GAEN Due Diligence: Verifying The Google/Apple Covid Exposure Notification API

Douglas J. Leith, Stephen Farrell
School of Computer Science & Statistics,
Trinity College Dublin, Ireland
16th June 2020

***Abstract*—We report on an independent assessment of the Android implementation of the Google/Apple Exposure Notification (GAEN) system. While many health authorities have committed to making the code for their contact tracing apps open source, these apps depend upon the GAEN API for their operation and this is not open source. Public documentation of the GAEN API is also limited, with few details of the way in which reported quantities are calculated. We find that the GAEN API appears to use a filtered Bluetooth LE signal strength measurement that can be potentially misleading with regard to the proximity between two handsets (it may suggest handsets are further apart than they really are). We also find that the exposure duration values reported by the API are coarse grained and can somewhat overestimate the time that two handsets are in proximity. Updates to the GAEN API that can affect contact tracing performance, and so public health, are silently installed on user handsets. While facilitation rapid rollout of changes, the lack of transparency around this raises obvious concerns.**

I. Introduction

There is currently a great deal of interest in the use of mobile apps to facilitate Covid-19 contact tracing. This is motivated by the hope that more efficient and scalable contact tracing might allow the lockdown measures in place in many countries to be relaxed more quickly [1] and that these systems can help "hedge" against the risk of a second wave of the pandemic [2]. In early April 2020, Apple and Google formed a partnership to develop contact event detection based on Bluetooth LE [3]. Following public launch of the Google/Apple Exposure Notification (GAEN) API on 20 May [4], GAEN implementations are now installed on many people's phones and this API is likely to be widely used by national health authority contact tracing apps.

In this report we focus on an independent assessment of the Android implementation of the GAEN API. We look forward to similar work being done with the Apple implementation.

Even though many health authorities have committed to making the code for their contact tracing apps open source, these apps depend upon the GAEN API for their operation. The GAEN API is not open source and the public documentation [5] is limited, with few details on the way in which reported quantities are calculated. On Android the GAEN API is implemented within Google Play Services[1]. The GAEN implementation is silently installed without user opt-in on handsets running Google Play Services, although so far its functionality appears to be disabled until a Google approved app is installed that offers the user the choice to opt-in. It then subsequently also silently updates itself. There is almost no public information on the nature of updates (such as a changelog) nor notice of their release, and users running Google Play Services have no opt out from them.

This state of affairs would be undesirable for any software, but is especially so for software that is likely to be widely used in apps backed by national governments and health authorities and which may impact public health. In this report we take a first step towards public "due diligence" of the GAEN API.

Using experimental measurements we assess the relationship between Bluetooth LE received signal strength (RSSI) and the attenuation duration values reported by the GAEN API. We find that the GAEN API appears to use a filtered RSSI value that can be potentially misleading with regard to the proximity between two handsets (it may suggest handsets are further apart than they really are).

The GAEN documentation is largely silent regarding how the *exposure duration* (durationMinutes) and *attenuation duration* values are calculated and how they should be interpreted.

The "exposure duration" terminology suggests that this value is an estimate of the aggregate time that two handsets are in close proximity, but in fact we find that this value is an estimate of the time during which a handset can see beacons from another handset, regardless of the distance between the two handsets.

Since the GAEN API only scans for Bluetooth LE beacons about every 4 mins, the reported exposure duration information is necessarily coarse-grained. Our measurements show that the reported duration values increase by a sequence of sharp jumps as time progresses and that the reported duration values may run ahead of clock time, e.g. a duration of 20 mins may

[1]The choice to implement the Android GAEN API within Google Play Services mandates the installation and use of Google Play Services in order to use contact tracing apps based on the API. Google Play Services is closed-source proprietary software that is used by many Google products. Coupling national contact tracing apps with Google Play Services therefore means that national governments are potentially being co-opted for the commercial benefit of Google. This warrants further analysis, but is out of scope of this report. Note also that due to this packaging choice and current US export controls, some handset manufacturers and countries may be prevented from using the GAEN API.

be reported when the actual time that two handsets are in proximity is 15 mins.

The behaviour of the reported duration values is not clear when two handsets are only intermittently in proximity, e.g. when two handsets spend several 5 min intervals in close proximity but in between spend time apart. We find that intermittent handset proximity can be roughly inferred from the number of exposure information records with which the API responds, but that periods apart of less than 10 mins may be missed by the API and treated as a single continuous exposure, thus overestimating the exposure time, e.g. two periods of 5 min exposure separated by 5 mins apart is reported as 20 mins exposure. We note that intermittent scanning generally means that changes in proximity that happen between scans cannot be detected.

It is not the inaccuracy or otherwise per se of these duration values that is the primary concern here. Rather it is the lack of information about the design decisions that have been made within the GAEN API, and the lack of open discussion/evaluation of the potential impact of these design decisions on contact tracing efficacy. Note that the GAEN design decisions that happen to have been made are not the only ones possible, other design choices are available. For example the GAEN API might scan more frequently when it detects other handsets nearby so as to obtain more fine-grained information on exposure time. Whether this is a good idea or not is of course unclear, the point is that open discussion and evaluation of the impact that choices such as these have on contact tracing and infection control is needed.

Similarly, while silent updating of the GAEN implementation by Google allows rapid rollout of any "fix", should a "fix" be needed, such updates can affect the performance of contact tracing apps based on the GAEN API, and so in turn affect public health. The lack of transparency around this raises obvious concerns.

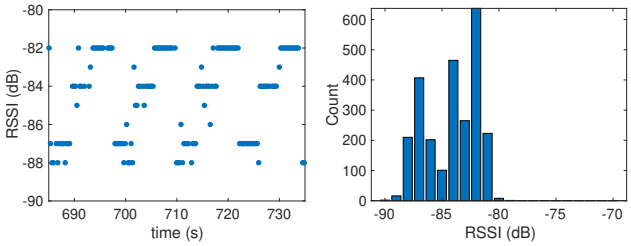
II. Google/Apple Exposure Notification API

A. Hardware & Software Used

We used Google Pixel 2s running GAEN API version 202490002[2].

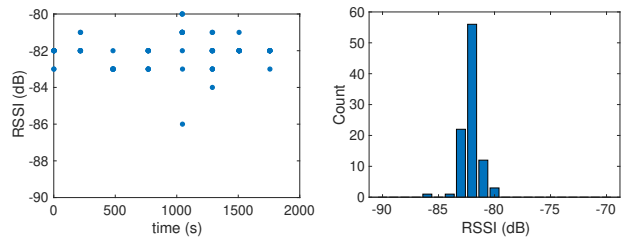
We used a version of the Google exemplar Exposure Notification app modified to allow us to query the GAEN API over USB using a python script (the source code for the modified app is available on github [6]). In addition we also wrote our own GAENAdvertiser app that implements the transmitter side of the GAEN API. GAENAdvertiser is open source and can be obtained by contacting the authors (we have not made it publicly available, however, since it can be used to facilitate a known replay attack against the GAEN API [7]).

[2]As reported using the command "adb shell dumpsys activity provider com.google.android.gms.chimera.container.GmsModuleProvider | grep -i nearby", although in recent versions of the GAEN API the version number is now also shown in the *Settings-COVID 19 Notifications* display.



(a) (b)

Fig. 1: Measured RSSI for two handsets placed 1m apart. (a) Snapshot of the corresponding RSSI time history, (b) histogram showing number of times each RSSI value is observed. The regular pattern that can be seen in (a) is associated with channel hopping between the three channels on which Bluetooth advertises beacons, and observe also the three peaks in (b) at around -82dB, -84dB and -87dB.



(a) (b)

Fig. 2: RSSI values logged by GAEN API for two handsets placed 1m apart. Comparing (a) with Figure 1(a) it can be seen that the GAEN API focusses on the RSSI values around -82dB. This is also evident when comparing the histogram data in (b) with that in Figure 1(b) from which it can be seen the the GAEN API RSSI values correspond to the first peak at -82dB, with the other peaks having been filtered out.

B. Measuring Bluetooth RSSI

Bluetooth advertises beacons on three radio channels so as to improve tolerance of interference. Beacons are broadcast on each channel in turn. Each channel may have different radio propagation characteristics, and be subject to different levels of interference from other transmitters, and so beacons sent from different channels may be received with different signal strengths. This behaviour can be seen, for example, in Figure 1(a) which plots the RSSI values reported by the standard Android BluetoothLeScanner API[3] when two handsets are placed 1m apart in an indoor location. It can be seen that the reported RSSI values roughly hop between -82dB, -84dB and -87dB in a regular, repeating pattern. Figure 1(b) plots counts of the number of times that each RSSI value is reported over a 20 minute period. Note the three peaks at -82dB, -84dB and -87dB, consistent with Figure 1(a).

Although we cannot see inside the GAEN API, on Android

[3]See https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner. The scanner was configured to use SCAN_MODE_LOW_LATENCY so that it tries to report the RSSI of every beacon observed.

it does report RSSI values in the handset log. Figure 2 plots these values for the same setup at in Figure 1 (so these two figures can be directly compared). It can seen that the GAEN API RSSI values seem to correspond to beacons from only one channel, namely the channel corresponding to the -82dB peak in Figure 1(b). Similar behaviour is observed across a range of handset configurations and indicates that the GAEN API is carrying out filtering of the RSSI values (we will return to this later). One immediate implication of this is that some caution is needed if using RSSI values reported by the BluetoothLeScanner to estimate/predict performance of the GAEN API.

Also evident from Figure 2 is that the GAEN API only scans for beacons roughly every 250s (about 4 mins). In other data we have observed the GAEN API scanning roughly every 150s (about 2.5 mins) and so it appears that the scan interval used may be variable.

C. GAEN Attenuation Thresholds & Durations

The GAEN API does not report RSSI values but rather *attenuation durations* above/within/below *attenuation thresholds* that are specified when querying the API. The API documentation [5] is silent on how the attenuation thresholds relate to RSSI, but we can deduce a fair amount from measurements.

1) Attenuation: When we query the API we supply two attenuation thresholds A and B. The API then responds with three values:[4] (i) the duration the attenuation was below A, (ii) the duration the attenuation was between A and B and (iii) the duration the attenuation was above B. For example we might choose A=48dB, B=63dB and obtain values 0, 10, 30 indicating no attenuation values below 48dB were observed, attenuation values between 48dB and 63dB were observed for a duration of 10 mins and attenuation values above 63dB were observed for 30 mins.

We can make repeated queries with different values of A and B. In particular, by holding A constant at a low value, e.g. 48dB, and then increasing B from 48dB to a high value, e.g. 100dB, in steps of 1 dB then we can extract fine-grained information on the durations at each attenuation value in the range 48dB to 100dB.

Figure 3(a) plots the attenuation value durations obtained using this repeated query approach for two handsets placed 1m apart. That is, corresponding to Figures 1 and 2. It can be seen that the attenuation values are tightly concentrated on 70dB whereas in Figure 2 the RSSI values are concentrated around -82dB. The "attenuation" terminology suggests that the attenuation value is given by the transmit power minus the RSSI, perhaps with some additional calibration for device specific characteristics. Supposing the transmit power to be -16dB (we will return to this shortly) then that suggests an attenuation value of 66dB. The handset log entries generated by the GAEN API suggest an offset of -4dB is added[5] to the RSSI to obtain a "calibrated RSSI". Adding this offset then suggests an attenuation value of 70dB, matching what we see in Figure 3(a).

The transmit power used in this attenuation calculation appears to be derived from the metadata included in GAEN beacons. We wrote an app that generates beacons according to the GAEN format and which allows us to specify the metadata sent in the beacons. Figure 3(b) plots the attenuation durations as the *transmit power level* value in the beacon metadata is varied from 0dB to 15dB for two handsets placed 1m apart. Note that in this plot we present the data using a coloured heatmap. We split the range of attenuation values shown on the y-axis into 2dB bins i.e. 70-72dB, 72-74dB and so on. Within each bin the colour indicates the percentage of the total reported duration that was spent in that bin, e.g bright green indicates that more than 90% of the time was spent in that bin. The mapping from colours to percentages is shown on the righthand side of the plot. Bins with no entries (i.e. with duration zero) are left blank. The solid line indicates the average attenuation level at each transmit power level (the average is calculated by weighting each attenuation level by the duration at that level and then summing over all attenuation levels).

It can be seen from Figure 3(b) that the GAEN API reports that the attenuation durations are consistently concentrated in a single bin, as expected from Figure 3(a) (which is for the same physical setup). It can be seen that the attenuation value where the API reports most of the time being spent decreases linearly as the transmit power level sent in the beacon metadata is increased, as expected if the metadata transmit power value is being used in the attenuation calculation.

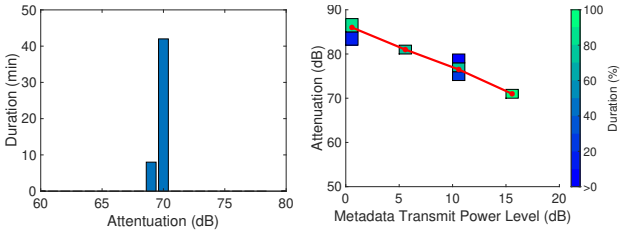
In these tests we used Google Pixel 2 handsets. The Android Bluetooth LE Advertiser API reports that the transmit power associated with advertising beacons at the TX_POWER_LOW setting is -16dB. When we use this metadata value in our app the attenuation durations reported by the GAEN API are essentially identical to those shown in Figure 3(a)[6].

In summary, by making repeated queries to the GAEN API we can obtain the reported duration value at each attenuation level. The attenuation value itself seems to be calculated as the $P_{TX} - P_{RX}$ where P_{TX} is the transmit power level sent in the beacon metadata and P_{RX} is given by the filtered RSSI plus a calibration value which is -4dB for the Google Pixel 2 handsets used in these tests.

[4]The wording in the GAEN API documentation is that the duration value is "the sum of all durations for all exposures when attenuation was less than the low threshold" and that this value is in minutes. However, it is not clear what is meant by an "exposure" here, nor how the duration value is actually calculated bearing in mind that the API only scans for beacons intermittently at intervals of up to 5 mins.

[5]This value seems to be dependent on the handset model, e.g. we observe that it is -4dB for Google Pixel 2s and -3dB for Samsung Galaxy A10s.

[6]Note that it is possible, indeed likely, that in future Google may adjust the transmit power level value sent in the beacon metadata in an attempt to compensate for differences in Bluetooth LE transmission characteristics between different models of handset. For example, in the most recent version of GAEN it appears that on Pixel 2s the transmit power level sent in the beacon metadata is now -17dB rather than -16dB.



(a) (b)

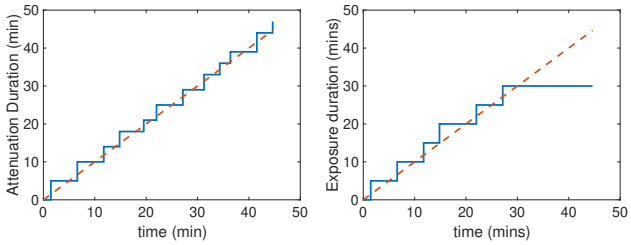
Fig. 3: Measured attenuation durations reported by the GAEN API vs time for two handsets placed 1m apart. (a) shows data corresponding to that in Figure 2, (b) shows the reported durations as the transmit power level value in the GAEN beacon metadata is varied from 0 to -15dB.

2) Attenuation Duration: Figure 4(a) plots the total[7] reported attenuation duration vs time for two handsets placed 1m apart, i.e. the same setup as considered above. We queried the GAEN API regularly every minute and the solid line in Figure 4(a) shows the evolution of reported attenuation delay value over time. It can be seen that the attenuation duration increases in discrete jumps, rather than continuously, presumably reflecting the intermittent nature of the beacon scanning process noted previously. Also, that it can get ahead of clock time (this occurs when it lies above the dashed line marked in Figure 4(a)). For example, early in this test (well before 5mins clock time has elapsed) the attenuation is reported as being 5mins.

In Figure 4(a) the two handsets are in continuous proximity. We also collected measurements when are in proximity for 5 mins, separated for a variable period and then brought in proximity again for 5 mins (we simulated separating the handsets by disabling beacon transmission for a specified period). When the handsets are separated for 5 mins the API responds to a query with a single exposure information records that reports the attenuation duration 19 mins, suggesting that the GAEN API ignores the period when the handsets are separated. When the handsets are separated for 10 mins the API responds to a query with two exposure information records, the first reporting an attenuation duration on 5 mins and the second an attenuation duration of 10 mins. So although the aggregate attenuation duration of 15 mins is an overestimate (the actual time the handsets are together in only 10 mins) by responding with two exposure information records the API does seem to recognise that the handsets have been separated. When the handsets are separated for 15 mins the API response with two exposure information records each reporting an exposure duration of 5 mins, i.e. 10 mins in total.

In summary, the attenuation duration value seems to roughly track clock time, in line with the GAEN API documentation [5]. Intermittent handset proximity can be roughly inferred from the number of exposure information records with which the API responds.

[7]That is, the sum of the three attenuation duration values reported by the API.



(a) Attenuation duration (b) Exposure duration

Fig. 4: Attenuation and exposure durations reported by the GAEN API vs time for two handsets placed 1m apart. The 45° line is indicated by dashes.

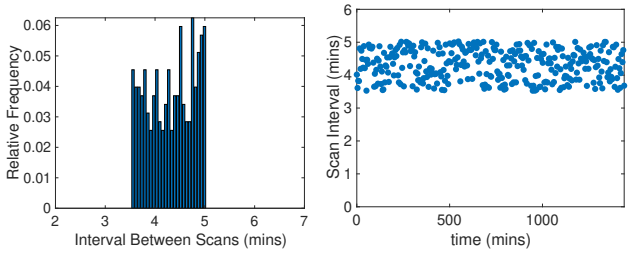
D. GAEN Exposure Duration

The GAEN API reports an *exposure duration* value, but the documentation [5] says little about how this value is calculated beyond the fact that its value is given in units of 5 mins,.

Figure 4(b) plots the exposure duration reported by the GAEN API vs time when two handsets are placed 1m apart. We queried the GAEN API regularly every minute and the solid line in Figure 4(b) shows the evolution of reported exposure duration value over time. It can be seen that the exposure duration increases in 5 min jumps. Also, that it is frequently ahead of clock time (this occurs when it lies above the dashed line marked in Figure 4(b)) and is capped at 30 mins.

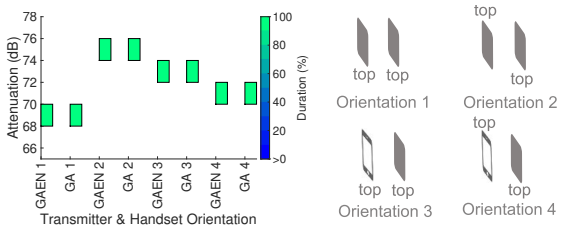
The terminology "exposure duration" may suggest that the reported value only counts the time when two handsets are in proximity e.g. when the attenuation is below one of the thresholds specified when querying the API or when handsets are less than 2m apart. However, we observed the reported exposure duration value to be the same regardless of the thresholds specified and that the exposure duration value reports time spent when handsets are 4m apart. It therefore seems more appropriate to interpret the reported exposure duration value as the time during which two handsets are in close enough proximity for one to observe the Bluetooth beacons of the other, regardless of the received signal strength. Note that Bluetooth LE beacons typically have a range of 5-10m (and potentially further for devices using the newer Bluetooth LE v5.0).

Similarly to Section II-C2 we also observed the reported exposure duration when two handsets are in proximity for 5 mins, separated for a variable period and then brought in proximity again for 5 mins. When the handsets are separated for 5 mins the API responds to our query with a single exposure information record that reports an exposure duration is 20 mins. When they are separated by 10 mins the API responds to our query with two exposure information records, one reporting an exposure duration of 5 mins and the second reporting an exposure duration of 10 mins. So, once again, although the aggregate exposure duration of 15 mins is an overestimate, by responding with two exposure information records the API seems to recognise that the handsets have been separated. When the handsets are separated for 15 mins



(a) (b)

Fig. 5: Measured intervals between GAEN API Bluetooth LE scans, based on 24 hours of data. (a) Relative frequency of scan intervals, (b) time history.



(a) (b)

Fig. 6: Attenuation durations reported by GAEN API for two Google Pixel 2 handsets placed 0.5m apart. One handset runs both the GAEN API and the GAENAdvertiser app and its orientation is changed as shown in (b). The x-axis labels indicate the transmitter (GAEN and GA, respectively) and the handset orientation for each GAEN report.

the API response with two exposure information records each reporting an attenuation duration of 5 mins, i.e. 10 mins in total.

In summary, the exposure duration value reported by the GAEN API seems to roughly track the aggregate time during which a handset can see beacons from another handset, regardless of the distance between the two handsets. Intermittent handset proximity can be roughly inferred from the number of exposure information records with which the API responds.

E. Scanning Frequency

The GAEN API scans intermittently for beacons. We left two handsets beside one another for 24 hours and using the GAE log reports recorded the intervals between scans for Bluetooth LE beacons. Figure 5 shows the relative frequency of the interval between scans. It can be seen from Figure 5(a) that the scan intervals are roughly uniformly distributed between 3.5 mins and 5mins and from the time history in Figure 5(b) that the distribution of scan intervals is roughly constant over time (it does not, for example, increase overnight to reduce battery drain).

F. Inter-operability of GAEN Beacons

The GAEN API documentation [5] specifies the format of the Bluetooth LE beacons and the cryptographic protocol used to generate the payload of each beacon. To verify compliance of the Google Android GAEN API implementation with the specification, and to confirm inter-operability, we wrote the GAENAdvertiser app that generates beacons that follow the Bluetooth LE beacon protocol specified in the documentation.

This highlighted two lacuna's in the API documentation. Firstly, AES-CTR encryption is specified for the beacon metadata, but it is not stated when the counter used in this encryption is initialised. Our tests indicate that it is initialised when the first RPI is sent, and then incremented each time the RPI changes. Secondly, it is not stated how the metadata transmit power level value should be chosen. As discussed above, our measurements indicate that in fact this value is used by the GAEN API to calculate the attenuation level and so plays a key role in its operation.

Figure 6 shows example measurements comparing the reports generated by the GAEN API when it receives beacons generated by the GAEN API and beacons are generated by our GAENAdvertiser app (both running on the same handset to remove any other differences). This data is for two Pixel 2 handsets placed 0.5 apart. The relative orientation of the handsets is changed as shown schematically in Figure 6(b) and the GAEN API on one handset queried. Figure 6(a) shows the attenuation durations reported by the GAEN API in response to these queries for each orientation and for both the GAEN API on the second handset and the GAENAdvertiser app (labeled GAEN and GA respectively on the x-axis of Figure 6(a)). It can be seen that the reports are identical for the GAEN API and GAENAdvertiser beacons, and we also see similar behaviour in other measurements.

GAENAdvertiser is open source and can be obtained by contacting the authors. However, we have not made it publicly available since it can be used to faciltate a known replay attack against the GAEN API [7].

G. MAC Address Randomisation

The GAEN documentation specifies that Bluetooth LE beacon MAC address randomisation should be used when available, and that when the beacon content periodically changes the MAC address should also change so as to prevent linking of beacons from the same handset. In our measurements we observed that the GAEN API respected this behaviour.

We note in passing that we found that when using the standard Android Bluetooth LE advertiser API the beacon MAC address changes whenever advertising is stopped and then restarted. Co-ordination of MAC address changes with changes in beacon content is therefore also straightforward to implement in, for example, the GAENAdvertiser app.

III. GAEN RSSI Filtering

In this section we take another look at the RSSI values which the GAEN API reports in the handset log. Recall that the attenuation durations reported by the API in response to queries seem to effectively be derived from these RSSI values. Namely, by subtracting the measured RSSI values from

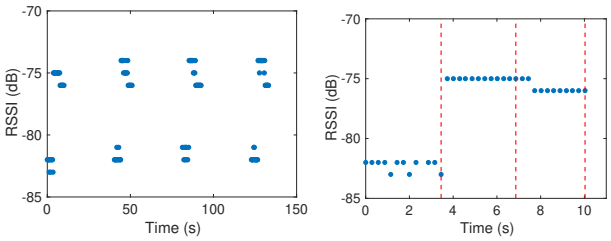


Fig. 7: RSSI measurements reported by the standard Android Bluetooth scanner API when it is periodically (roughly every 30s) switched on, the first 36 RSSI values reported logged and then scanning halted. (b) is a zoomed-in view of the first set of RSSI measurements in (a). This data is for two Google Pixel 2's positioned 0.5m apart.

the transmit power level value sent in the beacon metadata, together with a calibration adjustment to try to compensate for differences between handset models. These RSSI measurements are therefore central to the functioning of the GAEN API for proximity detection.

The GAEN API code on Android is part of Google Play Services, and closed source. Further, as already noted the documentation is limited. We therefore do not know the way in which it operates internally when measuring the RSSI of Bluetooth LE beacons. With that caveat in mind, we can nevertheless make a reasonable guess.

We start by looking at the standard Android BluetoothLE scanner. The relevant code for this is also closed source, but we can deduce some aspects of its operation from Figure 7. This figure shows the RSSI values reported by the Android BluetoothLE scanner API when it is periodically (roughly every 30s) switched on, the first 36 RSSI values reported logged and then scanning halted. The beacons in this example are generated by a second handset running the GAEN API and located 0.5m away, but the behaviour is not specific to this. The scanner is configured to use SCAN_MODE_LOW_LATENCY, and so it tries to report on every beacon observed.

It can be seen from Figure 7(a) that the RSSI values reported by the scanner follow a regular pattern. Figure 7(b) shows a zoomed-in plot of the first set of samples, the vertical lines marking every 12 readings. Recall that Bluetooth LE transmits beacons on three separate radio channels, and so the scanner needs to listen to each of these three channels. What appears to be happening is that the scanner listens to one channel for a period of time (around 3s, corresponding to 12 beacons sent at 250ms intervals), then switches the next channel and listens to that for a period time (again, around 3s), then switches to the third channel. Since the channels are at different radio frequencies they have different signal propagation between the two handsets and so different RSSI values. Hence, as the scanner hops between the three channels we see jumps in the RSSI values roughly every 3s or 12 beacons. If we configure the scanner to collect more than 36 beacons then this pattern of RSSI jumps appears to repeat, with a switch back to values around -82dB for the next 12 beacons after the 36th beacon.

This behaviour is not especially interesting in itself, but is relevant to the GAEN API and Bluetooth LE contact detection for at least two reasons.

A. GAEN Effectively Monitors Only One Channel

Firstly, it means that it is likely that the GAEN API only listens to one out of the three channels that Bluetooth LE uses for transmitting beacons.

As can be seen from Figure 7(a) when first switched on the scanner always starts at the same channel (the one with RSSI around -82dB in this example). The channel hopping part of the scanner functionality is managed internally within Android, likely within the Bluetooth hardware driver[8]. That means it is also likely shared by the GAEN API. Based on the entries that it writes to the handset log the GAEN API scans for beacons periodically, around every 4 mins, and records the RSSIs for around 12 beacons. With Figure 7 in mind this suggests that the GAEN API collects RSSI values for only one of the three channels that Bluetooth LE uses for transmitting beacons. This is also consistent with the behaviour previously noted in Figure 2, and the observation that the GAEN RSSI values seem to correspond to only one of the three peaks in Figure 1(a).

B. Which Channel to Choose?

Collecting RSSI values for only one channel is potentially appealing as it removes the "noise" caused by hopping between channels, e.g the "smoother" nature of the values in Figure 2(a) compared to those in Figure 1(a) is immediately apparent. However, it can have some unwanted side effects.

Comparing again Figure 2(a) with Figure 1(a), it can be seen that the channel monitored by the GAEN API has the highest RSSI of the three channels at -82dB. However, this need not always be the case. For example in Figure 7 the channel that the scanner listens to first has the *lowest* RSSI of the three channels and so can potentially give a misleading view of proximity to the second handset, namely suggesting that the handset is further away than it really is.

This concern is not a hypothetical one. Figure 8 plots measurements taken on two Google Pixel 2 handsets placed 0.5m apart as their relative orientation is varied. The handsets are placed vertically, as illustrated schematically in Figure 6(b). The left-hand handset is rotated and flipped through each of the orientations shown in Figure 6(b) in turn. In this schematic the top of the handsets is indicated, and the back of a handset is indicated as shaded. For example, in orientation 1 both the both handsets are top down with the screen facing to the right. In orientation 4 the left-hand handset is bottom down with the screen facing left, and so on.

[8]Note that this means that the behaviour may change with the wireless chipset used in a phone. We used Google Pixel 2's on our tests but also see similar behaviour on a Samsung Galaxy A10 and a Huawei P10.

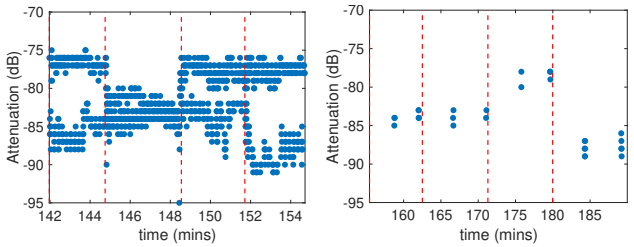


Fig. 8: Measurements taken on two Google Pixel 2 handsets placed 0.5m apart as their relative orientation is varied as shown schematically in Figure 9(b). In (a) the RSSI values are measured using the standard Android Bluetooth LE scanner, while in (b) the values are those written to the handset log by the GAEN API.

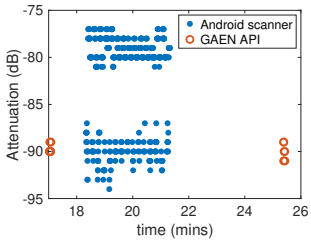


Fig. 9: Measurements taken on two Google Pixel 2 handsets placed 0.5m apart for orientation 4. The Android Bluetooth LE scanner is switched on to collect samples between two scans by the GAEN API. See Figure 6(b) for schematic showing orientations.

In Figure 8(a) RSSI measurements from the standard Android Bluetooth LE scanner API are plotted as the orientation of the left-hand handset is changed. The vertical dashed lines indicate the times when the orientation changes. It can be seen that the changes in relative orientation change the RSSI, and in particular the spread of RSSI values across the three channels. For example, for the first orientation from time 142-144mins the RSSIs are divided into two bands, one around -87dB and another around -77dB, but for the second orientation from 144-148mins the RSSI/attenuations are bunched around -84dB.

Figure 8(b) shows the RSSI values logged by the GAEN API when the same experiment is repeated. It can be seen that for the first orientation the GAEN API uses RSSI values from the lower -85dB band rather than the higher -77dB band, whereas for the third orientation the GAEN API uses RSSI values from the higher 77dB band rather than the lower -84dB band that can be seen in the third section of Figure 8(a). For the last orientation the GAEN API uses RSSI values from the lower -87dB band in Figure 8(a) rather than the higher -77dB band.

In Figure 8 we carried out the experiment twice so that the Android Bluetooth LE API operation does not contaminate the GAEN API operation. We also carried out an experiment where we switched the Android scanner on between two scans by the GAEN API, see Figure 9, and it can be seen that the behaviour is similar (this time with the GAEN API using RSSI values drawn from the -90dB band rather than the -77dB band)..

C. A Possible Fix

The concern with the GAEN API taking RSSI measurements from only one out of the three channels used to transmit Bluetooth LE beacons is that this may give a misleading view of the proximity of handsets. In particular, if the GAEN API measures RSSI on a channel which is subject to more attenuation then it may appear that the two handsets are further away than they really are.

There is at least one simple fix to this, assuming the above analysis of the channel hopping used for scanning is correct. Namely, on each scan wait for 36 packets (or whatever time interval is needed for the scanner to scan all three channels) instead of 12 packets. In that way we can observe RSSI values from all three channels, and so gain a more complete view of channel conditions. One might, for example, then choose to use the RSSI values from the channel with highest RSSI so as to try to provide a more consistent measure of proximity. Or perhaps record the RSSI value per channel and then use these three RSSI values together to more reliably estimate proximity.

IV. Summary and Conclusions

We first note that from an architectural perspective Google and Apple are today probably well placed to handle the kinds of issue noted here and to provide a "fix" that gets widely deployed, should a "fix" be needed. However, silently shipped changes to closed-source proprietary GAEN implementations can affect the performance of public health contact tracing apps based on the GAEN API, e.g. affecting the rate of exposure detection false positives and false negatives. The lack of transparency around this raises obvious concerns and reduces the value of health authorities and other app implementers making their client code publicly available.

Acknowledgements

The authors would like to extend their thanks to the Irish HSE for arranging with Google for us to have whitelisted access to the GAEN API. We emphasise that any views expressed in this report are the authors own, and may not be shared by the HSE. Trinity College Dublin, (the authors' employer) funded the "Testing Apps for Contact Tracing" (TACT) project [9] that has allowed us the time and handsets required for this work.

References

[1] L. Ferretti, C. Wymant, M. Kendall, L. Zhao, A. Nurtay, L. Abeler-Dörner, M. Parker, D. Bonsall, and C. Fraser, "Quantifying sars-cov-2 transmission suggests epidemic control with digital contact tracing," *Science*, 2020.

[2] Irish Times, "EU urges vigilance to avoid coronavirus second wave," 17 May 2020. [Online]. Available: https://www.irishtimes.com/news/world/europe/eu-urges-vigilance-to-avoid-coronavirus-second-wave-1.4255632

[3] "Apple and Google partner on COVID-19 contact tracing technology," 10 April, 2020. [Online]. Available: https://www.apple.com/newsroom/2020/04/apple-and-google-partner-on-covid-19-contact-tracing-technology/

[9] See https://down.dsg.cs.tcd.ie/tact/

[4] Google Blog, “Exposure Notification API launches to support public health agencies,” Accessed 13 June 2020. [Online]. Available: https://blog.google/inside-google/company-announcements/apple-google-exposure-notification-api-launches/

[5] “Exposure Notifications: Android API Documentation,” accessed 6 June 2020. [Online]. Available: https://static.googleusercontent.com/media/www.google.com/en//covid19/exposurenotifications/pdfs/Android-Exposure-Notification-API-documentation-v1.3.2.pdf

[6] D. Leith and S. Farrell, “Modified Exposure Notification App,” 9 June 2020. [Online]. Available: https://github.com/doug-leith/BLEapp

[7] S. Farrell and D. Leith, “A Coronavirus Contact Tracing App Replay Attack with Estimated Amplification Factors,” 19 May 2020. [Online]. Available: https://down.dsg.cs.tcd.ie/tact/replay.pdf