On the Accuracy of Measured Proximity of Bluetooth-Based Contact Tracing Apps

Qingchuan Zhao(✉), Haohuang Wen, Zhiqiang Lin, Dong Xuan, and Ness Shroff

Department of Computer Science and Engineering, The Ohio State University, Columbus, USA
{zhao.2708,wen.423,lin.3021,xuan.3,shroff.11}@osu.edu

Abstract. A large number of Bluetooth-based mobile apps have been developed recently to help tracing close contacts of contagious COVID-19 individuals. These apps make decisions based on whether two users are in close proximity (*e.g.*, within 6 ft) according to the distance measured from the received signal strength (*RSSI*) of Bluetooth. This paper provides a detailed study of the current practice of *RSSI*-based distance measurements among contact tracing apps by analyzing various factors that can affect the *RSSI* value and how each app has responded to them. Our analysis shows that configurations for the signal transmission power (*TxPower*) and broadcasting intervals that affect *RSSI* vary significantly across different apps and a large portion of apps do not consider these affecting factors at all, or with quite limited tuning.

Keywords: Bluetooth · BLE · Proximity measurement · COVID-19 · Contact tracing

1 Introduction

COVID-19 has created an unprecedented social and economic crisis across the globe. As of August 2020, there are more than 25 million infected patients. and over 840 thousand deaths worldwide. Since COVID-19 will not disappear shortly, practical techniques must be used to fight this pandemic before vaccines are available. Contact tracing, i.e., identifying people who have been in close contact with contagious individuals, has been such a practical technique for a long time. However, existing contact tracing is manual, and is hard to scale to large and rapidly moved populations. Further, manual tracing may result in delays, which could limit its utility. Therefore, recently numerous digital contact tracing systems have been developed and deployed across the globe, by using a variety of sources including locations measured from cellular networks, WiFi hotspots, or GPS, and cryptographic tokens exchanged via Bluetooth Low Energy (BLE).

Among the digital contact tracing systems, BLE has emerged as a promising solution [21] due to its ubiquity (almost everyone holds a smartphone today),

N. Park et al. (Eds.): SecureComm 2020, LNICST 335, pp. 49–60, 2020.
https://doi.org/10.1007/978-3-030-63086-7_4

availability (almost all smartphones have enabled Bluetooth by default), and privacy preserving (e.g., no real location is involved). The idea of using BLE for contact tracing is straightforward. When two users encounter, contact tracing apps automatically exchange information with each other to record such a contact event for both users. A contact event, in general, includes cryptography generated random tokens that represent users, timestamps for duration estimation, and information for distance measurements. In particular, most BLE contact tracing apps use the received signal strength indicator (*RSSI*) of the Bluetooth for distance measurements. In addition, for COVID-19, a close contact refers to a user who has been in within 6 ft range of a contagious individual for more than 15 min according to the recent CDC guidelines [14]. As such, the effectiveness of the Bluetooth-based contact tracing crucially depends on the accuracy of the measured distance from *RSSI*.

Unfortunately, in practice, numerous factors can affect the *RSSI* that can make the distance measurement inaccurate, such as the power of antenna used for broadcasting (i.e., the *TxPower*) and the obstacles blocking transmission paths. Moreover, Bluetooth-based proximity tracing can also raise false positives because of the potential misinterpretation of various scenarios. For example, a proximity tracing system may interpret two users have a contact even if they are separated by a solid wall, where the risk of infection is much lower than the risk indicated by the measured distance.

Therefore, it is imperative to study how current Bluetooth-based mobile contact tracing systems perform the proximity measurement. To this end, we exhaustively collect 20 Bluetooth-based mobile contact tracing apps from various public sources (e.g., [5,22,31]), systematically inspect the affecting factors that impact the *RSSI*, and examine how each app calculates the proximity distance. Our analysis results have revealed a number of findings:

- Advertising behaviors are highly customized by different mobile apps and the combination of the level of *TxPower* and advertising interval is inconsistent across mobile apps : our analysis have discovered 8 different combinations of the advertising interval and the level of *TxPower* from 20 mobile apps (Sect. 3).
- A large portion of apps do not have an accurate and reliable measured distance from Bluetooth: (*i*) our analysis has identified that 4 apps just use *RSSI* for distance measurement without any tuning, (*ii*) 60% of these apps do not consider affecting factors from hardware specifications, and (*iii*) none of them considers environmental factors (Sect. 4).

2 Background

2.1 BLE-based Contact Tracing

Bluetooth Low Energy (BLE) is a wireless communication technology that is designed to provide basic Bluetooth functionality while consuming considerably

low amount of energy. Because of its low energy consumption and its wide availability in almost every smartphone, BLE is considered a promising solution for mobile contact tracing [21].

Using BLE for contact tracing between smartphones is a complicated process. First, a BLE-based contact tracing mobile app will need to be installed. Then, the app will periodically generate a random token for each user as an identifier and constantly advertise this token (which works like a beacon) or BLE-contact tracing service information (in which establishing connection to exchange information is required) to nearby smartphones. Meanwhile, the app also keeps scanning for other smartphones. When two smartphone users encounter each other, the apps on two phones will automatically exchange the necessary information to record this *contact event*, such as the timestamp, the identifier of users, and most importantly, the data used for distance measurements [17]. In addition, data can be exchanged via device connections or by directly reading from the advertising packets. When a user is tested positive, the app will immediately inform all other users who have close contact with this individual. This exposure notification process is implemented differently according to the different architectures, *i.e.*, centralized and decentralized architecture.

- **Centralized.** In a centralized architecture, users receive exposure notifications from the server that remotely determines the risk of infection. In particular, a centralized service will require users who have tested positive to upload their recent contact events to the central server. Then the server will analyze these events to identify all other users who have been exposed to this individual, and notify each of them according to the user contact information (e.g., cell phone number) that is usually collected at user registration. There are several privacy preserving protocols using this type of architecture, such as BlueTrace [4] and ROBERT [24].
- **Decentralized.** In contrast, in a decentralized architecture, it is the client, instead of the server, that determines its own risk of infection. Only the contact events of the contagious users are shared on a public database, and each client will synchronize its own data with the database periodically. Whenever a synchronization is accomplished, the client app will locally check its own contact events against the updated data to determine its own risk of infection. Many privacy preserving contact tracing protocols such as DP-3T [28] and Notification Exposure [3] use such a decentralized architecture.

2.2 Proximity Measurement in BLE-based Contact Tracing

***RSSI* -based Proximity Measurement.** For two BLE devices, their proximity measurement depends on the received signal strength from each other, also known as *RSSI*, which is proportional to the distance of signal transmission in theory. However, in practice, *RSSI* can be impacted by many factors that may result in inaccurate proximity measurements, and these factors can be classified into internal factors and external factors.

(I) Internal Factors Affecting *RSSI*. Factors within a Bluetooth device including the specifications of both hardware and software can influence the *RSSI* value [9]. With respect to smartphones, the internal hardware factors are the Bluetooth chipset and its antenna layout, and the key software factors include both configurations of the operating system and the mobile app itself [6].

- **Factors from Hardware—chipset and antenna**: A Bluetooth chipset determines the maximum transmission power of the signal and maps the received signal strength to *RSSI* values. Such mapping is highly customized by manufacturers [9] that indicates the same signal can be interpreted as heavily different *RSSI* values across different chipsets. Additionally, the antenna layout, orientation, as well as the capability of data transmission can dramatically affect the strength of emitting and receiving signals [7].
- **Factors from Software—OS and App**: Both Android and iOS can significantly change the power consumption of BLE operations [6], *e.g.*, low battery mode, that could impact the transmission power and the *RSSI* value. In addition to the OS, mobile apps can use system APIs to configure its broadcasting attributes, such as *TxPower*, broadcasting interval, and duration. These attributes can also impact the reliability of *RSSI* values.

(II) External Factors Affecting *RSSI*. In addition to the internal factors, factors outside the device can also influence the *RSSI* value. At a high level, these factors can be classified into two categories: invisible radio waves and visible physical objects.

- **Invisible radio waves**: Bluetooth signals can be interfered by other types of radio waves. For example, if WiFi is mis-configured to use channels that overlap with channels used in Bluetooth, both signals may interfere with each other [12] that can make the obtained *RSSI* value less accurate.
- **Visible physical obstacles**: Obstacles on the transmission path can result in fluctuated *RSSI*. In particular, different materials, such as woods, water, and glass, as well as different textures on surface of objects can lead to different levels of signal interference, such as absorption, interference, and diffraction, that may make the *RSSI* unstable [11,12].

3 Analysis of BLE Software Configurations

In this section, we analyze the affecting factors of proximity accuracy from mobile apps. Ideally, we would like to analyze all affecting factors listed in Sect. 2.2. However, it is extremely challenging to analyze the affecting factors from the operating system because of their different battery management strategies that rarely quantitatively clarify the restrictions of BLE usage. Additionally, the affecting factors from hardware specifications have been studied before in TraceTogether [20]. As such, we focus on the available configurations in mobile apps that control either the settings of advertising or the data included in advertising packets.

BLE Advertising Settings. The settings of advertising determine how a device broadcasts BLE advertising packets. In total, there are three configurable behaviors (only in Android) that are relevant to proximity measurement.

- **Level of advertising interval**: The advertising interval is configurable in Android apps and it is controlled by the mode of advertising. In total, there are three modes of advertising: (*i*) LOW_POWER (0) mode, which is the default mode and broadcasts packets every 1 s; (*ii*) BALANCED (1) mode, which broadcasts every 250 ms; and (*iii*) LOW_LATENCY (2) mode, which broadcasts every 100 ms [2].
- **Duration of advertising**: While Android apps can constantly broadcast advertising packets until being terminated, they are allowed to limit the broadcasting duration (up to 3 min). The duration of broadcasting is important for receivers to read a reliable signal as more samples for adjustment are supposed to be received in a longer duration.
- **Level of transmission power (*TxPower*)**: This attribute controls the emission power of signals. In general, a stronger *TxPower* can increase the stability of signal in transmission [11]. In Android, there are four levels of *TxPower*: ULTRA_LOW (0), LOW (1), MEDIUM (2), and HIGH (3), where the HIGH level provides the best range of signal visibility and the default level is LOW [2].

Data Included in Advertising Packets. In addition to configuring broadcasting behaviors, apps are also allowed to customize the data carried within their advertising packets. Among a variety of data that can be included in advertising packets, we focus on the data for proximity measurement, *i.e.*, the level of *TxPower*. In addition, this value is crucial to accurately determine the distance between users since the same signal strength can be interpreted as different *RSSI* values given different levels of *TxPower* [10]. In particular, the *TxPower* value can be included in a separate filed or in a general field, which is integrated with other information.

- ***TxPower* included in the separated field**: Both Android and iOS allow mobile apps to include the level of *TxPower* in a separate field in advertising packets but with different policies. Specifically, iOS apps are required to include this value in advertising packets and Android apps can choose whether to include this value or not.
- ***TxPower* included in integration**: Other than being included in advertising packet separately, the level of *TxPower* can also be integrated with other information and stored in general data fields, *i.e.*, the field of manufacture data and service data. In addition, while both fields are allowed for customization in Android, only service data can be customized in iOS.

Table 1. BLE advertising configurations in mobile apps (Note that ∞ represents infinity).

App name	Advertising (Adv.) settings			Adv. Data	
	TxPower	Mode	Duration	Separated	Integrated
COVIDSafe	HIGH	LOW_LATENCY	∞	✔	–
Stopp Corona	HIGH	LOW_LATENCY	∞	✔	–
BeAware	MEDIUM	LOW_POWER	∞	✗	✗
CoronApp	HIGH	LOW_POWER	∞	✔	–
eRouska	MEDIUM	LOW_POWER	∞	–	✗
StopCovid	LOW	BALANCED	∞	✔	–
Aarogya Setu	ULTRA_LOW	LOW_POWER	∞	–	✗
MyTrace	LOW	BALANCED	∞	✗	✗
StopKorona	HIGH	BALANCED	∞	–	✗
Smittestopp	MEDIUM	LOW_POWER	∞	✔	–
Ehteraz	MEDIUM	BALANCED	∞	✗	✗
TraceTogether	HIGH	LOW_LATENCY	∞	✔	-
Mor Chana	MEDIUM	LOW_POWER	∞	–	✗
Hayat Eve Sigar	LOW	BALANCED	∞	✔	–
NHS COVID-19 App	MEDIUM	LOW_POWER	∞	✔	–
Healthy together	ULTRA_LOW	LOW_POWER	∞	✔	–
Bluezone	LOW	LOW_LATENCY	∞	✗	✗
CovidSafePaths	HIGH	LOW_LATENCY	∞	✔	–
Coalition network	HIGH	LOW_LATENCY	∞	–	✗
Covid community alert	HIGH	BALANCED	∞	✗	✗

Results. From 20 apps in our dataset, there are 12 apps that intend to broadcast infinitely until being enforced to close, while 8 have not specified their broadcasting duration. Given the default setting of this attribute is infinity [2], as presented in Table 1, all these apps will constantly broadcast advertising packets. In addition, we have identified that 10 apps have included *TxPower* separately in advertising packets, 5 apps are set to not carry this value in an individual field, and 5 apps have not specified this property. Moreover, neither the manufacturer nor service data fields in the latter 10 apps include *TxPower*.

Observation 1. *All apps in our analysis intend to broadcast advertising packets constantly without time limit and half of them have not included the level of TxPower in advertising packets.*

In terms of advertising interval, we have identified three apps—BeAware, eRouska, and Aarogya Setu—that have not explicitly specified their advertising interval (the default value is lower power mode [2] , while the remaining 17 apps have specified this attribute. Among these 17 apps, there are 6 apps that are set to broadcast advertising packet with minimum intervals, 6 apps that use the balanced mode, and 5 apps that apply the low power mode. Moreover, BeAware, eRouska, and Mor Chana are the only 3 apps that do not explicitly specify their level of *TxPower* in broadcasting (the default value medium will be used in this

case). For the remaining 17 apps that specify the *TxPower*, there are 8 apps that specify the highest transmission power, 3 apps that specify it to be medium, 4 apps that set themselves as low level, and 2 apps that apply the lowest level.

Surprisingly, from Table 1, we also observed that the combination of the level of *TxPower* and advertising interval is inconsistent across different apps. In particular, (*i*) among 8 apps that use the high level of *TxPower*, there are 5 apps that broadcast with the minimum interval, 2 apps with medium interval, and 1 app with the maximum interval; (*ii*) among 6 apps using the medium level of *TxPower*, we have identified that 5 apps are set with the maximum broadcasting interval and 1 apps is set with medium interval; (*iii*) for the 4 apps using low *TxPower*, 3 of them broadcast with medium interval and 1 app with the maximum interval; and (*iv*) the remaining 2 apps share the same combination of the lowest level of *TxPower* and the maximum interval.

In general, the combination of *TxPower* and advertising interval can impact the accuracy of *RSSI* value read at receivers [11]. However, in practice, we have not observed a consensus view toward this combination across contact tracing apps. Additionally, the magnitude of the impact on the *RSSI* value from different combinations remains unclear.

Observation 2. *The combination of the level of TxPower and advertising interval is inconsistent across contact tracing apps. Meanwhile, the impact on distance measurement from different combinations is also unclear.*

4 Analysis of Proximity Measurement Approaches

After analyzing the BLE software configurations, we next understand how each app measures the proximity. To this end, we first recognize which type of data is collected in Sect. 4.1, and then uncover how the collected data is used in the proximity measurement in Sect. 4.2.

4.1 Data Collected for Proximity Measurement

The first step to understand how each app measures the distance is to recognize which type of relevant data would be collected. Unfortunately, proximity measurements are rarely mentioned or vaguely expressed in the documentation (*e.g.*, app description and privacy policy) of an app. As such, we need to analyze the code of an app to understand the semantics of its collected data. However, identifying which one is used for proximity measurement is challenging since multiple types of data are processed within an app. Fortunately, we have observed a special feature in BLE-based contact tracing services that can narrow down the scope. That is, the *contact events* will be temporarily stored locally and all necessary data of each event will be stored together as an entry in a database or a local file. Therefore, we focus on the database or file operations, *e.g.*, read and write, of an app to uncover which type of data is collected for proximity measurement.

Table 2. Data collected for distance measurement (Note that ● represents collection).

App name	RSSI	Affecting factors		
		Software	Hardware	Others
COVIDSafe	●	Level of TxPower	modelP; modelC	
CoronApp	●	Level of TxPower	modelP; modelC	
eRouska	●			
StopCovid	●		BuildNumber; Version Manufacturer; Model	
Aarogya Setu	●	Level of TxPower		GPS
StopKorona	●			
Smittestopp	●	Level of TxPower		GPS, Altitude Speed, Accuracy
Ehteraz	●			GPS
TraceTogether	●	Level of TxPower	modelP; modelC	
Mor Chana	●			
NHS COVID-19 App	●	Level of TxPower		
Healthy together	●	Level of TxPower		
Bluezone	●	Level of TxPower		
CovidSafePaths	●	Level of TxPower		
Covid community alert	●		BuildNumber; Version Manufacturer; Model	
Coalition network	●			

Results. We have uncovered the data collected for proximity measurement from 16 apps (note that the rest 4 use native code and reflection to collect data, and we leave them in future work) and present them in Table 2. In addition, we have classified the uncovered data into the following three categories.

- ***RSSI***: Our analysis reveals that all 16 apps collect *RSSI* value. In addition, 4 of them collect this value exclusively and the remaining apps collect other types of data such as *TxPower* as well.
- **Affecting Factors**: Our analysis has identified that 9 apps collect the level of *TxPower* and 5 apps gather phone models. Specifically, among these 5 apps, 3 of them collect phone models of senders and receivers, which are required by *BlueTrace* protocol [4], and 2 apps only collect its own (receiver) phone model, which is needed by the AltBeacon library [1] for distance calculation.
- **Other Distance Measurements**: There are 3 apps even collecting GPS coordinates for distance measurement. In particular, unlike Ehteraz and Aarogya Setu that only collect GPS coordinates, Smittestopp also collects altitude, speed, and their degrees of accuracy.

Based on the uncovered data collection from these apps, it would be challenging to obtain an accurate distance. That is, only 5 apps have considered the affecting factors from hardware specifications but the number of specifications is limited. Moreover, 3 apps only use *RSSI* for distance calculation without considering the level of *TxPower*, and none of them considers external affecting factors from the environment, such as having a phone in a pocket.

Observation 3. *Data collection for proximity measurement is inconsistent across different contact tracing apps. Unfortunately, only a few of them consider the affecting factors from hardware specifications, e.g., phone models, some apps do not consider the affecting factors from software configurations, e.g., TxPower, and there is no evidence indicating that external affecting factors have been considered.*

4.2 Data Used in Distance Calculation

In addition to understanding the types of data collected for distance measurement, we also seek to know how such data is exactly used. In this regard, directly checking the formula used for distance calculation is a reliable solution. Because the distance measurement is based on *RSSI*, the formula must use this value in the calculation. As such, we can track the dataflow of *RSSI* value within mobile apps to discover the formula. We follow such an approach in our analysis.

Results. Uncovering this formula from contact tracing apps is challenging. First, in a centralized service, the formula is supposed to exist on the server side whose code is inaccessible to us. Additionally, in a decentralized service, it is also non-trivial to uncover the distance calculation formula from mobile apps because of a variety of obfuscations on the code (*e.g.*, 4 out 6 decentralized apps in our dataset use obfuscation), from variable and method renaming to using reflections. In the end, with our best effort, we have uncovered the distance calculation formula from three apps. Interestingly, they use the same distance measurement model as the following:

$$(\frac{RSSI}{TxPower})^{Coef_1} \times Coef_2 + Coef_3$$

where the three coefficients are used to tune the accuracy for different hardware specifications. Among the apps we analyzed, StopCovid and Covid Community Alert use the AltBeacon library [1] for proximity measurement whose coefficients for 4 phone models are available online[1].

Observation 4. *Different contact tracing apps may use the same formula to measure the proximity. However, their tuning is quite limited to only a few phone models or without tuning at all.*

5 Discussion

From our analysis, a large portion of BLE-based contact tracing apps do not have an accurate and reliable proximity measurement from Bluetooth.

A practical and effective solution to improve the accuracy could be tuning *RSSI* for different phone models, because different models provide a variety of

[1] https://s3.amazonaws.com/android-beacon-library/android-distance.json.

BLE hardware specifications and their impacts on the robustness of the *RSSI* values are significant. Fortunately, some groups (e.g., OpenTrace [6] have started conducting experiments and collecting data for this tuning. However, only a limited number of phone models have been involved. More efforts are needed to cover more phone models.

Additionally, we have identified a variety of advertising behaviors with different combinations of the advertising interval and the power of transmission. Unfortunately, it is unclear whether these different behaviors can impact the *RSSI* value as well as their corresponding magnitude of influence. Further studies in this direction could help identify an effective BLE advertising behavior that improves the robustness of *RSSI* values.

6 Related Work

Recently, there are multiple privacy-preserving contact tracing protocols having been proposed. Some of them [4,8] are centralized and some [3,15,19,23,28] are decentralized. Accordingly, there is also a body of research [13] analyzing the potential privacy issues in these protocols. In addition, many studies have focused on the analysis of COVID-19 themed apps. For instance, several studies [16,29] focus on privacy issues of one specific contact tracing app (*e.g.*, TraceTogether [16]), and the rest (*e.g.*, [18,25,27,30]) present empirical analysis with these apps. Similarly, there are also many efforts (*e.g.*, [26,32]) that focus on security issues in BLE mobile apps in general. Unlike these efforts that aim at analyzing privacy and security issues, we focus on the accuracy of proximity measurement in contact tracing apps.

7 Conclusion

To fight COVID-19 pandemic, a large number of BLE proximity tracing apps have been developed and deployed. These apps use the received signal strength indicator, *RSSI*, to measure the proximity between two smartphones. However, multiple factors can impact the *RSSI* value that makes the proximity measurement challenging. In this paper, we provide a detailed study on the accuracy of *RSSI*-based proximity measurements that are applied in 20 BLE-based contact tracing apps. Our study has revealed that different apps configure a variety of BLE broadcasting behaviors and only a small portion of them have tuned *RSSI* to improve the accuracy of measured proximity.

Acknowledgement. This work was supported in part by the National Science Foundation (NSF) under Grant No. CNS 1618520, CNS 1834215, and CNS 2028547. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. Android beacon library. https://altbeacon.github.io/android-beacon-library/index.html. Accessed 15 Aug 2020
2. Android open source project. https://source.android.com/. Accessed 15 Aug 2020
3. Apple and google partner on covid-19 contact tracing technology. https://www.blog.google/inside-google/company-announcements/apple-and-google-partner-covid-19-contact-tracing-technology/. Accessed 15 Aug 2020
4. Bluetrace. https://bluetrace.io/. Accessed 15 Aug 2020
5. Covid-19 apps - wikipedia. https://en.wikipedia.org/wiki/COVID-19_apps. Accessed 15 Aug 2020
6. Github - opentrace-community/opentrace-calibration: Opentrace calibration. device calibration data and trial methodologies for testing implementations of the bluetrace protocol. https://github.com/opentrace-community/opentrace-calibration/. Accessed 18 Aug 2020
7. Mimo - wikipedia. https://en.wikipedia.org/wiki/MIMO. Accessed 15 Aug 2020
8. Pepp-pt. https://www.pepp-pt.org/. Accessed 15 Aug 2020
9. Proximity and rssi – bluetooth®technology website. https://www.bluetooth.com/blog/proximity-and-rssi/. Accessed 15 Aug 2020
10. Transmission power, range and rssi - support center. https://support.kontakt.io/hc/en-gb/articles/201621521-Transmission-power-Range-and-RSSI. Accessed 15 Aug 2020
11. What are broadcasting power, rssi and other characteristics of a beacon's signal? - estimote community portal. https://community.estimote.com/hc/en-us/articles/201636913-What-are-Broadcasting-Power-RSSI-and-other-characteristics-of-a-beacon-s-signal-. Accessed 15 Aug 2020
12. Will wireless interference and wi-fi impact beacons? - estimote community portal. https://community.estimote.com/hc/en-us/articles/200794267-What-are-potential-sources-of-wireless-interference-. Accessed 15 Aug 2020
13. Baumgärtner, L., et al.: Mind the gap: Security & privacy risks of contact tracing apps. arXiv preprint arXiv:2006.05914 (2020)
14. CDC. Public health guidance for community-related exposure. https://www.cdc.gov/coronavirus/2019-ncov/php/public-health-recommendations.html. Accessed 15 Aug 2020
15. Chan, J., et al.: Pact: privacy sensitive protocols and mechanisms for mobile contact tracing. ArXiv arXiv:2004.03544 (2020)
16. Cho, H., Ippolito, D., Yu, Y.W.: Contact tracing mobile apps for covid-19: privacy considerations and related trade-offs. arXiv preprint arXiv:2003.11511 (2020)
17. Crocker, A., Opsahl, K., Cyphers, B.: The challenge of proximity apps for covid-19 contact tracing—electronic frontier foundation (2020). https://www.eff.org/deeplinks/2020/04/challenge-proximity-apps-covid-19-contact-tracing. Accessed 15 Aug 2020
18. Li, J., Guo, X.: Covid-19 contact-tracing apps: A survey on the global deployment and challenges. arXiv preprint arXiv:2005.03599 (2020)
19. Niyogi, S., et al.: Tcncoalition/tcn: Specification and reference implementation of the tcn protocol for decentralized, privacy-preserving contact tracing (2020). https://github.com/TCNCoalition/TCN. Accessed 15 Aug 2020
20. Government of Singapore. Trace together, safer together (2020). https://www.tracetogether.gov.sg. Accessed 15 Aug 2020

21. O'Neill, P.H.: Bluetooth contact tracing needs bigger, better data—mit technology review (2020). https://www.technologyreview.com/2020/04/22/1000353/bluetooth-contact-tracing-needs-bigger-better-data/. Accessed 15 Aug 2020
22. O'Neill, P.H., Ryan-Mosley,T., Johnson, B.: A flood of coronavirus apps are tracking us. now it's time to keep track of them.—mit technology review (2020). https://www.technologyreview.com/2020/05/07/1000961/launching-mittr-covid-tracing-tracker/. Accessed 15 Aug 2020
23. Rivest, R.L., et al.: The pact protocol specification (2020). https://pact.mit.edu/wp-content/uploads/2020/04/The-PACT-protocol-specification-ver-0.1.pdf. Accessed 15 Aug 2020
24. ROBERT ROBust and privacy-presERving proximity Tracing protocol (2020). https://github.com/ROBERT-proximity-tracing. Accessed 12 May 2020
25. Simko, L., Calo, R., Roesner, F., Kohno, T.: Covid-19 contact tracing and privacy: Studying opinion and preferences (2020). arXiv preprint arXiv:2005.06056
26. Sivakumuran, P., Blasco, J.: A study of the feasibility of co-located app attacks against BLE and a large scale analysis of the current application layer security landscape. In: 28th USENIX Security Symposium. USENIX Association. USENIX Sec, Santa Clara (2019)
27. Tang, Q.: Privacy-preserving contact tracing: current solutions and open questions. arXiv preprint arXiv:2004.06818 (2020)
28. Troncoso, C., et al.: Decentralized privacy-preserving proximity tracing (2020). https://github.com/DP3T/documents. Accessed 15 Aug 2020
29. Veale, M.: Analysis of the NHSX contact tracing app 'isle of wight' data protection impact assessment (2020)
30. Wen, H., Zhao, Q., Lin, Z., Xuan, D., Shroff, N.: A study of the privacy of covid-19 contact tracing apps. In: International Conference on Security and Privacy in Communication Networks (2020)
31. Woodhams, S.: Covid-19 digital rights tracker (2020). https://www.top10vpn.com/research/investigations/covid-19-digital-rights-tracker/. Accessed 15 Aug 2020
32. Zuo, C., Wen, H., Lin, Z., Zhang, Y.: Automatic fingerprinting of vulnerable BLE IoT devices with static uuids from mobile apps. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019)