Mobile Networks and Applications
https://doi.org/10.1007/s11036-023-02136-4

Contact Tracing App Privacy: What Data is Shared by Non-GAEN Contact Tracing Apps

Douglas J. Leith[1]

Accepted: 13 May 2021

Abstract
We describe the data transmitted to backend servers by the contact tracing apps now deployed in France (TousAntiCovid), Australia (CovidSafe), Singapore (TraceTogether), US/Florida (CombatCovid), Israel (HaMagen), India (Aarogya Setu) with the aim of evaluating the privacy of these contact tracing apps as actually deployed. To the best of our knowledge, the measurements we present are the first public data of this sort for the French, Australian, Israeli, Indian and Florida apps. We find that TousAntiCovid and CovidSafe are generally well-behaved with regard to privacy. TraceTogether and CombatCovid make extensive use of Google Firebase services which means that that there are two main parties involved in handling data transmitted from these apsp, namely Google and the health authority operating the app itself. HaMagen is well-behaved for uninfected users of the app but GPS location data associated with infected is publicly published on the HaMagen server. Aarogya Setu, with >160M users, is found to have a number of serious privacy issues, including silent upload of logged location data.

Keywords Contact Tracing · Covid · Privacy

1 Introduction

Over the last year there has been a great deal of interest in the use of mobile apps to facilitate Covid-19 contact tracing. We report here on measurements of the actual data transmitted to backend servers by the contact tracing apps now deployed in France (TousAntiCovid) [1], Australia (CovidSafe) [2], Singapore (TraceTogether) [3], US/Florida (CombatCovid) [4], Israel (HaMagen) [5] and India (Aarogya Setu) [6] with a view to evaluating user privacy. Our aim is to examine the privacy of these contact tracing apps as actually deployed.

While many countries have now adopted the Google/Apple Exposure Notification (GAEN) system for contact tracing [7, 8], France, Australia, Singapore, Florida, Israel and India have developed their own alternative approaches. These countries have often been early adopters, e.g. Singapore in March 2020 was the first country to roll out a contact tracing app, with Israel (also March 2020), Australia (April 2020) and India (April 2020) close behind. With >160M users [6] the Indian app Aarogya Setu is probably the most widely used contact tracing app globally. To the best of our knowledge, the measurements we present are the first public data for the French, Australian, Florida, Israeli and Indian apps. We focus on the Android implementations of contact tracing apps. Of course the Apple implementations are also important, and we look forward to measurement studies of these.

The results of our study can be summarised as follows.

TousAntiCovid and CovidSafe The TousAntiCovid app appears exemplary from a privacy point of view, sharing minimal information with central servers and sharing no data with third parties such as Google. TousAntiCovid is open source and its operation publicly documented. CovidSafe is also generally well-behaved with regard to privacy, as well as being open-source. The app uses Google's Firebase Cloud Messaging service for push notifications, but in our tests we found that this makes few network connections. Since CovidSafe already makes regular connections to its own servers this dependency could be removed, which would improve privacy. The app requires the user to enter a valid Australian phone number. Since photo ID must be presented to obtain a SIM in Australia, a phone number is linked to a user's real

✉ Douglas J. Leith
doug.leith@tcd.ie

[1] School of Computer Science & Statistics, Trinity College Dublin, Dublin, Ireland

Published online: 05 September 2023

identity which creates an immediate privacy concern. Similarly to TousAntiCovid, this might be avoided by use of push notifications based on randomised identifiers.

TraceTogether and CombatCovid Both apps are both derived from the Open Trace app and backend server. Both apps make extensive use of Google Firebase services. This means that there are two main parties involved in handling data shared by each app, namely Google (who operate the Firebase service) and the health authority operating the app itself. As owner of Firebase, Google has access to all data transmitted by the app via Firebase but filters what data is made available to the health authority e.g. to present only aggregate statistics. Both apps also use Firebase Analytics to monitor app usage. Note that the Google Analytics documentation [9] states that "Analytics derives location data from users' IP addresses". Hence, the data sent by the handset potentially allows its location to be tracked over time. The Firebase Analytics documentation states that "Thresholds are applied to prevent anyone viewing a report from inferring the demographics, interests, or location of individual users" [9]. Assuming this is effective (note that the effectiveness of de-anonymisation methods is far from clear when applied to location data over time), then the health authority operating the app cannot infer individual user locations. The primary privacy concern therefore lies with the holding of rough location data by Google itself, especially as there is an obvious potential conflict of interest for Google whose primary business is advertising based on collection of user personal data. We recommend stopping use of Firebase, and in particular Firebase Analytics, to minimise data sharing with Google. TraceTogether and CombatCovid both require the user to enter a valid phone number and, similarly to CovidSafe, we recommend replacing this by use of push notifications based on randomised identifiers. While OpenTrace is open source, TraceTogether and CombatCovid are not. We could find no public technical documentation on the operation of CombatCovid, the observation that it is derived OpenTrace is based on our inspection of the decompiled code.

HaMagen GPS location data associated with infections is publicly published on the HaMagen server, raising obvious privacy concerns for infected users asked to upload their data to the server. We could not find any documentation describing mitigating measures (such as obfuscation/redaction of locations) taken by the HaMagen server prior to publishing this location data. On the plus side for privacy, for uninfected users of the app HaMagen shares no identifiers with its own servers or with third-party servers such as Google. We note, however, that HaMagen requires Google Play Services to be enabled in order to operate, which is known to result in substantial sharing of information on device activity with Google [10]. HaMagen is open source.

Aarogya Setu From a privacy perspective we found a number of serious issues with the Aarogya Setu app. Firstly, upon launch of the app the user's location is automatically shared with the central server, with no dialogue or popup raised notifying the user of this or asking for their consent. Secondly, and this is positively disturbing, the app silently uploads the bluetooth and GPS location data stored on the handset upon receipt of an instruction from the server. This happens without notifying the user or asking for their consent. Both of these are clearly problematic and we recommend that as a matter of urgency the app be updated to always require explicit user consent for uploads of such sensitive data. Thirdly, the bluetooth frames that the app broadcasts send a persistent identifier of the handset in the clear. This is in marked contrast to all of the other apps studied (and also all GAEN-based apps) which take care to frequently change the identifier that is broadcast so as to make it difficult for observers to use bluetooth transmissions to track users movements. Note that commercial providers are already seeking to build bluetooth sensor networks specifically targetting COVID-19 surveillance of this sort by embedding code within common apps [11, 12], and so concerns regarding such attacks are not just hypothetical. Fourth, the Aarogya Setu server can be queried for statistics on the number of infected and unwell people in a 500m vicinity of a specified location. This might be used, for example, to discover data about a hospital or business, a politician's house and so on. By intersecting data from nearby locations the precision could be reduced to below 500m. Access to this server API requires an authorization header but it is easy to extract this header from app network connection traces and use this to make requests, as we demonstrate. Based on our measurements the data returned is not obfuscated i.e. exact values are returned. Aarogya Setu requires Google Play Services to be enabled in order to operate which, as already noted, is known to result in substantial sharing of information on device activity with Google. The app also makes use of the Firebase Analytics, Remote Configuration and Cloud Messaging services. We recommend that these dependencies on Google be removed. Aarogya Setu is open source but otherwise we failed to find public technical documentation on app operation.

2 Related work

There are three main strands of work on privacy analysis of contact tracing apps. The first, and most prominent, is analysis of cryptographic schemes for managing the identifiers transmitted in Bluetooth frames for proximity detection, see for example DP3T [13] (used by the Google/Ap-

ple Exposure Notification system), ROBERT (ROBust and privacy-presERving proximity Tracing protocol) [14] (used by the French app) and BlueTrace [15] (used by OpenTrace and associated apps). The second strand is applying static code analysis tools to contact tracing apps, i.e. inspecting permissions requested, tracing ithe flow of identifiers within the code and so on, building on existing analysis tools developed for Android, see [16] and [17]. The third strand, to which the present work belongs, involves taking network measurements to evaluate the privacy of contact tracing apps as actually deployed. Previous work includes early analysis of the network traffic generated by the Singapore OpenTrace/TraceTogether app, see [18], and analysis of the network traffic generated by the GAEN-based apps now deployed across much of Europe, see [10]. The present work is complementary to [10] since that work studies GAEN-based apps whereas here we focus on non-GAEN apps. The present work substantially extends that in [10] to include measurements from the French TousAntiCovid, Australian CovidSafe, Indian Aarogya Setu, Israeli HaMagen and Florida CombatCovid apps plus updated measurements for the Singapore TraceTogether app that reflect recent software releases.

3 Threat model: What do we mean by privacy?

It is important to note that transmission of user data to backend servers is not intrinsically a privacy intrusion. For example, it can be useful to share details of the user device model/version and the locale/country of the device and this carries few privacy risks if this data is common to many users since the data itself cannot then be easily linked back to a specific user [19, 20]. Issues arise, however, when data can be tied to a specific user. One common way that this can happen is when an app generates a long randomised string when first installed/started and then transmits this alongside other data. The randomised string then acts as an identifier of the app instance (since no other apps share the same string value) and when the same identifier is used across multiple transmissions it allows these transmissions to be tied together across time. Linking a sequence of transmissions to an app instance does not explicitly reveal the user's real-world identity. However, the data can often be readily de-anonymised. One way that this can occur is if the app directly asks for user details (e.g. phone number, address). But it can also occur indirectly either when the user's location is explicitly shared or when it is implicitly shared using the fact that transmissions by an app always include the IP address of the user device (or more likely of an upstream NAT gateway). The IP address acts as a rough proxy for user location via existing geoIP services and many studies have shown that location data linked over time can be used to de-anonymise [21, 22].

4 Measurement setup

4.1 Viewing content of encrypted Web connections

All of the network connections we are interested in are encrypted. To inspect the content of a connection we route handset traffic via a WiFi access point (AP) that we control. We configure this AP to use mitmdump [23] as a proxy and adjust the firewall settings to redirect all WiFi traffic to mitmdump so that the proxying is transparent to the handset.

The immediate difficulty encountered when using this setup is that all modern apps carry out checks on the authenticity of server certificates received when starting a new connection and aborts the connection when these checks fail. To circumvent these checks we use a rooted phone and make twoi interventions. Firstly, we install the mitmproxy cert as a trusted root cert, which induces most system network libraries to pass basic cert checks. Secondly, for each contact tracing app we use Frida [24] to patch the app on the fly to replace custom certificate validation functions (typically identified by decompiling the apps and carrying out manual inspection) with dummy functions that always report validation checks as being passed.

4.2 Additional material: Connection data

The content of connections is summarised and annotated in the additional material.

4.3 Hardware and software used

Mobile handset: Google Pixel 2 running a fresh factory install of Android 10. Rooted using Magisk v19.1 and running Frida Server v12.5.2. Contact tracing app versions used: TousAntiCovid v2.1.8, TraceTogether v2.5.2, CovidSafe v1.14.0, CombatCovid v1.0.7, HaMagen v2.2.14, Aarogya Setu v1.4.1. Laptop: Apple Macbook running Frida 12.8.20 and mitmproxy v5.0.1. Using a USB ethernet adapter the laptop is connected to a cable modem and so to the internet and the laptop is configured to operate as a WiFi AP that routes wireless traffic over the wired connection via the mitmproxy listening on port 8080. The handset is also connected to the laptop over USB and this is used as a control channel (no data traffic is routed over this connection) to carry out dynamic patching using Frida. In the Developer Options screen on the handset the "Stay Awake" option is set on.

4.4 Test design

Test design is straightforward since the all of the apps support only a single flow of user interaction. Namely, in all of the apps there is an "onboarding" process that involves stepping through a sequence of screens until the main screen is reached. Once the main screen is arrived at this is displayed thereafter. Testing therefore consists of recording the data sent upon installation and startup of the app, followed by navigation through these screens until the main screen is reached. We distinguish between connectiions made on launch of the app, with no user interaction, and later connections made after clicking on the start/proceed/consent button that many apps display on their splash page. The data sent by the app when left idle at the main screen is also recorded. A number of apps also have symptom checking functions and we also left these untouched to avoid interfering with the live service.

Non-GAEN contact tracing apps are not constrained to use Google Play Services, although the app developers may still choose to create a dependency on Google Play Services. In our tests we therefore repeat measurements with and without Google Play Services enabled.

4.5 Finding identifiers in network connections

Potential identifiers in network connections were extracted by manual inspection. Basically any value present in network messages that stays the same across messages is flagged as a potential identifier. Some values are app instance identifiers and change between fresh installs of an app, and so to assess this we took measurements for several fresh app installs and looked for changes in the identifiers. Many of the apps studied here link identifiers to a phone number entered by the user. To evaluate this dependency we also took measurements with different phone numbers. As we will see, many of the values of interest are associated with Firebase. We therefore try to find more information on the nature of observed values from Google privacy policies and other public documents as well as by comparing them against known software and device identifiers e.g. Firebase ID's and the AndroidID of the handset.

5 Google firebase

Most of the contact tracing apps studied here make use of Google's Firebase service (the notable exception is TousAntiCovid). In particular, the services used are Firebase Crashlytics, Cloud Messaging, Authentication, Cloud Functions, Remote Configuration and Analytics (also referred to as Google Analytics).

To use Firebase, an app initially makes a connection to Google servers to register with Firebase. A typical connection[1] is:

POST https://firebaseinstallations.googleapis.com/v1/projects/covidsafe-prod/installations
"appId": "1:510731179958:android:73323bd27306bf626d5317",
"fid": "**d_CbabxsSGyWT7N8cJKZM-**",
Response:
"authToken": {
"expiresIn": "604800s",
"token": "eyJhbGciOiJFUzI1NiIsInR<...>"
},
"fid": ''**d_CbabxsSGyWT7N8cJKZM-**",
"refreshToken": ''2_−456Fwbbqe_WCn6Xt<...>"

Here the "fid" value is the FirebaseID, a unique identifier of the app instance (to reset it requires uninstall and reinstall of the app). The token value in the response is used to authenticate subsequent connections to Firebase servers. It is JWT-encoded and the token in the above connection decodes to:

"fid": "**d_CbabxsSGyWT7N8cJKZM-**",
"exp": 1608525081,
"appId": "1:510731179958:android:73323bd27306<...>"

from which it can be seen that it includes the FirebaseIF and so acts as a unique identifier of the app instance. The refreshToken value is used to authenticate periodic requests for fresh tokens. The "appId" value identifies the app, e.g the value shown is for TraceTogether, and is the same for all instances of the app.

When Google Play Services is enabled, registration with Firebase generates a second connection to https://android.clients.google.com/c2dm/register3:

POST https://android.clients.google.com/c2dm/register3
Headers:
Authorization: AidLogin **3876027569814251330**:4707843177977026675
Body:
X−appid: **d_CbabxsSGyWT7N8cJKZM-**
X−Goog−Firebase−Installations−Auth: eyJhbGciOiJFUzI1NiIsInR<...>
device: **3876027569814251330**

The "X-Goog-Firebase-Installations-Auth" value is the Firebase token and so linked to the FirebaseID. The "device" value (also echoed in the "Authorization: AidLogin" header) is the AndroidID, a long-lived device identifier that requires a factory reset to change. This connection therefore shares the AndroidID with Google together with the FirebaseID, thereby linking the two. This means that while the FirebaseId changes upon re-install of the app, once linked with the AndroidID then the FirebaseID is effectively converted into a strong, long-lived identifier.

[1] Taken from TraceTogether, but the connections made by the other apps studied are of the same form.

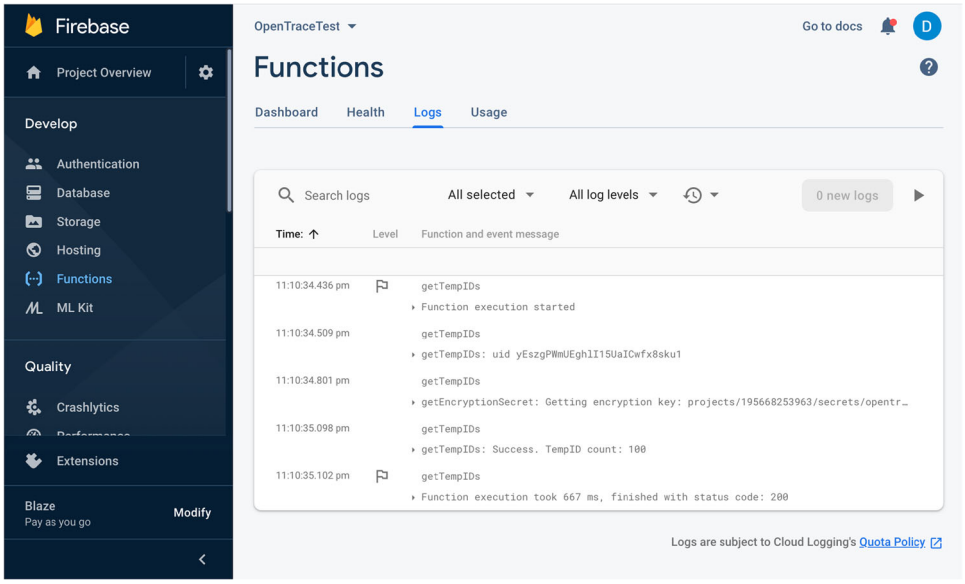

Fig. 1 Example of Firebase Functions logging visible to the operator of the OpenTrace app. Observe that there is fine-grained logging of individual function calls per user (the uid value in these logs is a unique identifier linked to a users phone number). The tempIDs function is , for example, regularly called by the OpenTrace app to refresh the set of tempIDs available for a mobile handset to advertise on Bluetooth

Subsequent Firebase connections are linked together by use of the authentication token (and so also linked to the FirebaseID/AndroidID), and share data with Google servers. When Firebase services are used this means there are at least two parties involved in handling data shared by the app, namely Google (who operate the Firebase service infrastructure) and the health authority operating the contact tracing app. As owner of Firebase, Google has access to all data transmitted by the app via Firebase but filters what data is made available to the operator of the app e.g. to present only aggregate statistics [9].

TraceTogether and CombatCovid, in particular, make heavy use of Firebase services. Both are based on the OpenTrace app and server backend, which is open source. The Firebase Authentication service is used on startup of the app to record the phone number entered by the user and verify it by texting a code which the user then enters into the app. The phone numbers entered are recorded by Firebase and linked to a Firebase identifier. These two apps also use Firebase Cloud Functions to generate tempIDs for broadcast over bluetooth and for upload of logged tempIDs upon the user becoming infected with Covid-19. The tempIDs are generated by reversible encryption using a key stored in Google Cloud's Secret Manager service and accessed by the getTempIDs function hosted on Firebase Cloud Functions. Figure 1 shows an example of the Firebase Functions logging visible to the operator of OpenTrace (and so presumably TraceTogether and CombatCovid). This fine-grained logging data shows individual function calls together with the time and user making the call (the uid value is the user identfier used by Firebase Authentication and so can be directly linked to the user's phone number). The Firebase privacy documentation states that the Firebase Authentication service always processes its data in US data centres and non-US app's may prefer to avoid this.

The Firebase privacy documentation[2] outlines some of the information that is exchanged with Google during operation of the API. This privacy documentation notes that Firebase Authentication logs user phone numbers and IP addresses. Also that Firebase Analytics makes use of a number of identifiers including: (i) a user-resettable Mobile ad ID to "allow developers and marketers to track activity for advertising purposes. They're also used to enhance serving and targeting capabilities."[3], (ii) an Android ID which is "a 64-bit number (expressed as a hexadecimal string), unique to each combination of app-signing key, user, and device",[4] (iii) a InstanceID that "provides a unique identifier for each app instance" and "Once an Instance ID is generated, the library periodically sends information about the application and the device where it's running to the Firebase backend."[5] and (iv) an Analytics App Instance ID that is "used to compute user metrics throughout Analytics". The Firebase Analytics documentation[6] states that "As long as you use the Firebase SDK, you don't need to write any additional code to collect a number of user properties automatically", including Age, Gender, Interests, Language,Country plus a variety of device information. It also states that "Analytics derives location data from users' IP addresses".

[2] https://firebase.google.com/support/privacy

[3] https://support.google.com/admanager/answer/6274238

[4] https://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID

[5] https://firebase.google.com/docs/reference/android/com/google/firebase/iid/FirebaseInstanceId

[6] https://support.google.com/firebase/answer/6317486

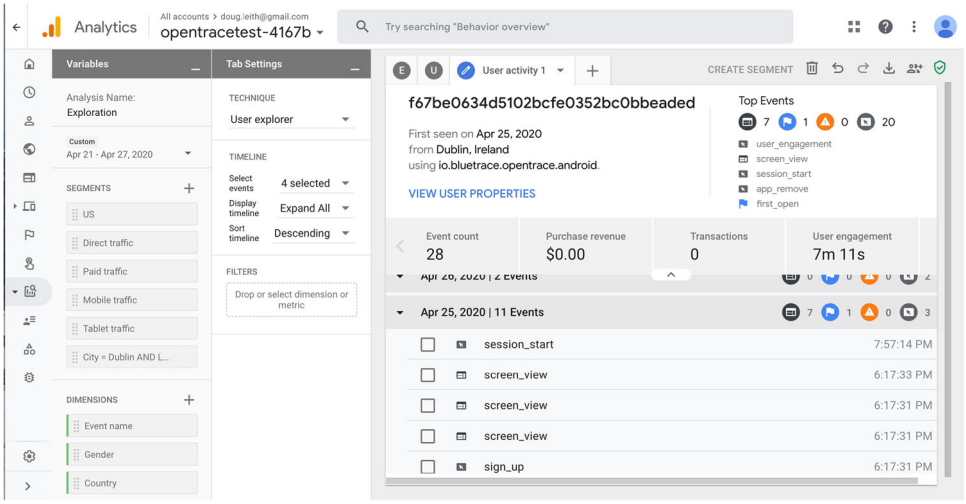

Fig. 2 Example of Firebase Analytics data visible to the operator of the OpenTrace app. Observe that data is available on events occurinng per individual device

Figure 2 shows an example of the data made available to the operator of the OpenTrace app by Firebase Analytics. It can be seen that per device event data is available showing for example when OpenTrace is started on the device, when it is viewed etc.

The data collected by Google during operation of its Firebase services need not be stored in the same country as the user of an app is located. The Firebase privacy documentation states that "Unless a service or feature offers data location selection, Firebase may process and store your data anywhere Google or its agents maintain facilities". It also states "A few Firebase services are only run from US data centers. As a result, these services process data exclusively in the United States" and it appears that these services include Firebase Authentication, which OpenTrace uses to stores user phone numbers.

It is important to note that only a filtered version of this data collected by Google is made available to users of its backend Firebase services. The Firebase Analytics documentation states that "Thresholds are applied to prevent anyone viewing a report from inferring the demographics, interests, or location of individual users" [9].

6 Google play services

Google Play Services is closed-source propietary software with only rather limited documentation on its operation and privacy. Android is open-source and devices running Android do not require Google Play Services to be installed or enabled in order to operate (although the great majority of Android handsets outside of China come pre-installed with Google Play Services). Recent measurement studies of contact tracing apps based on the Google/Apple Exposure Notification (GAEN) system have highlighted the extensive data sharing with Google that takes place when Google Play Services is enabled on a handset, even when the handset is configured so as to minimise data sharing [10]. This includes sharing of long-lived device identifiers including the IMEI, SIM serial number, hardware serial number, WiFi MAC address, user email address and of course the phone number, see [10] for details. On Android users of GAEN-based contact tracing apps are obliged to enable Google Play Services since the GAEN system itself is implemented within Google Play Services, and this raises concerns regarding privacy for those users who did not previously have Google Play Services enabled.

Non-GAEN contact tracing apps are not constrained to use Google Play Services and our test protocol therefore evaluates whether app developers have chosen to introduce a dependency on Google Play Services.

As noted above, enabling Google Play Services converts the FirebaseID into a strong, long-lived identifier. Google Play Services also independently collects its own telemetry on app operation, including contact tracing app operation, and transmits this to Google servers. In particular, for most of the apps that we studied we observed that Google Play Services, when enabled, shares telemetry with Google server using requests of the following form:

POST https://play.googleapis.com/log/batch
Headers:
authorization: Bearer ya29.m.EgwIARIEENfHARjHqw<...>
user−agent: com.google.android.gms/204516028 <...>
cookie: NID=204=MuyLPNP6QaT4<...>
Body:
<...>
\x08\x01\x10\x01\x18\x00"\x19
fr.gouv.android.stopcovid\xa0\x01\x01\xea\x01\x06∗
\x0410.0\xb2\x03\x02\x08\x01\xe2\x03\x06\x08\x01\x10\x01\x18\
x01\xfa\x04\x02\x08\x01X x\x00\x88
<...>

The "authorization: Bearer" header token links the request to the many strong, persistent user and device identifiers shared with Google by Google Play Services, observe

also the cookie. Note that the TousAntiCovid app (which has identifier fr.gouv.android.stopcovid on Android) has no dependencies on Firebase or other Google services, and so this telemetry collection is happening independently of the app configuration. The body of the request uses a proprietary protobuf format for which there is no public documentation, so we do not know the nature of the data shared with Google.

7 Data transmitted by contact tracing apps

In this section we summarise the data shared by the apps, see the Additional Material for more details.

7.1 TousAntiCovid

The TousAntiCovid app uses the ROBERT protoco [14]. On first use the app sends a unique identifier (clientPublicECDHKey) of the app instance to the central server. The server uses this identifier to generate a sequence of ephemeral bluetooth identifers (EBIDs) that are then sent to the app. An EBID is included each bluetooth transmission made by the app and logged by other devices running the app that observe the transmissions, together with the bluetooth received signal strength (RSSI). The EBID value transmitted changes every 15 minutes to mitigate tracking of transmissions by the same device over time. Upon being discovered to be infected with COVID-19 an app user uploads the EBIDs logged by their phone. The server uses a private key to decrypt these EBIDs and recover the corresponding clientPublicECDHKey values, which can then be used by contact tracers to identify handsets that may have been in proximity to the infected person. Each copy of the app periodically contacts the server to download fresh EBIDs and at this point the server can also push a message notifying the user that they may have been near an infected person.

The app has no dependence on Google Play Services, and we confirmed that it can be used with Google Play Services disabled. It also does not share the handset phone number with the server. The app is open source, https://gitlab.inria.fr/stopcovid19/stopcovid-android.

7.1.1 Data sent on initial startup

When launched, TousAntiCovid makes 12 requests to https://app.stopcovid.gouv.fr which download app UI components and covid statistics data. The requests themselves transmit no identifiers. Upon clicking "I'm in" the app fetches a captcha image. When the user enters the four letter code shown in the image the app sends that code to https://app.stopcovid.gouv.fr together with the image id for verification. In this request the app also sends the clientPublicECDHKey value that is a unique identifier of the app instance. The server responds with a sequence of encoded EBID values.

7.1.2 Data sent when sitting idle at main screen

The app periodically contacts https://app.stopcovid.gouv.fr to download fresh EBIDs and check for messages. An example exchange is:

POST https://api.stopcovid.gouv.fr/api/v3/status

"ebid": "**thg266WE9B4=**",
"epochId": 18551,
"mac": "J49modR+Azle9gd2M3vHPZf/<...>Fo=",
"time": "432GyQ=="

Response:

"atRisk": false, "config": [], "lastExposureTimeframe": 0, "message": null, "tuples": "GO8TSqsdCNhSQln0oc<...>

Since the EBID value can be decrypted by the server to recover the clientPublicECDHKey it acts as a persistent identifier of the app instance. The "mac" value is a hash to prevent spoofing. The tuples value in the response encodes EBID values.

7.2 TraceTogether, CombatCovid

We treat these two apps together since they are both closely based on the OpenTrace app.[7] When first launched both of these apps require the user to enter a valid phone number, which is then uploaded to a central server. CombatCovidPBC restricts this to a US number but TraceTogether does not impose such a geo restriction. An SMS message containing a one-time password/PIN is sent to the number and must be entered to proceed with use of the app. In addition TraceTogether requires a user to state whether they are "Visiting Singapore" etc and depending on the answer to enter further identification details e.g. a visitor is asked to enter their name, date of birth, nationality and passport number. The values entered do not seem to be validated (we entered dummy values to check).

The apps use the BlueTrace bluetooth protocol [15]. The app uploads the phone number to the server, which uses this to generate tempIDs that are then sent back to the app. These act similarly to the EBIDs of TousAntiCovid: they are included in bluetooth transmissions made by the app, logged by other devices running the app that observe the transmissions, together with the RSSI, and values logged by infected users are decrypted by the server to recover the phone numbers of people who may have been in proximity to the infected user.

While OpenTrace is open source, the derived TraceTogether and CombatCovid apps are not. Inspection of the decompiled app code indicates differences between TraceTogether/CombatCovid and OpenTrace but that the bluetooth

[7] https://github.com/opentrace-community/opentrace-android.

protocol remains similar. The device identifiers transmitted in the bluetooth frames are changed every 15 minutes to mitigate linking attacks.

Tracetogether and CombatCovid both use Firebase Analytics (also known as Google Analytics), Firebase Remote Configuration, Firebase Cloud Functions, Firebase Authentication, Firebase Cloud Messaging. In addition, TraceTogether uses Firebase Crashlytics.

While TraceTogether makes extensive use of Firebase it does not critically depend on Google Play Services to run: we confirmed that it can be launched and used with Google Play Services disabled, although push notifications via Firebase Cloud Messaging will likely fail when Google Play Services is disabled.[8] When run with Google Play Services enabled then we observed that additional telemetry data is shared with Google, see below. CombatCovid requires Google Play Services to be enabled in order to pass the phone number verification page in the app (when Google Play Services is disabled the app hangs indefinitely at this page). However, once onboarding is complete we found that the app could be used with Google Play Services disabled.

7.2.1 Data sent on initial startup

When launched, both apps make a connection to Google servers to register with Firebase.

TraceTogether In addition to registering with Firebase on first launch, TraceTogether makes connections to https://firebase-settings.crashlytics.com and to https://app-measurement.com to register the app instance with, respectively, Crashlytics and Google Analytics. It also sends telemetry to https://app-measurement.com. Clicking the "Open Message" button and following the onboarding process generates further requests to https://firebaseremoteconfig.googleapis.com, https://asia-east2-govtech-tracer.cloudfunctions.net, www.googleapis.com/identitytoolkit, https://app-measurement.com which access, respectively, Firebase Remote Configuration, Firebase Cloud Functions, Firebase Authentication and Google Analytics. The Firebase authentication token is sent with the request to https://firebaseremoteconfig.googleapis.com, linking it to the FirebaseId. For the other requests a new JWT-encoded token is generated and sent along with the requests. An example of the decoded token contents is:


```
  "aud": "https://identitytoolkit.googleapis.com/google.identity.
identitytoolkit.v1.IdentityToolkit",
  "iat": 1607920385,
  "exp": 1607923985,
  "iss": "govtech−tracer@appspot.gserviceaccount.com",
  "sub": "govtech−tracer@appspot.gserviceaccount.com",
  "uid": "aVPkeB9TYLW3h0xA2VuMfl5DIV92"
```


Here the uid value is linked to the phone number and so acts as a long-lived identifier of the device (breaking the connection between the uid and the device requires the device phone number/sim to be changed and the app re-installed).

CombatCovid In addition to registering with Firebase on first launch, CombatCovid makes connections to https://app-measurement.com to share telemetry with Google Analytics. During the subsequent onboarding process the app makes 15 connections to https://combatcovidapp.com which download app UI components and covid statistics data. These requests transmit no identifiers. However, the app also makes connections to www.googletagmanager.com and www.google-analytics.com to download Google Analytics javascript. This javascript appears to be embedded in the Privacy Policy and Terms of Use page of the app and used to send telemetry for that page to www.google-analytics.com/collect. Similarly to TraceTogether the CombatCovid app connects to https://firebaseremoteconfig.googleapis.com, https://us-central1-combatcovid-2d07d.cloudfunctions.net, www.googleapis.com/identitytoolkit in order to use Firebase Remote Configuration, Firebase Cloud Functions and Firebase Authentication. The app also sends telemetry on the onboarding process to https://app-measurement.com and makes a request to https://combatcovidapp.com/combatcovid-pbc-links which sets two Google Analytics cookies and sends telemetry to www.google-analytics.com/collect.

7.2.2 Data sent when sitting idle at main screen

When idle both apps make periodic requests for fresh tempIDs. These requests contain authentication tokens linking them to the phone number. TraceTogether and CombatCovid also make intermittent connections to https://app-measurement.com to share telemetry with Google Analytics.

7.3 CovidSafe

CovidSafe is also based on OpenTrace but appears to have been substantially modified. Unlike TraceTogether/CombatCobvid, CovidSafe is open source, https://github.com/AU-COVIDSafe/mobile-android. CovidSafe uses a variant of the BlueTrace bluetooth protocol [15] modified to use a different cryptographic scheme for the bluetooth payload.[9] The device identifiers (tempIDs) transmitted in the bluetooth frames are changed every 7.5 minutes to mitigate linking attacks. When first launched CovidSafe requires the user to enter a valid

[8] https://firebase.google.com/docs/android/android-play-services.

[9] https://dta-www-drupal-20180130215411153400000001.s3.ap-southeast-2.amazonaws.com/s3fs-public/files/COVIDSafe%20cryptography%20specification%20(with%20protocol%20version%20numbering)_v3.pdf

Australian phone number, which is then uploaded to a central server. The server uses this to generate tempIDs that are then sent back to the app. With the notable exception of an initial request registering with Firebase, the app does not contact Google servers and instead requests are directed to https://device-api.prod.lp.aws.covidsafe.gov.au, see below. CovidTrace does not depend on Google Play Services to run.

7.3.1 Data sent on initial startup

Similarly to TraceTogether/CombatCovid, when launched CovidSafe makes a connection to Google servers to register with Firebase and when Google Play Services is enabled a second connection is also made to https://android.clients.google.com/c2dm/register3 to register with Firebase. Inspection of the code indicates that the app makes use of Firebase Cloud Messaging and that these connections are associated with initialisation of that service. However, we observed no further connections to Google servers in our tests. Since the app seems able to fetch push messages via requests to https://device-api.prod.lp.aws.covidsafe.gov.au additional use of Firebase Cloud Messaging seems, on the face of it, unnecessary.

Clicking on the "I want to help" and following the onboarding process then generates a sequence of connections to https://device-api.prod.lp.aws.covidsafe.gov.au. The first connection uploads the phone number and the second the PIN code sent by SMS. The response to this second request contains a JWT-encoded token which decodes, for example, to:

```
"iat": 1607750013,
"exp": 1639307613,
"aud": "COVIDsafe",
"sub": "360a4fab−d5b6−4c0e−a269−8e9e5983d48b"
```

The sub value appears to be a unique identifier linked to the phone number (it changes when the phone number used is changed, but stays the same when the app is re-installed but the same phone number is used). This token is sent along with all subsequent connections to https://device-api.prod.lp.aws.covidsafe.gov.au, thereby linking all of these together and to the phone number.

Unlike TraceTogether/CombatCovid, CovidSafe was not observed to use Google Analytics.

7.3.2 Data sent when sitting idle at main screen

When idle CovidSafe makes periodic requests to https://device-api.prod.lp.aws.covidsafe.gov.au/prod/getTempId to fetch fresh tempIDs. These requests contain an authentication token linking them to the phone number.

7.4 HaMagen

There appears to be little public technical documentation on the operation of the HaMagen app. However, the app is open source[10] and its operation can (somewhat painfully) be inferred from inspection of the source code. The app uses two separate proximity detection approaches.

Firstly, the GPS location of the handset running the app is logged over time, with nearby locations (within 20m) aggregated and stored as a single database entry. The react-native-background-geolocation library[11] is used to obtain the GPS location. This data is stored locally on the phone. When a person is discovered to be infected the logged data is uploaded to a server. It is not clear how this data is processed (e.g. whether locations are obfuscated to enhance privacy), but in due course the server publishes a list of GPS locations associated with infections, together with the date/time and duration at each location. The app periodically fetches this list from the server and intersects the locations on the list with the locally stored locations. The user is notified of matches.

Secondly, HaMagen broadcasts bluetooth beacons containing ephemeral identifiers while also logging any beacons observed from other devices. The cryptographic scheme used for the ephemeral identifiers appears to be documented at https://github.com/eyalr0/HashomerCryptoRef/blob/master/documents/hashomer.pdf. As well as logging the ephemeral identifier in observed bluetooth beacons and the RSSI (similarly to other apps) HaMagen also stores the GPS location where the observation took place. This data is stored on the phone. The ephemeral identifier transmitted in a bluetooth frame changes every 5 minutes to mitigate tracking of transmissions by the same device over time. The app periodically downloads a list of keys from the server that can be used to reconstruct the ephemeral id's associated with infected people, compares these with the bluetooth data logged on the phone and notifies the user of matches. This appears to be a decentralised approach, similar to DP3T and Google/Apple Exposure Notifications but using a different cryptographic approach. In our tests the downloaded list of bluetooth keys was always empty and so it may be that this functionality is not being actively used.

We note that access to the main HaMagen server https://gisweb.azureedge.net appears to be geo-restricted. In our tests we therefore used a VPN so that our handset traffic appeared to be from a location in Israel.

When HaMagen is run with Google Play Services disabled it repeatedly raises popups asking for Google Play Services to be enabled (saying "won't work unless you enable Google

[10] https://github.com/MohGovIL/hamagen-react-native and https://github.com/MohGovIL/rn-contact-tracing.

[11] https://github.com/transistorsoft/react-native-background-geolocation.

Play Services"). From inspection of the app code it seems that these popups are generated by the embedded react-native-background-geolocation library, which relies on Google Play Services to access devioce motion sensors for battery saving. Its therefore appears that Google Play Services is required to be enabled when using HaMagen.

In summary, apart from an initial connection to Firebase, HaMagen shares no identifiers with servers. However, unlike TousAntiCovid, TraceTogether and CovidSafe, HaMagen requires Google Play Services to be enabled, which is known to result in substantial sharing of information on device activity with Google. From the point of view of uninfected app users who are already using Google Play Services HaMagen is arguably amongst the most private of the apps studied here. However, for users who have Google Play Services disabled, the requirement to enable it in order to use HaMagen results in a substantial loss of privacy. From the point of view of infected users uploading data to HaMagen servers the GPS location data associated with infections is publicly published by the HaMagen server. On the face of it this creates an obvious privacy risk since location time histories can often be relatively easily de-anonymised. We could not find any documentation describing mitigating measures (such as obfuscation/redaction of locations) taken by the HaMagen server prior to publishing this location data.

7.4.1 Data sent on initial startup

When first launched HaMagen makes a connection to Google servers to register with Firebase and a second connection is also made to https://android.clients.google.com/c2dm/register3 to register with Firebase. Telemetry data is sent to https://app-measurement.com. The app also makes 4 requests to https://gisweb.azureedge.net to fetch config information but these requests contain no identifiers.

Note that in our tests we saw no further connections to Google servers by the app. Inspection of the source code indicates that the app uses Firebase Cloud Messaging for push notifications. Similarly to CovidSafe, we comment that this data sharing with Google might be avoided by checking for messages during the periodic connections that the app already makes to https://gisweb.azureedge.net.

Upon clicking "Start" and proceeding with onboarding the app repeats (four times, so five times in total) the earlier fetches of config information from https://gisweb.azureedge.net. It also makes requests to https://gisweb.azureedge.net/BleUtc.json.sign and https://gisweb.azureedge.net/PointsUtc.json.sign to fetch, respectively, the lists of bluetooth keys and GPS locations associated with infections. No identifiers are sent with any of these requests.

The lists of bluetooth keys and GPS locations are publicly visible within Israel (the servers appear to be geo-restricted). As noted above, in our tests the downloaded list of bluetooth keys was always empty. An example of a request to fetch GPS locations is:

GET https://gisweb.azureedge.net/PointsUtc.json.sign?r=0.3700706993999472

Response consists of a list of entries of the following form:

<...>
{"type":"Feature","id":1,"geometry":{"type":"Point","coordinates"
:[5.6843418860808e−14,5.6843418860808e−14]},"properties":{"
OBJECTID":24024957,"ID":0,"Name":"","Place":"\xd7\xaa\xd7<...>
","Date":1607183070000,"types":"","Comments":"","POINT_X"
:5.6843418860808e−14,"
POINT_Y":5.6843418860808e−14,"fromTime":1607000460000,"
toTime":1607002260000,"sourceOID":1,"flight":0,"flightFrom":0,"
flightArrival":0,"stayTimes":"\xd7\x9e 2020−12−03 12:01 \xd7\xa2\
xd7\x93 2020−12−0312:31","fromTime_utc":1606989660000,"
toTime_utc":1606991460000,"Key_Field":24024957,"radius":0,"
valid":1,"
address":"","geohash":"s00000000000","geohashFilter":"s00000"}}
<...>

Observe that the GPS data includes not only a location and a time interval but also an OBJECTID value (echoed in the Key_Field) that can link multiple entries, and also binary Place data.

7.4.2 Data sent when sitting idle at main screen

When idle HaMagen periodically fetches config information and the lists of bluetooth keys and GPS locations from https://gisweb.azureedge.net. No identifiers are sent with these requests.

7.5 Aarogya Setu

According to the stats displayed in the app UI the Aarogya Setu app has 16.7 crore users, i.e. 167M users, and that is also consistent with the Google Play Store which states that the app has had > 100M downloads. Aarogya Setu is therefore probably the most widely downloaded contact tracing app globally. Unfortunately, there is a notable lack of technical documentation on the operation of Aarogya Setu. However, the app is open source at https://github.com/nic-delhi/AarogyaSetu_Android and https://github.com/tachyons/aarogyasetubackend, and at the cost of some effort we can infer operation details from that. Note that we decompiled the app apk and compared the code with the source code since some comments in github suggested that the published source code may be stale. Although we indeed observed differences between the decompiled code and the source code (e.g. in file FcmMessagingService.java), they appeared to be minor in nature.

When first launched Aarogya Setu registers with Firebase and then requires the user to enter a valid Indian phone number, which is then uploaded to a server at https://api.swaraksha.gov.in. An SMS message containing a one-time password/PIN is sent to the number and must be entered to

proceed with use of the app. The phone number and PIN are uploaded to the server, which responds with an authentication token. This token is therefore linked to the phone number. A second request to https://fp.swaraksha.gov.in sends both this token and a Firebase authentication token together, thereby linking the two. The response is a DiD value. Owing to the process used, this DiD value is linked to the phone number entered, the FirebaseID and the AndroidID. It therefore acts as a strong, long-lived device identifier (breaking the connection between the DiD and the device requires a factory reset, the device phone number/sim to be changed and the app re-installed).

The app transmits bluetooth beacons that contain the DiD value. Unlike the other apps analysed here, which frequently change the identifier broadcast in the bluetooth beacons, with Aarogya Setu this value is fixed and does not change over time. This means that it is easy to link beacons transmitted by the same device and so potentially reconstruct the movements of the device, provided a suitable network of bluetooth sensors is available. Commercial providers are already seeking to build bluetooth sensor networks specifically targetting COVID-19 surveillance by embedding code within common apps [11, 12] and so concerns regarding linking attacks are not just hypothetical.

The app also listens for beacons and for observed beacons records the beacon DiD, RSSI and the GPS location where the observation took place. This data is stored locally on the phone. It can be uploaded to a server via three separate mechanisms. Firstly, the user can click on a button within the app. Secondly, a push notification can be remotely sent to the app commanding it to upload the data – this upload can occur silently, without notifying the user or requesting consent.[12] Thirdly, when the app is launched it makes a request to https://fp.swaraksha.gov.in/api/v1/users/status (see below). The response to this request can instruct the app to upload the stored bluetooth/location data, again without notifying the user or requesting consent.[13] An example of the type of data uploaded is:

"dl": [{"d": "0f150ec4","dist": −78,"tx_level": "−2147483648","
tx_power": "127"}],
"l": {"lat": "53.3<...>","lon": "−6.2<...>"},"ts": "1608483030"

Here the "dl" value is Bluetooth data (the observed DiD identifier 0f150ec4 and RSSI -78dB), the "l" value is the GPS location when the Bluetooth beacon was observed and "ts" the unix timestamp.

Aarogya Setu uses Firebase Analytics, Firebase Remote Configuration and Firebase Cloud Messaging. Aarogya Setu requires Google Play Services to be enabled. When launched with Google Play Services disabled the app halts with a popup asking for it to be enabled and if that popup is ignored then the app exits.

In summary, Aarogya Setu seems significantly less private than the other apps examined here and we urgently recommend that it be modified to (i) not upload the user's location upon launch without explicit notification and consent, (ii) not upload logged bluetooth/location data without explicit notification and consent and (iii) not send the DiD value in the clear in Bluetooth transmissions. In addition, we recommend that the dependencies on Google services and Google Play Services are reduced, or preferably removed and that documentation giving technical details of the operation of the app be published.

7.5.1 Data sent on initial startup

When first launched Aarogya Setu makes three connections to Google servers. Namely, (i) https://settings.crashlytics.com/spi/v2/platforms/android/apps/nic.goi.aarogyasetu/settings to register with Crashlytics, (ii) https://firebaseinstallations.googleapis.com/v1/projects/covid19-6c396/installations to register with Firebase and (iii) https://android.clients.google.com/c2dm/register3 to link Firebase with Google Play Services (linking FirebaseID with the AndroidID). The app then makes a request to https://firebaseremoteconfig.googleapis.com/v1/projects/645345756042/namespaces/firebase:fetch, sending the FirebaseID and Google Play Services authentication token and calls https://app-measurement.com to send telemetry to Google Analytics.

After selecting the language, clicking "next" and proceeding with the onboarding the app sends the entered phone number to https://api.swaraksha.gov.in/generateOTP, and then the phone number plus PIN (which has been sent by SMS) to https://api.swaraksha.gov.in/validateOTP. The response is a JWT-encoded auth_token value that decodes to:

"exp": 1607773824, "iat": 1607687424, "sub": "+919<...>",
"username": "8bae6238−66ac−4bda−9731−508a742d75c0"

Here the "sub" value is the phone number entered and "username" is self-explanatory. Also sent is a JWT-encoded refresh_token which decodes to the same content but a different "exp" expiry time.

The app next sends a request to https://fp.swaraksha.gov.in/api/v1/users/register:

POST https://fp.swaraksha.gov.in/api/v1/users/register
Headers:
  authorization: eyJ0eXAiOiJKV1QiL<...>
Body:
    "d": "02:00:00:00:00:00",
    "ft": "crUG7jyPRES5ghOHE9F6<...>",
    "<...>

[12] onMessageReceived() in file FcmMessagingService.java calls pushDataToServer() when an appropriate message is pushed.

[13] checkStatus() in file CorUtility.kt calls uploadDataUtil.startInBackground() when the response to the status request contains appropriate json entry.

Response:

"data": { "did": "**17d8a9b1**" }, "error": {}

The authorization header is the auth_token value and the "ft" value in the body of the request is the Google Play Services authentication token. The request therefore acts to link these. Recall that the Google Play Services authentication token is linked to the FirebaseID and the AndroidID, and so the effect of this request is to link the phone number, FirebaseID and the AndroidID. The response is a DiD value that acts as a strong, long-lived device identifier, as already noted. We note that inspection of the code indicates the "d" value sent in the request is intended to be the MAC address of the handset bluetooth adapter, but for security reasons ("MAC addresses are globally unique, not user-resettable, and survive factory resets") Android 6 and later block access to the MAC address.[14]

The app now makes a request to https://fp.swaraksha.gov.in/api/v1/users/status, also sending the auth_token value in an authorization header. An example response is:

"did": "17d8a9b1", "full_upload": "0",
"meta": {"color": "green","radius": 0,<...>},
"p": 0, <...>

When the "p" value in this reponse is set to 1 then the app responds by uploading its logged bluetooth/location data to the server (we identified this functionality by inspection of the source code and confirmed its operation by intercepting the response and setting p to 1). The "full_upload" value is used in the decompiled apk but not in the published source code.

Next it sends requests to https://fp.swaraksha.gov.in/api/v1/openapi/approval/ and then to https://web.swaraksha.gov.in/ncv19?locale=en. The response to the latter request is HTML, and the app then proceeds to download the associated HTML resources from using 62 separate requests for images, fonts, CSS and javascript – in this way the bulk of the app UI appears to be dynamically loaded. These requests do not carry any identifiers. The app also makes connections to https://webapi.swaraksha.gov.in/ncv19/nearby-stats/, https://fp.swaraksha.gov.in/api/v1/openapi/userpref/, https://webapi.swaraksha.gov.in/ncv19/did-state, https://webapi.swaraksha.gov.in/ncv19/show-policy/ which send the auth_token value. Importantly, the connection to https://webapi.swaraksha.gov.in/ncv19/nearby-stats/ sends the device GPS location to the server (see below). That is, upon launch of the app the user's location is automatically shared with the central server. No dialogue or popup is raised notifying the user of this or asking for their consent. Finally the app calls https://app-measurement.com/a to share telemetry with Google Analytics.

[14] https://developer.android.com/training/articles/user-data-ids#mac-addresses.

7.5.2 Data sent when sitting idle at main screen

When left idle, in our tests we observed no further connections by the app. However, interaction with the UI does generate network connections. In particular, navigating to the 'Your Status" page prompts a call to https://webapi.swaraksha.gov.in/ncv19/nearby-stats, for example:

GET https://webapi.swaraksha.gov.in/ncv19/nearby-stats/?dist=1km
Headers:
lon: −6.2<...>
distance: 0.5km
lat: 53.3<...>
authorization: eyJ0eXAiOiJKV1Q<...>
Response:"bluetoothPositive": 0, "infected": 0, "selfAsses": 1, "
success": true, "unwell": 1, "usersNearBy": 3

Here the "lat" and "lon" header values are the device location, specified to high accuracy (they correctly locate the house where the handset is located). The response appears to be various statistics on infections in the vicinity (given by the "dist" header value) of this location. The accuracy of these statistics is unclear. They may be obfuscated in some way, we found no documentation on this, but based on our measurements they are not. When we changed the location to be in the ocean the response correctly indicated no people while when the location was set to Bombay Hospital in Mumbai a typical response was "infected":60,"unwell":3,"bluetoothPositive":10530,"success":true,"selfAsses":93,"usersNearBy":46278 and for the parliament in New Dehli "infected":38,"unwell":5,"bluetooth Positive":14301,"success":true,"selfAsses":66,"usersNearBy": 33917.

While this request requires an authorization header, it is easy to extract this header from app network connection traces and use this to make requests to https://webapi.swaraksha.gov.in/ncv19/nearby-stats targetting arbitrary locations, raising obvious privacy concerns.

8 Conclusions

We find that TousAntiCovid and CovidSafe are generally well-behaved with regard to privacy. TraceTogether and CombatCovid make extensive use of Google Firebase services which means that that there are two main parties involved in handling data transmitted from these apsp, namely Google and the health authority operating the app itself. HaMagen is well-behaved for uninfected users of the app but GPS location data associated with infectied is publicly published on the HaMagen server. Aarogya Setu is found to have a number of potentially serious privacy issues.

Funding Open Access funding provided by the IReL Consortium.

References

1. TousAntiCovid App (Accessed 15 Dec, 2020). www.bonjour.tousanticovid.gouv.fr
2. CovidSafe App (Accessed 15 Dec, 2020). www.covidsafe.gov.au
3. TraceTogether App (Accessed 15 Dec, 2020). www.tracetogether.gov.sg
4. CombatCovid App (Accessed 15 Dec, 2020). www.combatcovidapp.com/pbc/
5. HaMagen App (Accessed 15 Dec, 2020). www.govextra.gov.il/ministry-of-health/hamagen-app/download-en/
6. Aarogya Setu App (Accessed 15 Dec, 2020). www.aarogyasetu.gov.in
7. Exposure Notifications: Android API Documentation (accessed 6 June 2020). www.static.googleusercontent.com/media/www.google.com/en/www.covid19/exposurenotifications/pdfs/Android-Exposure-Notification-API-documentation-v1.3.2.pdf
8. Leith D, Farrell S (2021) GAEN Due Diligence: Verifying The Google/Apple Covid Exposure Notification API Proc Corona Defcon21, NDSS
9. Firebase Help: Automatically collected user properties (Accessed 26 April 2020). www.support.google.com/firebase/answer/6317486
10. Leith DJ, Farrell S (2021) A Measurement-Based Study of the Privacy of Europe's Covid-19 Contact Tracing Apps. In: Proc IEEE INFOCOM
11. Dehaye P,Reardon J (2020) Dehaye P,Reardon J (2020) Proximity Tracing in an Ecosystem of Surveillance Capitalism. www.arxiv.org/pdf/2009.06077.pdf
12. Cuebiq Mobility Insights (Accessed 15 Dec, 2020). www.cuebiq.com/visitation-insights-covid19/
13. Decentralised Privacy-Preserving Proximity Tracing (DP-3T) Documents (Accessed 26 April, 2020). www.github.com/DP-3T/documents
14. ROBust and privacy-presERving proximity Tracing protocol (Accessed 17 Dec, 2020). www.github.com/ROBERT-proximity-tracing/documents
15. BlueTrace: A privacy–preserving protocol for community–driven contact tracing across borders (9 April, 2020). www.bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf
16. Wen H, Zhao Q, Lin Z, DoXuan,Shrof N (2020) A Study of the Privacy of COVID-19 Contact Tracing Apps. In: Proc SECURECOMM
17. Sun R, Wang W, Xue M, Tyson G, Camtepe S, Ranasinghe DC (2020) An Empirical Assessment of Global COVID-19Contact Tracing Applications. www.arxiv.org/pdf/2006.10933.pdf
18. Leith D, Farrell S (2020) Coronavirus Contact Tracing App Privacy: What Data Is Shared By The Singapore OpenTrace App? In: *Proc SECURECOMM*
19. Sweeney L (2002) k-anonymity: A model for protecting privacy. International Journal of Uncertainty. Fuzziness and Knowledge-Based System 10(05):557
20. Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) ACM Transactions on Knowledge Discovery from Data (TKDD) 1(1):3
21. Golle P, Partridge K (2009) On the Anonymity of Home/Work Location Pairs Pervasive Computing
22. Srivatsa M, Hicks M (2012) Deanonymizing mobility traces: Using social network as a side-channel. In: Proc CCS. pp. 628–637
23. Cortesi A, Hils M, Kriechbaumer T, contributors (2020) mitmproxy: A free and open source interactive HTTPS proxy (v5.01). www.mitmproxy.org
24. Frida: Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers (Accessed 26 April 2020). www.frida.re