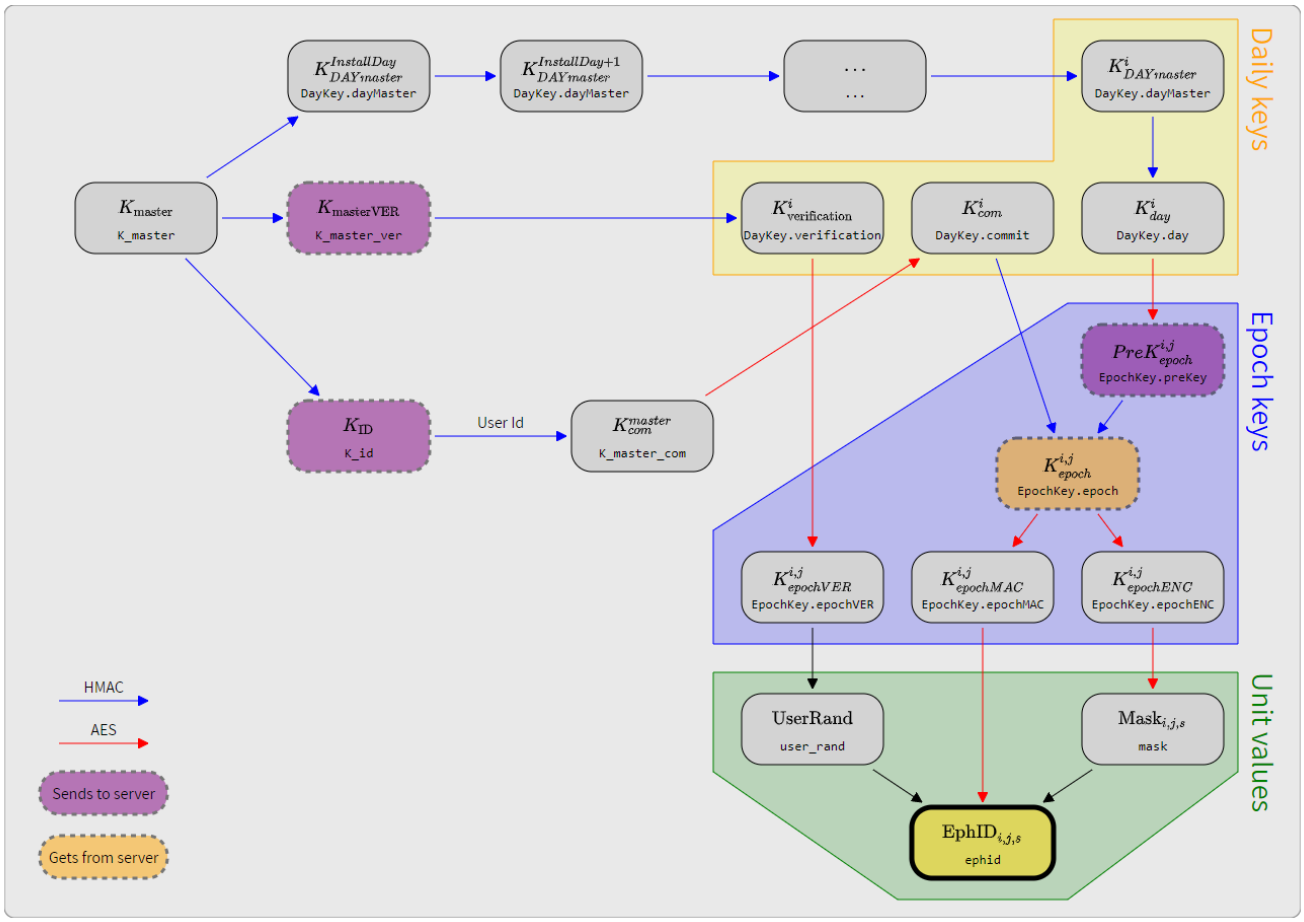

Figure 1: The key derivation mechanism.

The general structure of computing keys and values is by computing

$$\text{Truncate}(\text{HMAC}(\text{key}||\text{value}), \text{length})$$

where "key" is a key, "value" is an identifier which is guaranteed to be unique for all usage of the same key, and "length" is the byte length of the output, which is typically 16.

5.1 Key Derivation and Ephemeral Id

We divide the time to days, and each day to epochs of T_E minutes. We suggest setting $T_E = 60$ and therefore a day has 24 epochs. We further divide the time to units of T_U minutes, and suggest using $T_U = 5$. Each device will change the ephemeral ID that it sends every T_U minutes. (The reason for changing the ephemeral ID more frequently than changing the epoch is to prevent linkage between different ephemeral IDs that a user sends in the same epoch, unless the user is later diagnosed as COVID-19 positive.)

We use a tree-like key derivation scheme. The goal is to allow the server to broadcast separate keys for each epoch. This makes it much harder to link messages of the same infected person across different epochs. However, the scheme also allows the server to broadcast daily keys for reducing the total bandwidth.

We chose to derive all keys in two steps. First derive daily keys from the master key, and from these daily keys derive epoch keys. This allows the server to store the daily or epoch keys in a lexicographic order,

without information that allows to link contacts across different days or epochs. Note that from a privacy perspective it is always preferable to only store epoch keys. However, we support the option of storing and sending daily keys in case that the of number of infected persons is large and it is required to optimizing the bandwidth.

The following description describes the cryptographic keys, and the operation of the system. The key derivation is depicted in Figure 1.

Master key

When the application is installed on a new machine it generates a uniformly random new master key, K_{master}, which is 16 bytes long. This key will be used to generate all keys and messages sent by the application. It is never replaced, unless the user reports herself as a COVID patient. (In that case a new master key is generated, in order to preserve the privacy of the future contacts of the user. See details below.) The master key is deleted as soon as it is used to derive the initial keys, as is depicted in Figure 1.

Identity Commitment

The goal of the identity commitment is to bind a unique ID to the ephemeral IDs, in a privacy preserving way. This is done by adding a commitment to a specific identity, that can be proven to the central server but is not disclosed to other users.

When the application is installed it generates a random key K_{id} from the master key K_{master}.

$$K_{id} = \text{Truncate}(\text{HMAC}(\text{K}_{\text{master}}, \text{CHAR}(\text{“IdentityKey}'')), 16)$$

The application then derives the commitment value K_{com} from K_{id}.

$$K_{com}^{master} = \text{Truncate}(HMAC(K_{id}, \text{UserID}||\text{CHAR}(\text{“IdentityCommitment}'')), 16)$$

UserID may include a unique identifier such as phone number or national identity number, and optional other identifiers such as country, health provider etc.

If unique identifier is not available upon installation, UserID is set to a random number 16 byte number.

UserID is sent to the MoH as part of the infection testing process, and is used to bind the keys to a specific test.

Note that we assume that the identify of the infected person is known to the MoH. However, the public value K_{com} does not reveal any information on UserID.

Verification master key

The goal of the verification keys is to allow users to provide a “proof” that they received an ephemeral ID sent a COVID-19 positive person.
The application derives the master verification key $K_{\text{masterVER}}$ from K_{master}.

$$K_{\text{masterVER}} = \text{Truncate}(HMAC(K_{\text{master}}, \text{CHAR}(\text{“VerificationKey}'')), 16)$$

The ephemeral IDs sent by each user will contain information (several bytes) that is derived from the $K_{\text{masterVER}}$ key. As the verification keys of infected persons will not be broadcast, only users that have received an ephemeral ID can learn this information. They can later use this information to “prove” to the MoH that they indeed received an ephemeral ID sent from a COVID-19 positive person. These proofs are anonymous and do not reveal the identity of the prover, and therefore a user that received such a message can pass the verification bytes to other users.

Day master key

Each day uses a different master key. Knowledge of a master key of a day enables to compute the master keys of all future days. (This enables an infected person to send a very short message, containing only the day master key, that lets the ministry of health compute all keys of the following 14 days. This option is not implemented in the current version.) Day InstallDay is the DayNumber when the application is installed.

- The day master key of day InstallDay, $K^{\text{InstallDay}}_{\text{DAYmaster}}$, is generated as

$$K^{\text{InstallDay}}_{\text{DAYmaster}} = \text{Truncate}(\text{HMAC}(\text{K}_{\text{master}}, \text{CHAR}(\text{“DeriveMasterFirstKey”})), 16)$$

- The day master key of day i for $i >$ InstallDay, is generated as

$$K^{i}_{\text{DAYmaster}} = \text{Truncate}(\text{HMAC}(\text{K}^{\text{i}-1}_{\text{DAYmaster}}, \text{CHAR}(\text{“DeriveMasterKey”})), 16)$$

Day keys

The day key is generated from the day master key and is used for computing all values used in that day. The day key is separated from the day master key to allow sending only a subset of the day keys, while ensuring that they cannot link day keys used by the same infected person in different days. The application generates the day key for day i as:

$$K^{i}_{\text{day}} = \text{Truncate}(\text{HMAC}(\text{K}^{\text{i}}_{\text{DAYmaster}}, \text{CHAR}(\text{“DeriveDayKey”})), 16)$$

The day verification key is used to define part of the ephemeral IDs sent in BLE messages. It is later used to verify that users who claim that they were next to an infected person indeed received a message from that person. The day verification key is generated as

$$K^{i}_{\text{verification}} = \text{Truncate}(\text{HMAC}(\text{K}_{\text{masterVER}}, \text{i (4BYTES)} \,||\, \text{CHAR}(\text{“DeriveVerificationKey”})), 16)$$

We also derive a daily commitment key from the master commitment key K_{com}. Again, this is done to prevent linking keys of the same infected person across different days.

$$K^{i}_{com} = \text{AES}(\text{K}^{\text{master}}_{\text{com}},\ \text{i (4 BYTES)} \,||\, 0\ (12\ \text{BYTES})\ )$$

Epoch key

Each time epoch (e.g., hour) has a different epoch key. The application computes an epoch key, and from this key it derives an encryption key and a MAC key that will be used in the epoch. In addition, there is an epoch verification key which is used to define part of the ephemeral IDs sent in BLE messages. The key derivation for epoch j in day i is done in the following way (i is encoded as little endian `uint32_t` and j is encoded as `uint8_t`; the derived keys $K^{i,j}_{\text{epochENC}}$ and $K^{i,j}_{\text{epochMAC}}$ are each 16 bytes long):

$$PreK^{i,j}_{\text{epoch}} = \text{AES}(\text{K}^{\text{i}}_{\text{day}},\ \text{i (4 BYTES)} \,||\, \text{j (1 BYTES)} \,||\, 0\ (11\ \text{BYTES})\ ) \tag{1}$$

$$K^{i,j}_{\text{epoch}} = \text{Truncate}(\text{HMAC}(\text{PreK}^{\text{i,j}}_{\text{epoch}}, \text{K}^{\text{i}}_{\text{com}} \,||\, \text{i (4 BYTES)} \,||\, \text{j (1 BYTES)} \,||\, \text{CHAR}(\text{“DeriveEpoch”})), 16) \tag{2}$$

$$K^{i,j}_{\text{epochENC}} = \text{AES}(\text{K}^{\text{i,j}}_{\text{epoch}}, \text{i (4 BYTES)} \,||\, \text{j (1 BYTES)} \,||\, 0\ (11\ \text{BYTES})\ ) \tag{3}$$

$$K^{i,j}_{\text{epochMAC}} = \text{AES}(\text{K}^{\text{i,j}}_{\text{epoch}}, \text{i (4 BYTES)} \,||\, \text{j (1 BYTES)} \,||\, 1\ (1\ \text{BYTES}) \,||\, 0\ (10\ \text{BYTES})\ ) \tag{4}$$

$$K^{i,j}_{\text{epochVER}} = \text{AES}(\text{K}^{\text{i}}_{\text{verification}}, \text{i (4 BYTES)} \,||\, \text{j (1 BYTES)} \,||\, 0\ (11\ \text{BYTES})\ ) \tag{5}$$

Note that the identity commitment key K^{i}_{com} is used in the derivation of the epoch key. The output of HMAC is 32 bytes long and is divided to two 16 bytes keys.

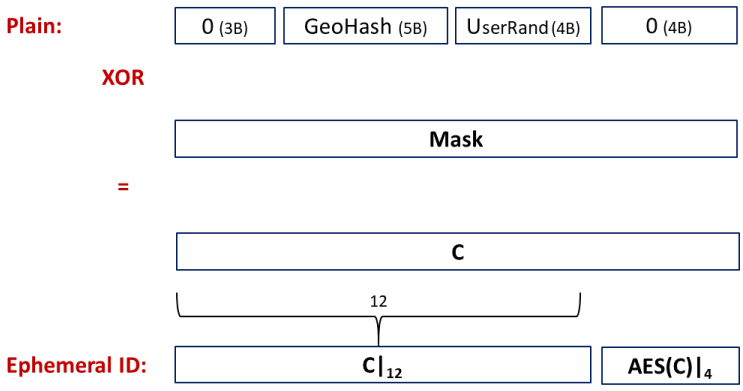

Figure 2: The derivation of the Ephemeral ID.

Design Intuition: The key $PreK^{i,j}_{\text{epoch}}$ is derived from the day key. The key $K^{i,j}_{\text{epoch}}$ is derived from $PreK^{i,j}_{\text{epoch}}$ using the key K^{i}_{com} which binds the keys to a unique ID value. On the other hand, we do not want to require the server to broadcast K^{i}_{com} to users since this value will enable them to connect values sent by an infected person in different epochs. Therefore we derive $K^{i,j}_{\text{epoch}}$ from $PreK^{i,j}_{\text{epoch}}$, and have the server only broadcast $K^{i,j}_{\text{epoch}}$ to users. The keys $K^{i,j}_{\text{epochENC}}$ and $K^{i,j}_{\text{epochMAC}}$ are derived from $K^{i,j}_{\text{epoch}}$. These keys will be used by users to verify that the BLE message they received corresponds to a infected person. The key $K^{i,j}_{\text{epochVER}}$ will be used to ensure that only users which receive the ephemeral ID can reconstruct a "proof" which depends on this key. This value verifies to the server that the user indeed received this BLE message from a COVID-19 positive person.

Ephemeral IDs

Let the number of time units in an epoch be $n = T_E/T_U$ (for our recommended settings $n = 60/5 = 12$). The epoch key is used for generating n ephemeral IDs.

We first list the rational for the design that we suggest:

- The ephemeral ID should include an authenticated encryption of the location of the transmitting device. This is required in order to prevent relay attacks that broadcast the ephemeral ID to users in other locations. The design must also prevent attacks where the relay changes the original ephemeral ID, for example by xoring values to the ciphertext in order to change the original encrypted location to another location.
- There is a constraint of sending only 16 bytes in the ephemeral ID.
- Another constraint is that the computation must be quick, since it must be run in real time (unlike the key derivations). Therefore we prefer using AES to using HMAC.
- Checking at the client side for physical contacts, should be run in time which is linear in the number of infected persons and the number of recorded beacons.

The computation is done in the following way, which is also described in Figure 2.

- The ephemeral ID for time unit s in epoch j of day i is a 16 byte value that is generated in the following method. The application sets a random 4 byte value, UserRand, as a truncation of the epoch verification key $K^{i,j}_{\text{epochVER}}$. It also computes a 5 byte GeoHash of its current location. It then runs the following computation.

$$\text{Mask}_{\text{i,j,s}} = \text{AES}(\text{K}^{\text{i,j}}_{\text{epochENC}}, \text{s})$$

$$\text{UserRand} = \text{Truncate}(\text{K}^{\text{i,j}}_{\text{epochVER}}, 4)$$

$$\text{Plain}_{\text{i,j,s}} = 0 \text{ (3 BYTES)} \,||\, \text{GeoHash (5 Bytes)} \,||\, \text{UserRand (4 Bytes)} \,||\, 0 \text{ (4 BYTES)}$$

$$\text{C}_{\text{i,j,s}} = \text{Plain}_{\text{i,j,s}} \oplus \text{Mask}_{\text{i,j,s}}$$

$$\text{EphID}_{\text{i,j,s}} = \text{Truncate}(\text{C}_{\text{i,j,s}}, 12) \,||\, \text{Truncate}(\text{AES}(\text{K}^{\text{i,j}}_{\text{epochMAC}}, \text{C}_{\text{i,j,s}}), 4)$$

  A note on UserRand: The UserRand field is used to enable the receiver of the ephemeral ID to prove that it received this message. Ideally, this message would be unique per ephemeral ID, for example computed as $\text{Truncate}(\text{AES}(\text{K}^{\text{i,j}}_{\text{epochVER}}, \text{s}), 4)$. However, since the ephemeral ID is generated in real time, we prefer to improve the latency of this operation and use the same UserRand value for all ephemeral IDs sent in an epoch. This will enable the receiver to prove that it received an ephemeral ID in this epoch, but not the exact time and number of ephemeral IDs received.

  MAC size: Due to communication constraints, our MAC is only 32-bits long. However, we assume that for deciding on exposure, the application will require at least two valid ephemeral IDs in a short time interval. And that means that in most cases, a relay attacker needs to forge two or more messages, and that will happen with a negligible probability of approximately 2^{-64}.

  A note on GPS information accuracy and availability: We propose sending only coarse geo-location information (e.g., accuracy of ± 500 meters). We believe this is sufficient to prevent most cases of relay attacks, without compromising the users' privacy (Bluetooth range is much less than 500 meters). As location information might not be available (e.g., a long tunnel, location is disabled by the user, etc.), the information encoding should allow notifying that no location information is available (e.g., all zero bytes).

  A note on the storing the GPS information: The GeoHash, which encodes the location, need not be kept secret from the receiver of the BLE message, since the receiver is physically close to the sender. Moreover, we assume that the phone may record its own GPS information. However, one can be worried that external access to the data stored on the device (for example by compromising the device, or using a court order), will reveal the location data. It would have therefore been preferable for the device to store a MAC of the geo-location, keyed by a key which is sent in the ephemeral ID and deleted afterwards. However, the current length of the ephemeral ID does not support a solution of this type.

Sending ephemeral IDs

The transmitting application sends over BLE communication the ephemeral ID corresponding to the current time unit.

Receiving ephemeral IDs

The receiving application receives over BLE communication an ephemeral ID corresponding to the current time unit. It stores the received ephemeral ID together with its own current location data (if available). The stored location data should be sufficient for computing the GeoHash of the current location and of adjacent locations.

For each time interval of T_U minutes, the application will limit the number of received ephemeral IDs it stores (e.g., at most 1000 messages). After that, the app will stop storing the ephemeral IDs until the next time interval. This is done to protect against malicious transmitters that try to drain the local storage of the user's phone.

"Snoozing" Transmission

The application will allow the user to "snooze" the sending of ephemeral IDs.

Keys Redaction

The application will allow the user to redact keys for specific epochs. After the deletion of the required keys only the the remaining subset of keys from the tree will be stored on the device.

Reporting to the Ministry of Health

When an application user learns that she carries the corona virus, the user is given the option to send to the MoH information that enables to compute the ephemeral IDs sent by application in the last 14 days.

The key derivation supports several options for uploading keys:

- Sending the day master key corresponding to 14 days ago, namely $K^{d-14}_{\text{DAYmaster}}$, where d is the current day, and also the master verification key $K_{\text{masterVER}}$. We discourage the use of this option.
- Sending the daily keys of the last 14 days, $K^{i-14}_{\text{day}}..K^{i}_{\text{day}}$.
- Sending all of the epoch keys that were used in the last 14 days. This enables the user to redact part of the history of ephemeral IDs sent from the user's device by deleting the corresponding epoch keys.
- A combination of daily and epoch keys.

There are some important properties to note:

- The decision whether to send the key is given to the user. The server does not otherwise know any information about the ephemeral IDs sent by the user. Furthermore, user can hide the fact that she is hiding something, by generating dummy keys. The server cannot verify the correctness of the pre-epoch keys reported by the user.
- There must be an external procedure to verify that the user indeed tested positive to the virus, and that only such users can report their master key in the application.
- Note that given $K^{d-14}_{\text{DAYmaster}}$ the server can compute the master day keys for all following days. From these keys it is possible to generate the day keys, and from these keys the epoch keys needed to identify an verify the infected person's ephemeral IDs.
- The master verification key enables to compute the daily verification keys, which enable to verify that a user that claims to have been exposed to an infected person indeed received such an ephemeral ID.
- After sending the keys to the server (and receiving an acknowledgment that this key was received) the client application is reset. It chooses a new master key, and generates all new keys from that key, as if this was the initial installation day. (This is an ideal description for the operation of the application. The developers might choose to wait with the reset until it is further verified that proper actions were taken.)
- Although there is a procedure for the client to send to the server information that reveals only the ephemeral IDs for part of the 14 day period (a subset of the keys in the tree). As this operation might be hard for many users, the key redaction can be done on the server side by the MoH. However, in order to encourage trust in the system it is also important to allow users who wish to do that with an option to redact part of their history.

Updates Sent from the Ministry of Health to Users

The server receives from the user the keys for the last 14 days, as well as the verification key and the identity commitment key. (If the system received only the key $K^{d-14}_{\text{DAYmaster}}$ or daily keys, it can can compute all the corresponding pre-epoch keys used in each of these days.

Important note about which keys to broadcast:

- At one extreme, the server broadcasts to all users only the day keys of all newly discovered infected persons. The local copy of the application on users devices can derive all epoch keys from these keys, and check if it received an ephemeral ID corresponding to any of these keys. This approach requires very little communication, but on the other hand is bad for privacy, since it enables to link all ephemeral IDs sent from an infected person on the same day.

- The other extreme is for the server to derive the epoch keys of all users for the last 14 days, and broadcast this information while permuting the order of the epoch keys of different users for the same epoch. This approach is good for privacy but requires high communication ($16 \cdot 24 \cdot 14 = 5374$ bytes user if epoch keys are changed every hour.)

  This approach also enables the server to send only a subset of the ephemeral keys of each user. This can help prevent false positives (e.g., driving in a traffic jam) or to protect the infected person's privacy (e.g., when she was at home).

We recommend using the second approach as long as the length of the update message fits in the communication limits (e.g. less than 5 MB). Afterwards, it is possible to replace, for some of the users, sending the ephemeral keys with sending the day keys. *We discourage sending any* $K^{i}_{\text{DAYmaster}}$ *keys.*

Computing the update message: For every infected person that sends a master key, and for every day, the MoH will make a decision whether to publish the epoch keys of the infected person for that day (or a subset of them), or to publish only the day key (which will save communication but will enable users to link different exposures to that user).

Let (id, i) denote a pair of a person and a day. The MoH defines a set S_E of (person,day) pairs for which epoch keys will be published, and a set S_D of (person,day) pairs for which day keys will be published. Each (person,day) pair appears in exactly one of these sets.

For each day, the server groups together all (person,day) pairs in S_D which correspond to that day, and publishes all the corresponding day keys in a lexicographic order, with a header identifying the day. (This enables the receiving user to identify the day, but not learn more than that.)

For each (epoch,day) time combination, the server groups together all (patient,day) pairs in S_E which correspond to that epoch, and publishes all the corresponding epoch keys in a lexicographic order, with a header identifying the epoch and day. (Grouping by epoch enables the receiving client to identify the epoch to which each key corresponds, and check it more efficiently.)

Communication cost: Denote the number of new infected persons as N. It holds that $S_E + S_D = 14N$. The communication cost in bytes is approximately $16 \cdot N \cdot 14 \cdot (24S_E + S_D)$. Suppose that the number of new infected persons is $N = 1000$, then sending only epoch keys requires sending $16 \cdot 24 \cdot 14 \cdot 1000 = 5.4$ MBytes, which might be too much. Sending only day keys will require 224 KBytes. Therefore with such a large number of users it might be preferable to send day keys for the minimal number of users required.

Redacting certain times: Suppose that the MoH was requested by an infected person not to publish (i.e., redact) contact data for some epochs. In that case the server should opt to sending epoch keys for that user, and not send the keys corresponding to the epochs which should be redacted.[2]

[2]There is some privacy leakage since users can count the number of published epoch keys for each epoch and identify that some epochs were redacted. If this is a concern then the server can publish dummy random keys instead of the redacted keys.

Checking for exposure

The client application stores internally all ephemeral IDs that it received in the last 14 days, together with the corresponding times and locations. We assume that this data is sorted by the reception time.

The client application only needs to compare the values that it received in a specific time to the ephemeral IDs that were sent in corresponding times, rather than to all ephemeral IDs. Recall that the system divides time to epochs and to smaller time units (corresponding in our example to one hour and 5 minutes, respectively). The application scans the ephemeral IDs that it received in sorted time order, and defines for each ephemeral ID the time units in which it could have been sent.

The exact correspondence between reception time and transmission time depends on system dependent parameters such as jitter and other issues. For example, a message that was received at the beginning of a time unit might have been sent in that time unit or in the preceding one. Let us denote the set of all these time units as the *potential transmission time units*. For each time unit in this set, the client application can identify the ephemeral IDs that it received and could have potentially been sent in this time unit.

The client application receives from the MoH sets of day keys and of epoch keys corresponding to new COVID-19 positive persons. It then scans the set of potential transmission time units. Let i, j correspond to a day and epoch that contain a potential transmission time unit. Let the set $S_{i,j}$ include all epoch keys that were reported as being used by COVID-19 positive persons in that epoch (these keys could have either been sent directly by the MoH or generated from the sent day keys).

For this epoch, the client performs the following operations:

- For each received epoch key $K^{i,j}_{\text{epoch}}$ in $S_{i,j}$, it generates the corresponding encryption and MAC keys $K^{i,j}_{\text{epochENC}}$ and $K^{i,j}_{\text{epochMAC}}$ using equations (3) and (4), and uses $K^{i,j}_{\text{epochENC}}$ to compute the corresponding $\text{Mask}_{i,j,s}$ values for all values of s ($s = 0, \ldots, 11$ in our example parameters) corresponding to potential transmission time units in this epoch.

- For each $\text{Mask}_{i,j,s}$ value, it checks if in the list of stored ephemeral IDs that it received in the corresponding reception time, there is a value $\text{EphID}_{i,j,s}$ whose first 3 bytes are equal to the first 3 bytes of $\text{Mask}_{i,j,s}$ (false negatives should happen with probability 2^{-24}).

- If a match occurs, the application check the part of EphID which corresponds to the MAC:

  - It sets x to the first 12 bytes of $\text{EphID}_{\text{i,j,s}}$, concatenated to the 4 last (least significant) bytes of $\text{Mask}_{\text{i,j,s}}$.
  - Sets y to be the last 4 bytes of $\text{EphID}_{\text{i,j,s}}$.
  - Checks if y is equal to the 4 last (least significant) bytes of $\text{AES}(\text{K}^{\text{i,j}}_{\text{epochMAC}}, \text{x})$.

- If this verification succeeds, the application recovers the first 12 bytes of $\text{Plain}_{\text{i,j,s}}$, as $\text{Truncate}(\text{EphID}_{\text{i,j,s}} \oplus \text{Mask}_{\text{i,j,s}}, 12)$, and checks if the next 5 bytes in the recovered value are equal to the GeoHash of the location that it stored for this $\text{EphID}_{i,j,s}$, or to the GeoHash of a location in an acceptable range (determined by the application).

  If no location information is available for either the sender or the receiver, the location check should be ignored. Note that the application must also support edge cases such as a user riding a fast train. If the location sampling rate is not high enough, the range should be extended using location from adjacent times.

- If the final verification succeeds, the application concludes that it received a valid ephemeral ID. In order to be able to prove the exposure to the server, the application recovers the UserRand field sent in the message, which is encoded in bytes 8 to 11 (starting from the most significant byte, and indexing the first byte as 0).

The application decides if the user should be notified as function of the number of matches, and other auxiliary data (such as RSSI records). If this decision is made then the application reports to the user that it identified an exposure in this specific epoch.

The application can offer the user the option to *voluntarily* report to the server the UserRand fields of the ephemeral IDs it received. The server can verify the physical contact to a COVID-19 infected person, by comparing these values to the UserRand values computed from the verification keys $K^i_{\text{verification}}$ for the corresponding epochs from all infected persons. This can be used to verify contact reports and prevent misuse (e.g., trying to get a payed sick leave) or to share data with epidemiologists. Note that this will **reveal to the server the fact that this user was in contact with that specific infected person**. If this information is considered private then the system should not support the usage of UserRand as a user dependent verification field.

Acknowledgment

We thank the many researchers with whom we discussed the design, including Manuel Barbosa, Eli Biham, Dan Boneh, Yehuda Lindell, Moni Naor, and Adi Shamir. We are very grateful for Ron Asherov and other contributors for their hard work on the Python reference implementation and helpful comments.