A Survey of COVID-19 Contact Tracing Apps

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Sean Burtner

May 10, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Sean Burtner

Technical advisors: Miaomiao Zhang and Yuan Tian, Department of Computer Science

A Survey of COVID-19 Contact Tracing Apps

Analysis of Research

Sean Burtner
University of Virginia
seb4jw@virginia.edu

ABSTRACT

The COVID-19 pandemic has sparked a variety of technological interventions, including smartphone contact tracing apps. These apps help notify the recent contacts of infected individuals, but they introduce many security and design concerns. This research will primarily analyze a 2020 IEEE article about this technology titled "A Survey of COVID-19 Contact Tracing Apps," by Ahmed et al.; the paper examines different system architectures, privacy considerations, potential attacks, and examples of existing contact tracing apps. Analyzing this research will expose unanswered questions and insights to guide future development of contact tracing technology. Additionally, the article's analysis of contact tracing architecture will be applied to the Virginia statewide contact tracing app (COVIDWISE). Promoting an understanding of the security and privacy implications of the COVIDWISE app architecture will encourage wider adoption of the technology. Since the efficacy of contact tracing apps relies on a sufficient adoption rate, increasing the public's trust in the technology is imperative.

1 Introduction

This work will summarize and analyze a 2020 article titled "A Survey of COVID-19 Contact Tracing Apps," which seeks to provide a comprehensive review of this technology [1]. The study primarily examines three broad tracing app architectures, their security and privacy implications, possible attacks and vulnerabilities, and example implementations of the technology. Analyzing this research will expose unanswered questions, deficiencies, and insights to promote further development of digital contact tracing. Additionally, summarizing the study will clarify the technology in hopes of mitigating privacy concerns and dispelling public misconception. This work also aims to extend the article's survey of contact tracing app implementations to the Virginia statewide tracing app called COVIDWISE. The article does not include COVIDWISE in its overview of example implementations; promoting an understanding of the security and privacy implications of the COVIDWISE app architecture will encourage wider adoption of the technology in Virginia.

1.1 Background

The Severe Acute Respiratory Syndrome Coronavirus 2 pandemic, otherwise known as COVID-19, has strained public health systems worldwide. The high infectivity rate of COVID-19 combined with asymptomatic spread has demanded rigorous pandemic response measures. In order to control transmission, a process known as contact tracing is often implemented. This process involves determining the close contacts of a newly identified case of infection, and subsequently notifying these contacts of their increased risk. Historically, contact tracing is conducted manually; that is, health authorities will interview infected individuals to collect information about their whereabouts and close contacts in the previous weeks. Health authorities then manually notify at-risk individuals, based on the proximity and duration of contact with the infected individual. However, a significant limitation of this process is its reliance on people to manually recall exactly who they contacted in the recent weeks.

To address such limitations, governments have looked to digital contact tracing solutions to automate the contact tracing process. This technology leverages the widespread possession of smartphones and their ability to interface with nearby devices via Bluetooth. The contact tracing apps passively collect contact data based on proximity, which can be used to more reliably identify at-risk individuals. With a sufficient adoption rate, this technology is capable of significantly mitigating disease spread [2]. However, as with most data aggregation technologies, adoption is hindered by concerns over privacy and security. System architecture and data management dictate the extent and nature of security vulnerabilities; as a result, it is crucial to understand the tradeoffs of different contact tracing app architectures.

2 Analysis of Research

This work is organized as follows. Section 2.1 examines the article's analysis of the three categories of app architectures: centralized, decentralized, and hybrid. Section 2.2 discusses the article's analysis of the security and privacy concerns associated with each architecture, and Section 2.3 reviews the article's examination of example implementations of contact tracing technology. Questions, insights, and gaps in analysis are discussed at the end of each section. Finally, Section 3 applies the work to the COVIDWISE app deployed by the Virginia Department of Health (VDH).

2.1 Architectures

The three classifications of system architectures pertain to the role of the server in processing and managing data; the centralized architecture handles much of the data at the server, the decentralized architecture much less so, and the hybrid architecture splits this role between the user device and the server.

2.1.1 Centralized Architecture. In the centralized architecture, the server is assumed trusted. It requires that users pre-register with the server, inputting personally identifiable information (PII) and verifying a phone number. Once verified, the user receives a Temporary ID (TID), which is encrypted and stored by the server. The secret key used for encryption is only known by the central server. These TIDs are only valid for a short period of time, typically 15 minutes, and are fundamental to the contact tracing process in this architecture.

Upon contact with another app user, an encounter message is exchanged via Bluetooth. This message contains the user's TID, phone model, and transmit power; each device also stores the Received Signal Strength Indicator (RSSI) and timestamp of the encounter. Therefore, between these two devices, privacy is preserved, as TIDs can only be mapped to specific users by the central server. These encounter messages are stored in each user's local storage, invisible to the server. Furthermore, a blacklist of recent contacts is maintained in order to prevent duplicate contacts from registering repeatedly in a short time.

If a user tests positive for COVID-19, they may voluntarily upload their list of encounter messages, which contain other users' TIDs, to the server. A health official must first confirm if the individual has the app, and if the user agrees to upload their data, the official sends the user a one-time password (OTP) to authorize the upload. After data is uploaded, the server maps the TIDs in the encounter messages to users' phone numbers and forwards this list to a health authority, who then notifies at-risk users. In creating this list, the server uses the transmit power and RSSI value recorded in each encounter message to estimate the risk profile of each encounter.

2.1.2 Decentralized Architecture. In contrast to the centralized architecture, the decentralized architecture moves core functionalities from the server to the end users. The main difference is its use of anonymous identifiers called 'chirps', which conceals identity from both other users and the server. In this implementation, no pre-registration with the server is required, and therefore no PII is stored by the server. The app simply verifies the user's device, and deploys a random seed generation algorithm to prepare for the creation of chirps.

Instead of TIDs, chirps are exchange during encounters with other app users. These privacy-preserving pseudonyms are generated using a random seed and the current time as input to a pseudorandom function. While seeds are randomly generated every hour, new chirps are generated every minute, and are broadcast via Bluetooth. When nearby devices receive a chirp, they are stored locally, along with the timestamp and the RSSI value. Identical chirps received within the same minute are not recorded, akin to the blacklist used in the centralized architecture.

Upon testing positive with COVID-19, a user must receive a unique permission number from a health authority to authorize upload of all of their seeds from the last 21 days, along with the associated creation and expiry times of the seeds. This is in heavy contrast with the centralized architecture, where a full list of other users' TIDs is uploaded; the decentralized architecture does not expose any information about other users to the server, as it only receives a single infected user's seeds. The contact tracing phase also differs significantly in that it takes place on the user's device rather than the server. Once a day, users can download the seeds and timestamps uploaded by infected users, and reconstruct every chirp. Then, it compares these reconstructed chirps with its recorded chirps to search for matches. If a match is found, risk analysis is conducted based on the RSSI value, and the user is notified.

2.1.3 Hybrid Architecture. In the hybrid architecture, core functionalities are split between the server and the devices. Devices generate and manage pseudonyms similar to chirps, but the responsibility of risk analysis and notification is transferred to the server. This provides benefit as the server can run statistical analysis to identify exposure centers, which is not possible in the decentralized architecture.

Upon registration, devices are sent a unique ID and encryption key, which is deleted from the server. Devices create a TID approximately every 15 minutes, synchronized with the Bluetooth MAC address rotation. This prevents a carryover attack, which is discussed in Section 2.2.2. These TIDs are broadcast via Bluetooth; once a device receives a TID, two Private Encounter Tokens (PETs) are generated using both exchanged TIDs as input. PETs are stored in two different tables, referred to as a query table and an upload table. When a user tests positive for COVID-19, they voluntarily upload their ID, encryption key, and PETs from the upload table. The server is able to use the encryption key to locate the user record, and associate the uploaded PETs with this user. Contact tracing is then conducted when another user uploads the PETs in their query table, allowing the server to match these with any PETs uploaded by the infected user. The server can subsequently send a notification to any at-risk users based on the time and duration values of the encounter. It's important to note that anonymity is maintained as the server cannot identify a user solely via their PETs.

2.1.4 Analysis. In regard to the centralized architecture, the article mentions that TIDs are generated by the central server and sent to the user's device; however, this would necessitate that the user constantly maintains an Internet connection in order to receive the next TID every 15 minutes. Such a constraint is an important aspect of the centralized architecture that must be considered in its evaluation, but the article fails to discuss this. Furthermore, the article mentions a blacklist that keeps track of recent contacts to prevent duplicate encounters from being recorded. However, it fails to discuss how this blacklist functions. A potential issue could occur where a user has an initial weak contact with another user, but then a much stronger, risker contact shortly after that does not get recorded. Improvement in this area could be explored, such that weak encounters are replaced with stronger encounters, despite being blacklisted.

The centralized architecture has the potential to be heavily optimized, which is not discussed by the article. After an infected user is identified, numerous steps must occur before at-risk users are notified: the health official must contact the user, then the user must agree to upload their data, receive an OTP, and upload their data, then server must map TIDs to at-risk users, and finally the health official must notify the at-risk users. Most of this overhead stems from the fact that the user must provide consent to upload the data, which could be eliminated if the user agreed to upload encounter data at the time of installation. After all, downloading the app demonstrates the intent of the user to engage in the contact tracing process, so explicit agreement earlier in the process would not significantly deter users. In a time-sensitive application such as contact tracing, optimizations must be made wherever possible, and such an opportunity exists in this aspect of the centralized architecture. Additionally, this optimization would address the potential issue of additional encounters occurring after the original encounter data is uploaded; the article fails to discuss how the user can continue to upload new encounter data after testing positive.

Another gap in its discussion of the centralized architecture is its failure to consider how encounter data is filtered from the past. Based on the article's analysis, it's implied that an infected user's full history of encounters is uploaded to the server for processing. This imposes increased computational load on the server, as it must decrypt each TID, match this to a user's phone number, and conduct risk analysis to determine the severity of the encounter. The list of relevant encounters is then sent to a health official for further processing. If encounters only from the last 14-21 days were uploaded, computational load could be reduced, and the amount of manual processing by the health official could be reduced.

The decentralized architecture also presents a potential for optimization. Processing of infected users' seeds occurs locally on the device. The app must reconstruct all 60 chirps that correspond with each hourly seed. If infected users upload 21 days' worth of data, with 24 seeds per day and 60 chirps per seed, that comes out to over 30,000 chirp calculations per case. If the app were to achieve successful rates of adoption, it would need handle tens of thousands of new cases per day; all of these chirp calculations would need to be processed by the user's device. Not only that, but for each chirp, it must conduct a search in its entire log of recorded chirps in order to look for a match. Therefore, the device is expected to handle an immense amount of computation in the background. The article fails to address this lack of efficiency, or how it could be optimized. The number of computations and searches could be reduced significantly with a minimal amount of location information. For a user that hasn't travelled, calculating the chirps for a user across the country is certainly unnecessary. Additionally, much of the computation could be moved to the server, while still retaining the privacy benefits of the decentralized architecture. Infected users' uploaded seeds and timestamps could be converted into chirps on the server-side, and other users could simply cross-reference their locally recorded chirps with these chirps. This spares processing power and battery for the

end user's device, without compromising efficiency or privacy.

2.2 Security, Privacy, and Attacks

2.2.1 Architectural Differences. The data management practices employed by each architecture directly determine the security and privacy implications of the implementation. In the centralized architecture, almost all of the data is managed by the central server, so a compromised server would allow a malicious user to identify all users and their contacts. Thus, in considering attacks on the centralized architecture, the server is assumed trusted, as it can readily access PII. This necessitates strong security, authentication, and access control mechanisms with regard to the server. Data exchange between the server and the devices must also be secure. These points of vulnerability are subject to attack; for example, the lack of authentication in Bluetooth message exchange could be exploited by attackers in a replay or relay attack, discussed in section 2.2.2.

The decentralized architecture allows users to access the public server in order to download seeds and metadata of infected individuals. Since this metadata includes timestamps, unauthorized identification of infected users is possible by exploiting "side-channel information." The attack model assumes an honest-but-curious server in this architecture, as they are built around the concern of data misuse by authorities. This architecture is generally well-protected from a single point of failure as data management is primarily on the user side, and any data stored on the server is already public.

The hybrid architecture combines the strengths of both the centralized and decentralized architectures, as the server has sufficient data to run statistical analysis on disease spread, but cannot identify users with this data alone. Thus, anonymity is maintained, while the computational load is effectively distributed. Sensitive information is not exposed even if one party, either the server or the user, is compromised.

2.2.2 Specific Attacks. Depending on app architecture, numerous possible attacks must be considered. In replay or relay attacks, an adversary seeks to create false positives in the contact tracing system by forwarding Bluetooth messages between two unencountered users. The message can be captured by the adversary and immediately relayed to users in a broader area that would otherwise have not received the message – this is known as a relay attack. In a replay attack, the message could be re-broadcast in a completely different location. This particular attack is mitigated by the expiry times associated with TIDs and chirps, but cannot be neglected.

The fundamental role of Bluetooth technology in any contact tracing implementation further invites attack in the form of wireless device tracking. In this attack, an adversary could track a device using numerous Bluetooth receivers. Since Bluetooth messages are constantly broadcast in each architecture, passive collection of signals could allow for reasonably accurate location estimation of the user. The centralized and hybrid architectures are more vulnerable to this attack as TIDs expire much less often than chirps in the decentralized architecture. A similar attack is simple location confirmation in some environment. An adversary, with previously collected information about a user's phone model, could confirm the user's presence via Bluetooth merely by listening to encounter messages. However, this is only possible in the centralized architecture.

In an enumeration attack, the goal of the adversary is to count infected users. The hybrid and centralized architectures store the requisite information on the server, so users could not launch this attack without infiltrating the server. In the decentralized architecture, since only infected users upload their seeds, an adversary could download this public data and estimate the infection rate based on the fixed number of seeds uploaded by each user.

A denial-of-service attack attempts to consume any possible resources in either user devices or the server. This includes battery, processing power, and bandwidth. This simple attack is accomplished by injecting phony encounter messages or chirps into the Bluetooth listening environment, which consumes resources on the user devices, regardless of architecture. User devices suffer the most in the decentralized architecture, since all chirps, even the bogus ones, must be compared against the reconstructed chirps of infected users. A denial-of-service attack in the centralized or hybrid architectures consumes server resources as bogus encounter messages are still uploaded and processed by the server.

In a linkage attack, adversaries attempt to de-anonymize users by matching broadcast messages with side-channel information in order to link pseudonyms with a user's identity. The decentralized architecture is most vulnerable as seed information is publicly available to all users, while the centralized architecture is more resilient since TIDs of infected users are not shared by the server. However, TIDs could potentially be associated with a particular user based on the broadcast phone model. A specific linkage attack known as a Paparazzi attack could be launched against decentralized contact tracing apps; this attack involves a

malicious user deploying passive Bluetooth devices to collect chirps, and when a user tests positive, the adversary could compare the reconstructed chirps against the aggregated data to trace user movements throughout the contagion period.

Carryover attacks are used to augment wireless device tracking attacks by taking advantage of out-of-sync Bluetooth MAC address rotation and TID rotation. If they are not synchronized, the Bluetooth MAC address can be linked to two consecutive TIDs, nullifying the attempt to anonymize them. Once a link has been identified, the MAC address and the TID can be used to de-anonymize each other, until their rotations fall back into sync. This allows for extended wireless tracking beyond the TID expiry time.

The final major attack considered in contact tracing app architectures is the disclosure of a social graph. A social graph depicts which users have been in proximity with other users, built by collecting and analyzing encounter data to infer contacts. This attack primarily considers malicious server management as the potential adversary, as users in any architecture lack access to the data needed for constructing a social graph. Only a partial social graph can be created in the decentralized architecture, while a more complete one can be created in a centralized app as the server possesses the mappings between TIDs and user identities. However, this attack is significantly limited by the fact that interactions cannot be tracked between users who neither tested positive nor have encounter an infected user.

2.2.3 Analysis. The major weakness of the centralized architecture is its susceptibility to a single point of failure. Its accumulation of responsibilities leaves it vulnerable, which is in sharp contrast to the other two architectures. The decentralized architecture presents its own vulnerabilities, which can be attributed to its full accessibility of server data. The article further attributes this to the potential exploitation of "side-channel information" by a malicious user, but only generally alludes to a separate app used for recording encounter details. Further exploration could be conducted on this aspect of the vulnerability and the ability of malicious users to use publicly available seed data and temporal metadata to de-anonymize users.

Another flaw in its discussion of security vulnerabilities relates to a traffic analysis attack. In this attack, a malicious user eavesdrops on server traffic in order to de-anonymize infected users; the article pins this as a weakness of the decentralized architecture, yet this is a potential issue in all three architectures. Each design is built around the voluntary upload of infected user data to the server, and is therefore susceptible to this attack. This vulnerability necessitates secure data transfer in any implementation of the technology, not solely in the decentralized design.

Furthermore, the discussion of the enumeration attack makes an oversight; in the decentralized architecture, it claims that this attack can be mitigated if an infected user could "redact some contact information while uploading their contacts." However, in this architecture, infected users do not upload contacts, but rather their seeds and associated metadata from the last 21 days. The decentralized architecture's defense against enumeration should be revisited, as seed data could still be exploited to launch an enumeration attack. The number of seeds uploaded by infected users could potentially be varied rather than fixed in order to better defend against enumeration by unauthorized users**.**

2.3 Example Implementations

The article proceeds to examine numerous example implementations of contact tracing apps around the world. This will be briefly summarized, and extended to the Virginia statewide tracing app known as COVIDWISE in Section 3.

2.3.1 Apps. Four apps based on the centralized architecture are discussed. The TraceTogether and CovidSafe apps are both similar to the base implementation described in Section 2.1.1, though differ in TID handling [3] [4]. TraceTogether supplies forward-dated TIDs to ensure devices have a steady supply of TIDs without having to contact the server; these are valid for 15 minutes as discussed earlier, while CovidSafe uses an expiry time of 2 hours. This makes it more vulnerable to the replay attack discussed in Section 2.2.2. The StopCovid app differs more in that PII is not stored at the server, and users must proactively check with the server to see if they have been flagged at-risk [5]. Aarogya Setu is an India-based app that additionally uses GPS to determine the number of infected users relative to a user's geographic location [6].

Several decentralized contact tracing designs are discussed. Another app called CovidSafe improves upon the base architecture by cutting down on the number of stored seeds while still rotating them at comparable intervals [7]. A European implementation called SwissCovid is very similar to the base architecture, but filters out irrelevant seeds from before the window of infectivity to cut down on computation [8]. Another implementation called DP-3T Un-linkable protects against the linkage and enumeration attacks discussed in Section 2.2.2, as it hashes uploaded seeds before advertising them to other users [9]. To check for risk, users must hash any

received chirps with the time of reception and reference the server data. CovidWatch and Pronto-C2 are other decentralized apps that vary seed generation protocols in order to protect against linkage attacks and surveillance [10] [11]. Hamagen and Covid Safe Paths are notable implementations in that they do not use Bluetooth to record encounter data at all; they solely rely on GPS location history on a user's device to determine if a user was in contact with infected individuals [12] [13].

Two protocols beyond the base implementation of the hybrid architecture are discussed. The ConTra Corona protocol utilizes several servers to protect against a single point of failure [14]. Though the implementation details differ significantly from the base implementation, the division of responsibilities between the devices and the servers are the same. EpiOne is another protocol that applies cryptography, namely Private Set Intersection, to ensure that neither the user nor server can obtain the complete set of TIDs or seeds [15]. This protects against linkage and social graph analysis attacks.

2.3.2 Analysis. The use of forward-dated TIDs is a strong addition to the base centralized architecture, as it relieves the significant constraint of maintaining a consistent Internet connection. Additionally, the StopCovid app could be argued to belong in the hybrid architecture classification, as it moves some functionality away from the central server, namely risk notification. Its removal of PII from the server is also reminiscent of the hybrid architectural analysis. The same can be argued for the DP-3T Un-linkable app, as it migrates risk checks to the user by having them hash data locally and check it with the server data.

Aarogya Setu interestingly uses GPS to augment the contact tracing process, though the article fails to discuss the strong privacy concerns associated with the use of location data in this technology. This could lead to significant resistance in app adoption due to user concern. In the extreme case of solely relying on GPS, as seen in the Hamagen and Covid Safe Paths implementations, user concerns over privacy would be the primary obstacle to adoption. Further investigation into how Israel promoted the Hamagen app while satisfying demands for location privacy is warranted, as these lessons could be generalized and applied to the adoption phase of other data aggregation technologies.

3 COVIDWISE

The COVIDWISE app was developed and released by the VDH, and employs exposure notification APIs created by Apple and Google [16] [17]. It matches most closely with the decentralized architecture discussed in Section 2.1.2. Devices are assigned a daily anonymous key, which is exchanged with nearby devices upon encounter. Along with the key, contact metadata such as time, duration, and the RSSI value are recorded. All of this is recorded locally, and no PII is stored on the server.

Upon testing positive with COVID-19, a user will upload the anonymous keys used in the last 14 days to the server. This is a voluntary process, as a positive test result must first be uploaded and confirmed by the VDH. Other users can download these keys, and reference them against their recorded keys from various encounters. If a match is found, risk is assessed using the contact metadata, and the user is notified. Notifications are only sent if the encounter was estimated to be within six feet, and for at least 15 minutes in a 24-hour period.

One potential area of improvement is a mechanism for allowing continual upload of keys after diagnosis. In the current implementation, only keys from the day of diagnosis, and 14 days prior, are uploaded. In other words, it neglects the infectivity of positive cases as they are only considered infective for the two weeks prior to diagnosis. Infectivity certainly remains a concern past this date, and therefore demands a mechanism for infected individuals to continue uploading keys.

Additionally, one of the strengths of the base decentralized architecture is its resiliency to a replay attack, as well as wireless device tracking (see Section 2.2.2). With a chirp expiry time of 1 minute, a replay attack is more difficult to launch, as the device will reject an expired chirp. However, the COVIDWISE app is significantly more susceptible to the replay attack as keys are only rotated once a day. An attacker could capture a broadcast key, and replay it in a completely different location on the same day. However, when comparing daily keys versus minute-long chirps, it's worth noting that there is a drastic difference in uploaded data volume and subsequent computation. In the base decentralized architecture, if an infected user uploads 14 days of seeds, over 20,000 chirps must be computed. Compared with the 14 daily keys needed in the COVIDWISE implementation, efficiency is a clear tradeoff for increased security against replay attacks and device tracking.

4 Conclusions

Contact tracing technology has the potential to significantly limit disease transmission. However, sufficient uptake is critical to its success, and it is often hindered by privacy and security concerns. This work promotes a greater understanding of the various app architectures and their

security implications in order to dispel unwarranted concern.

Each architecture and subsequent implementation present a unique attack surface, but these attacks generally pose a limited threat to the average user. For example, a common user concern relates to abuse of data by those managing the technology, particularly via de-anonymization. This is infeasible in both decentralized and hybrid architectures due to the extensive measures to preserve anonymity, and these two architectural classifications cover many of the existing app implementations today. App implementations also stress the importance of allowing users to voluntarily engage with the app, and stop at any time. This further enables the user to maintain control over their personal privacy. Measures like these must be clarified and communicated in order to encourage adoption of the technology. In general, contact tracing technology is not particularly vulnerable to attack or data misuse such as tracking. This is especially true in light of other apps that readily harvest location data. Dispelling this misconception is vital in promoting the technology and thereby aiding pandemic response.

4.1 Future Work

See Analysis Sections 2.1.4, 2.2.3, and 2.3.2 for specific areas to expand on. More generally, there is always opportunity to further explore security vulnerabilities and attacks. This is extremely important in the centralized architecture where the server maintains PII. Additionally, accuracy in proximity analysis is another significant area for improvement. The potential for false positives can be mitigated greatly with accurate proximity determination. Obstructions and orientation limit the ability of Bluetooth signals to properly identify relevant exposures, as it was not designed for such an application. Exploring new protocols for exchanging encounter messages could improve contact tracing technology markedly.

ACKNOWLEDGMENTS

Thank you Professor Miaomiao Zhang and Professor Yuan Tian for reviewing this report. Thank you Professor Aaron Bloomfield for guiding the overall capstone process.

REFERENCES

[1] Ahmed, N., Michelin, R. A., Xue, W., Ruj, S., Malaney, R., Kanhere, S. S., Seneviratne, A., Hu, W., Janicke, H., & Jha, S. K. (2020). A survey of COVID-19 contact tracing apps. *IEEE Access, 8*, 134577-134601. http://doi.org/10.1109/access.2020.3010226

[2] Ferretti, L., Wymant, C., Kendall, M., Zhao, L., Nurtay, A., Abeler-Dörner, L., Parker, M., Bonsall, D., & Fraser, C. (2020). Quantifying SARS-CoV-2 transmission suggests epidemic control with digital contact tracing. *Science, 368*(6491). http://doi.org/10.1126/science.abb6936

[3] Singapore Ministry of Health. (2020). TraceTogether. Gov.sg. https://www.tracetogether.gov.sg/

[4] Australian Government Department of Health. (2020). COVIDSafe app. Commonwealth of Australia. https://www.health.gov.au/resources/apps-and-tools/covidsafe-app

[5] French Department of Public Health. (2020). TousAntiCovid. Google Play Store. https://play.google.com/store/apps/details?id=fr.gouv.android.stopcovid&hl=en_US&gl=US

[6] Government of India. (2020). Aarogya Setu Mobile App. MyGov. https://www.mygov.in/aarogya-setu-app/

[7] University of Washington. (2020). CovidSafe Project. University of Washington. https://covidsafe.cs.washington.edu/

[8] Swiss Federal Office of Public Health. (2021). Coronavirus: SwissCovid app and contact tracing. Federal Office of Public Health. https://www.bag.admin.ch/bag/en/home/krankheiten/ausbrueche-epidemien-pandemien/aktuelle-ausbrueche-epidemien/novel-cov/swisscovid-app-und-contact-tracing.html

[9] Troncoso et al. (2020). Decentralized Privacy-Preserving Proximity Tracing. Cornell University. https://arxiv.org/abs/2005.12273

[10] WeHealth Solutions. (2021). Covid Watch Arizona. WeHealth. wehealth.org/arizona

[11] Avitabile, G., Botta, V., Iovino, V., & Visconti, I. (2020). Towards defeating mass surveillance and SARS-CoV-2: The Pronto-C2 fully decentralized automatic contact tracing system. International Association for Cryptologic Research. https://eprint.iacr.org/2020/493.pdf

[12] Israeli Ministry of Health. (2020). HaMagen 2.0. Ministry of Health. https://govextra.gov.il/ministry-of-health/hamagen-app/download-en/

[13] Massachusetts Institute of Technology. (n.d.). Project Safe Paths. MIT Media Lab. https://www.media.mit.edu/projects/safepaths/overview/

[14] Beskorovajnov, W., Dorre, F., Hartung, G., Koch, A., Muller-Quade, J., & Strufe, T. (2020). ConTra Corona: Contact tracing against the Coronavirus by bridging the centralized-decentralized divide for stronger privacy. International Association for Cryptologic Research. https://eprint.iacr.org/2020/505.pdf

[15] Trieu, N., Shehata, K., Saxena, P., Shokri, R., & Song, D. (2020). Epione: Lightweight contact tracing with strong privacy. Cornell University. https://arxiv.org/abs/2004.13293

[16] Virginia Department of Health. (2021). COVIDWISE. Virginia Department of Health. https://www.vdh.virginia.gov/covidwise/

[17] Google. (2020). Exposure notifications: Using technology to help public health authorities fight COVID-19. https://www.google.com/covid19/exposurenotifications/