Master Thesis

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Security and Privacy Aspects of Digital Contact Tracing

Filipp Roos
Department of Computer Science
Chair of Computer Science II (Secure Software Systems)

Prof. Dr.-Ing. Alexandra Dmitrienko
Reviewer/Advisor

Prof. Dr.-Ing. Samuel Kounev
Second Reviewer

Submission
6. October 2021

www.uni-wuerzburg.de

Acknowledgements

I am truly thankful to my advisor, Prof. Dr.-Ing. Alexandra Dmitrienko, for her guidance, support and patience throughout the research project and thesis writing process. Special thanks to Thien Nguyen for the discussions and feedback about the security analysis, as well as the rest of the TraceCORONA research and prototyping team at TU Darmstadt, which I am very grateful to have been a part of: Prof. Dr.-Ing. Ahmad-Reza Sadeghi, Dr.-Ing. Markus Miettinen, Joshua Kühlberg, Anel Muhamedagic and Richard Mitev. Without these people this work would not have been possible.

Additionally, I would like to express gratitude to my parents, who have encouraged me to never give up, as well as the friends and family members who have helped with proofreading the thesis.

Abstract

Contact tracing apps based on Bluetooth LE are used in the COVID-19 pandemic to automatically track virus transmissions. At the beginning of the pandemic in early 2020, a large amount of proposals for automated digital contact tracing were developed, with Google and Apple's Exposure Notifications (GAEN) being the most widely-used. This thesis explores and compares the security and privacy aspects of GAEN and TraceCORONA, a protocol developed at TU Darmstadt.

We co-develop the TraceCORONA Android app together with the development team from TU Darmstadt. Furthermore, we compare it to GAEN as implemented in the German Corona-Warn-App. For this purpose, we analyze the design and implementation of both systems, revealing and reporting multiple vulnerabilities in Corona-Warn. We perform an informal and formal security analysis of both protocols, revealing that TraceCORONA is more secure against known attacks and discovering a novel attack. Finally, we prove that TraceCORONA, although generating more network traffic, can reasonably be used in a national application scenario on the scale of Germany.

Zusammenfassung

„Corona-Apps“, die auf Bluetooth LE basieren, werden in der COVID-19-Pandemie verwendet, um automatisch die Gefahr der Ansteckung zu bestimmen. Zu Beginn der Pandemie im Frühjahr 2020 wurden eine große Anzahl Vorschläge für automatisierte digitale Kontaktnachverfolgung entwickelt. Die am weitesten verbreitete Lösung ist „Exposure Notifications“ von Apple und Google (GAEN). In dieser Arbeit werden die Sicherheit und der Datenschutz von GAEN und TraceCORONA, welches an der TU Darmstadt entwickelt wurde, untersucht und miteinander verglichen.

Die TraceCORONA-App wird zusammen mit dem Team der TU Darmstadt prototypisch auf Android entwickelt. Daraufhin wird diese mit der Corona-Warn-App, die auf GAEN basiert, verglichen. Hierfür wird der Entwurf und die Umsetzung der beiden Systeme analysiert, wobei mehrere Schwachstellen in Corona-Warn gefunden und gemeldet werden. Eine informelle und formelle Untersuchung der Sicherheitsaspekte beider Protokolle wird erstellt, mit dem Ergebnis, dass TraceCORONA sicherer gegenüber bekannten Angriffen ist. Zudem wird ein neuer Angriffsvektor entdeckt und analysiert. Obwohl TraceCORONA mehr Datenverkehr im Internet erzeugt, kann es sinnvoll in einer nationalen Anwendung in der Größenordnung von Deutschland eingesetzt werden.

Contents

1. Introduction **1**

2. Background **5**
2.1. Bluetooth Low Energy . . . 5
2.1.1. Addressing and Link Layer . . . 5
2.1.2. Attribute-Based Communication . . . 6
2.2. Technical Aspects of Bluetooth Contact Tracing . . . 7
2.2.1. Scanning and Advertising Intervals, Rotating Identifiers . . . 7
2.2.2. Contact Establishment, Matching and Exposure Notification . . . 7
2.2.3. Proximity Approximation and Transmission Risk Calculation . . . 8
2.2.4. Shifting and Padding of Transmitted Keys . . . 9
2.2.5. Background Services on iOS and Android . . . 9
2.3. Tradeoffs Between Efficiency and Privacy . . . 10
2.4. The TraceCORONA Protocol . . . 11
2.4.1. Bluetooth LE Handshake . . . 12
2.4.2. Elliptic Curve Diffie-Hellman (ECDH) Token Establishment . . . 12
2.4.3. Infection Verification . . . 14
2.4.4. Encounter Token Upload . . . 14
2.4.5. Encounter Token Download . . . 14
2.4.6. Matching . . . 14

3. Related Work **17**
3.1. Existing Contact Tracing Approaches . . . 17
3.1.1. Beginnings of Mobile Contact Tracing . . . 17
3.1.2. Centralized Approaches . . . 18
3.1.3. Decentralized Approaches . . . 19
3.1.4. Google and Apple Exposure Notifications . . . 21
3.2. Analysis of Contact Tracing Approaches . . . 21
3.2.1. Lists, Comparisons, Surveys, and Indexes . . . 22
3.2.2. GAEN and Corona-Warn-App Analysis Tools . . . 22
3.3. Attacks on Contact Tracing . . . 22
3.3.1. Profiling Using Bluetooth Sniffing . . . 22
3.3.2. Fake Alert Injection . . . 23
3.3.3. Linkability of Temporary Exposure Keys . . . 24
3.3.4. Server-Side Attacks and Gossip Attack . . . 24

4. TraceCORONA Implementation **27**
4.1. General Information . . . 27
4.2. Architecture and Packages . . . 27
4.3. Database . . . 28
4.4. User Interface . . . 30
4.5. Contact Establishment . . . 31

4.6. Encounter Download and Matching . . . 32
4.7. Infection Verification and Encounter Upload . . . 33

5. Comparative Design and Implementation Analysis 35
5.1. Server Side . . . 35
5.2. Server-Device Communication . . . 36
5.2.1. Generalized Worst Case . . . 37
5.2.2. GAEN . . . 37
5.2.3. TraceCORONA . . . 39
5.2.4. Comparison of Generated Traffic . . . 39
5.2.5. Plausible Deniability and Playbooks . . . 42
5.3. Device Application . . . 42
5.3.1. Multi-Platform Frameworks . . . 43
5.3.2. Risk Score Calculation . . . 43
5.3.3. Limits of GAEN . . . 44
5.4. Communication Between Devices . . . 45

6. Security Analysis 47
6.1. Known Attacks . . . 47
6.1.1. Profiling With or Without Bluetooth LE Communication . . . 47
6.1.2. Fake Alert Injection . . . 50
6.1.3. Time Modification Attacks . . . 52
6.1.4. Status Reveal Attacks, Digital Evidence . . . 52
6.1.5. Discussion of Attack Risk . . . 53
6.2. Formal Analysis . . . 54
6.2.1. Verifpal . . . 54
6.2.2. Models and Results . . . 55
6.3. Metadata Attacks . . . 57
6.4. Encounter Injection and Sybil Attacks . . . 58
6.4.1. Idea . . . 58
6.4.2. Assumptions . . . 58
6.4.3. Impact Analysis . . . 60
6.4.4. Countermeasures . . . 61

7. Traffic Measurements 63
7.1. Instrumentation . . . 63
7.2. TraceCORONA . . . 64
7.3. Corona Contact Tracing Germany . . . 65
7.4. Corona-Warn-App . . . 69
7.5. Comparison of TraceCORONA and Corona-Warn-App . . . 70
7.6. Comparison of Benchmarks and Theory . . . 71

8. Conclusion 75

List of Figures 79

List of Tables 81

List of Listings 83

Acronyms 85

Bibliography 87

Appendix **99**

A. Verifpal Models 99

B. Full TraceCORONA UML package diagram 112

1. Introduction

Epidemics and pandemics have been affecting human life through the ages with the earliest known pandemic being recorded in the year 430 BC in Ancient Greece [Thu13]. With advancing medical knowledge and technology, death tolls have been reduced considerably. Still, controlling a pandemic remains a challenging political and scientific task in current times, motivating research and development from experts in all scientific disciplines for ways to reduce the impact of infectious diseases.

The years 2020 and 2021 are overshadowed by the COVID-19 pandemic. As the SARS-CoV-2 virus is transmitted between humans through the air, isolating people who were in contact with a known-infected person helps with reducing infections. The effect of quarantine on the spread of the disease is further boosted as virus transmission happens in an exponential manner if no countermeasures are in effect. Consequently, preventive isolation as a countermeasure is facilitated by so-called contact tracing, which since the earlier SARS pandemic of 2002 and 2003 has been done manually by health officials using attendance lists and patients' own recollections.

Instead of or in addition to manual contact tracing, which is error-prone and labor-intensive, contact tracing can be performed automatically by devices such as smartphones. The basic idea of automatic contact tracing is to track the transmission of the virus as exact as possible using devices and technologies which can be easily and quickly used as an indication of a possible real virus transmission. In practice, this is mostly done using wireless radio technology, mainly Bluetooth Low Energy, which has a limited range and low power requirement, making it well-suited for continuous background scanning.

As with every technology, but especially so in the medical field, collection and usage of data exposes users to risks of abuse of this collected data. Amplified by the fact that receiving a warning as a result of tracing can lead to drastic measures such as quarantine, attackers collecting or modifying data are scenarios which need careful consideration. Bearing this in mind, if the tracing system loses its core functionality in pursuit of perfect privacy and security, it becomes useless.

At every step of the automatic contact tracing process there are technical decisions to be made which fundamentally impact the three core metrics of (1) efficiency (in terms of energy and internet bandwidth), (2) privacy and (3) effectiveness of the tracing protocol. These decisions, as well as the tradeoffs they lead to, are explored in detail throughout the thesis.

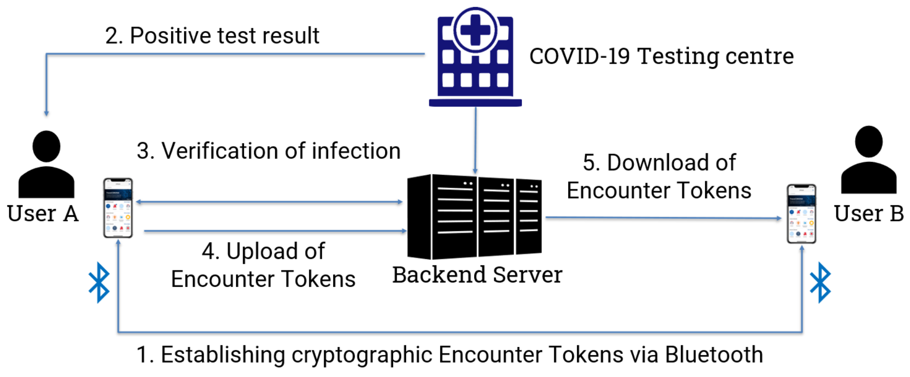

2. Positive test result
COVID-19 Testing centre
3. Verification of infection
5. Download of Encounter Tokens
User A
User B
4. Upload of Encounter Tokens
Backend Server
1. Establishing cryptographic Encounter Tokens via Bluetooth

Figure 1.1.: Overview of the TraceCORONA system, from [MNS20]

Analyzed Systems

In 2020, the first year of the pandemic, the race to become a local or national tracing app drove a high research interest in the field of digital contact tracing. After China announced the usage of digital contact tracing to combat the rapidly-spreading virus in February [hua20a], researchers and governments began development on various protocols and applications in the following months. Centralized systems such as TraceTogether [Gov20a] and PEPP-PT, which later became ROBERT [PF20], as well as decentralized systems such as Temporary Contact Numbers (TCN) [TCN20b] and DP-3T [TPH+20a] emerged.

During this time period, development of TraceCORONA was started at the System Security Lab of TU Darmstadt. The basic idea of the system, as visualized in Figure 1.1, is to use two-way communication to establish so-called encounter tokens every time two users of the system meet. When a user is diagnosed as positive, they upload hashes of their encounter tokens to the server, which then get distributed to all devices. Only the person on the other end of the encounter is then able to confirm their contact with an infected person.

While research and development for TraceCORONA was ongoing, the two leading mobile Operating System (OS) developers Apple and Google announced the integration of an automatic contact tracing system, Google/Apple Exposure Notification(s) (GAEN)[1] [AG20c], into their respective operating systems iOS and Android. Due to restrictions of the iOS platform (cf. Section 2.2.5), development of tracing applications able to run in the background was drastically limited, which was resolved by the OS gaining the capability to run tracing by itself. This new system uses a one-way communication approach to establish contact between users, as well as utilizing so-called diagnosis keys valid for one day to reduce the required internet bandwidth.

These design decisions centrally influence the security and privacy of users. The GAEN system sees worldwide usage to this day: 1.3 billion people live in areas where digital contact tracing apps based on GAEN are in operation[2]. Therefore, it is used in this thesis as a protocol to compare the security and privacy properties, as well as resource efficiency, of TraceCORONA against.

[1]Originally called Contact Tracing, Google calls the system Exposure Notifications, while Apple uses the singular form Exposure Notification.

[2]Based on the list of GAEN tracing apps in [Wik21] combined with the latest population estimates from [Uni21; US 21; Sta19].

Contributions

The main contribution of this work is the co-implementation of the client side of the previously-designed contact tracing protocol TraceCORONA using an Android application. We make significant contributions to the development of this application and document the finished prototype in detail. In addition to the implementation work, the research of this master thesis focuses on comparisons between the two systems TraceCORONA and GAEN. As a concrete tracing app to be investigated, we use the German Corona-Warn-App (CWA). In particular, we make the following contributions:

- Co-implementation of the TraceCORONA Android app in joint work with the development team from TU Darmstadt, as well as comprehensive documentation of the implementation
- Comparative analysis of design and implementation aspects of TraceCORONA and GAEN/CWA, revealing multiple bugs in Corona-Warn which allow attackers to gather information about infection state by network analysis
- Theoretical and empirical performance evaluation of TraceCORONA and comparison to GAEN/CWA, proving that TraceCORONA requires more bandwidth, but with some optimizations can still be reasonably used on the scale of CWA.
- Informal and formal security analysis of TraceCORONA and GAEN/CWA, concluding that TraceCORONA is more secure against known attacks and more privacy-preserving
- Discovery, analysis and discussion of a novel attack on contact tracing systems: the Sybil attack

We come to the result that TraceCORONA trades off resource efficiency for increased security and privacy of users.

Outline

The first part introduces the reader to the concepts necessary for understanding the work. Background information on the technical and ethical foundations of digital contact tracing, as well as the TraceCORONA protocol serving as the basis of our implementation, are introduced in Chapter 2. Afterwards, Chapter 3 introduces related work in the area of digital contact tracing protocols, attacks and security analysis.

After the introductory chapters, in Chapter 4 the implementation of the TraceCORONA protocol, as well as the technical and engineering choices, are described.

A large part of the thesis focuses on evaluations and comparisons. In Chapter 5 GAEN and TraceCORONA are explained in detail and compared on different theoretical aspects. Chapter 6 contains a formal and informal analysis of the two protocols, aiming to further verify the security and privacy claims made. In the practical evaluation of Chapter 7, traffic measurements are performed on the GAEN and TraceCORONA protocols using Android smartphone applications.

At the end of the thesis, Chapter 8 summarizes the results of the thesis and elaborates on potential for further work.

2. Background

In this chapter, the technological foundations common to all Bluetooth LE-based contact tracing protocols are described. In Section 2.1, the relevant parts of the Bluetooth wireless standard are described. Below, in Section 2.2, further technical considerations and tradeoffs common to all tracing protocols are explored. After touching upon the ethical considerations and privacy risks of digital contact tracing in Section 2.3, the focus of the rest of the chapter is the TraceCORONA protocol, whose co-implementation is a major contribution of this work (see Section 2.4).

2.1. Bluetooth Low Energy

Bluetooth Low Energy (BLE or Bluetooth LE) is a Wireless Personal Area Network (WPAN) technology introduced with the Bluetooth 4.0 standard in June 2010 [Blu10], and used in smartphones since October 2011 [OBr11]. It is a variant of the original Bluetooth protocol (named BR/EDR after its modes Basic Rate and Extended Data Rate), using similar physical and data link layers, but simplifying the upper layers allowing for lower power consumption at the expense of transmission throughput and security. These changed upper layers of the BLE stack are the Attribute Protocol (ATT) and Generic Attribute Profile (GATT). As these core protocols constitute the underlying technology of all BLE-based technologies, we describe their functionality in detail.

In contrast to BR/EDR, BLE does not require a manual connection process ("pairing"), although it is supported. Instead, device discovery is handled by the mechanisms of scanning and advertising in combination with the ATT and GATT protocols [Blu10, p.201]. This mechanism of automatic connections, as well as the large deployed user base in mobile phones makes the technology a good fit for beacon-based localization and contact tracing alike. In general, all Bluetooth packets, messages, commands, broadcasts and data – called Protocol Data Units (PDUs) – conform to the specified format.

2.1.1. Addressing and Link Layer

Every Bluetooth device is identified by a Bluetooth *device address* in EUI-48 format, similar to MAC addresses for Ethernet and WLAN devices. It can be either assigned by the manufacturer (*public address*) or randomly generated (*static address*), but by default it has a fixed value per device or power cycle. This fact enables indefinite tracking of devices by receiving advertisement messages on other devices, then linking the locations

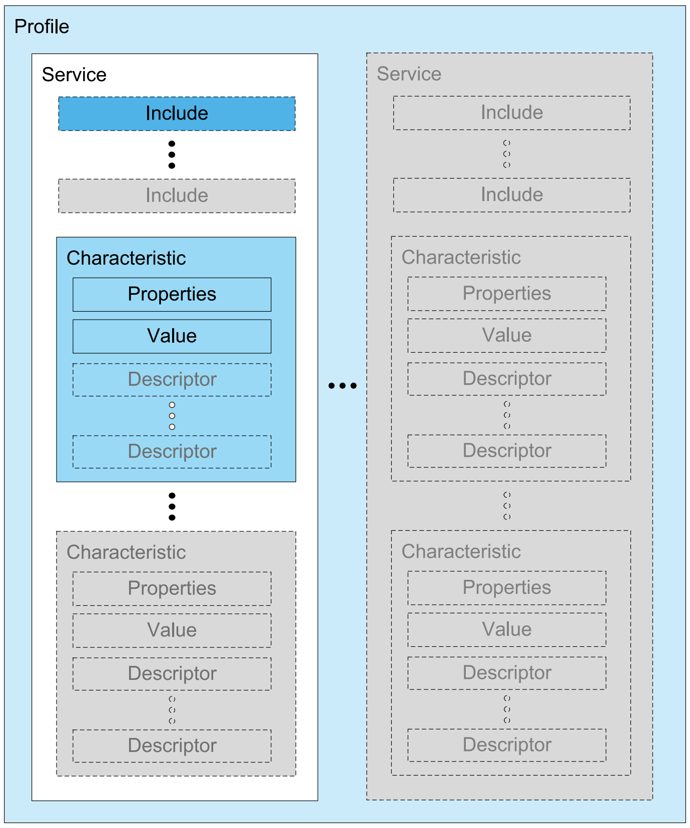


Figure 2.1.: Structure of the GATT hierarchy, from [Blu19, p.285]

and/or timestamps with the address. To combat this issue, modern Bluetooth devices support periodic randomization of device addresses (resulting in *private addresses*), which also helps protect the user's privacy while using contact tracing.

The Bluetooth Link Layer handles the device discovery using *scanning* and *advertising*. A device in the advertising state sends advertising events, which are chains of PDUs, with an interval from under 0.02 to 10.2 seconds[Blu10, p. 2223]. These can include extended information such as manufacturer information, which can be freely set by the devices to transmit additional data. Device scanning can be either *passive* or *active*: passive devices just receive and analyze the advertisements, while active devices query advertising devices for more information before potentially starting a connection. Once a connection is established, the higher-level protocols can be used.

2.1.2. Attribute-Based Communication

According to the ATT specification, every peer device takes a role of either server or client. The client sends command PDUs to the server, which allows the client to discover, read and write to *attributes* on the server side, while the server responds to the commands from the client and manages access. Every attribute type is identified by a standard 128-bit Universally Unique Identifier (UUID), allowing implementors to use random identifiers for every use-case with low chance of collisions. In addition, every attribute is identified by a *handle*, an identifier unique per server, and also used for ordering and grouping of attributes [Blu10, p. 1834].

To make sure devices implement attributes in a defined and standardized manner, use of the GATT is mandatory in BLE. It defines a hierarchy of *profile*, *service*, *characteristic*, and optional *descriptors*, in which the profile is the sum of all attributes a device implements. Services, characteristics, and descriptors are types of attributes with a fixed format.

A service groups multiple characteristics and can include other services. This is the unit that is actually advertised and scanned for device discovery purposes, however a device is free to implement multiple services for multiple functionalities. Every characteristic has a value, which can be read and written depending on its properties. It can also contain a number of descriptors further classifying its contents and context, e.g., the *Presentation Format* (name, data type, unit, order of magnitude, and namespace). Figure 2.1 is a visual representation of this hierarchy.

2.2. Technical Aspects of Bluetooth Contact Tracing

Now, once the underlying wireless protocol has been introduced, the focus shifts to concepts on a higher level present in all contact tracing applications. As these concepts are necessary for understanding and comparing protocols, in this section we explain concepts and tradeoffs to be made.

2.2.1. Scanning and Advertising Intervals, Rotating Identifiers

As scanning and advertising is energy-intensive, BLE tracing apps may not scan and advertise at all times. Instead, scanning and advertising can be started and stopped at certain intervals. Interval period, duty cycle, and possible randomization are factors important for effectiveness of tracing and power consumption.

Bluetooth device addresses can – and for privacy, should – be randomized, as stated in Section 2.1.1. This in turn makes the device address unsuitable as an identifier for tracing. Instead, at minimum one additional identifier or key has to be generated as a part of the tracing protocol and used in communications. How and how frequently these identifiers are generated, and for which purpose they are used, are parameters having an effect on power consumption and privacy of a protocol.

Some protocols save bandwidth by deriving multiple of these rotating identifiers from a master key, which is rotated in longer intervals, e.g., daily. This introduces an issue of linkability: All the keys derived from the master key can be provably linked to one person in the case of an infection, as the master keys are made public. See Section 3.3.1 for related work on this attack vector.

2.2.2. Contact Establishment, Matching and Exposure Notification

Once a device running a tracing application has been discovered by another device, there are different approaches to establishing and tracing the contact, or encounter, with the device. As with Bluetooth LE scanning itself (see Section 2.1.1), the establishment of contacts can be done in an active or passive fashion. A device utilizing an active scheme connects with the encountered device, exchanging additional information, while a passive scheme only logs the identifiers advertised by the encountered device and uses these for matching. Connecting to every device can lead to additional power consumption. Exchanging more data, however, also opens up more possibilities for use of algorithms to improve tracing accuracy by increasing resilience to attacks and protecting users' privacy by preventing passive profiling.

After contacts have been logged, and once an infection is discovered, users are notified of possible encounters with the infected person. The mechanism through which matching of infected users with user contacts is done can be grouped into two general approaches: *centralized* and *decentralized*. Both approaches are depicted in Figure 2.2.

Centralized approaches utilize a central system, such as a server, to detect contacts between users and match these to the user. In the example in Figure 2.2(a), all devices send the

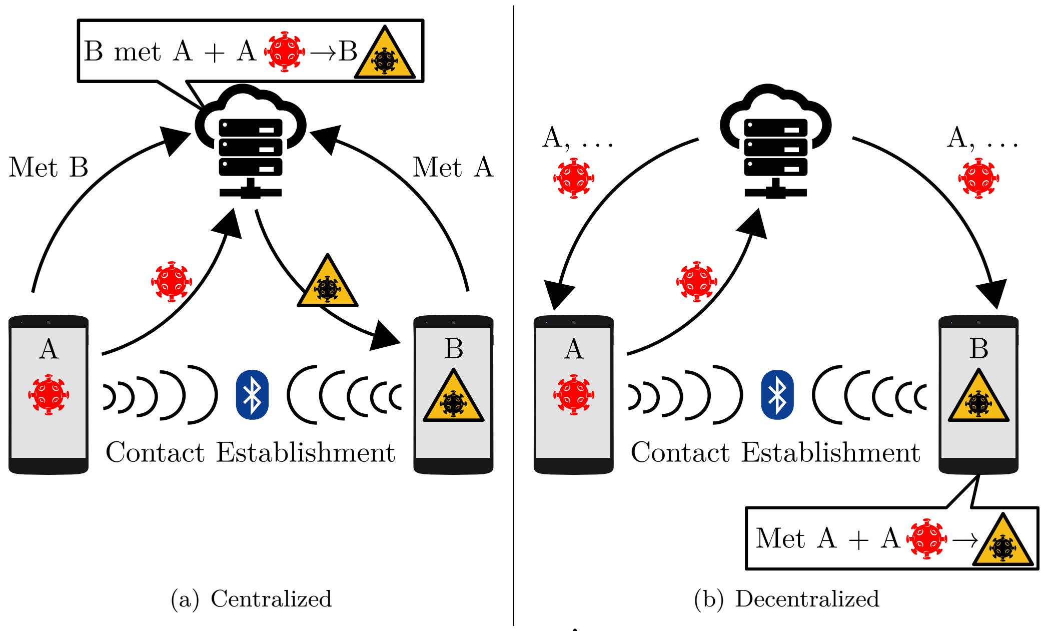


Figure 2.2.: Comparison of matching approaches

recorded Bluetooth contacts to a server. This allows the operator of the server to have a complete overview of contacts between users. On one hand, this allows for a more complete understanding of infection chains and enables faster responses, as well as allowing more flexibility with protocols to mitigate attacks on privacy by third parties. On the other hand, it opens the system up for abuse, as the tracing data can be used to extract detailed movement profiles of users. In case of an exploitation of the server side, this data can all be leaked to a malicious actor. Even without an exploit, it is available to a central actor and is more readily available for surveillance purposes, which is not usually a specified use-case of tracing apps.

Decentralized approaches instead perform the matching of contacts on the device itself. This is accomplished by synchronizing a list of tokens belonging to devices of known-infected patients, which the recorded contacts are then matched to. Depending on the algorithm, the list of tokens may be hashed to add another layer of obfuscation and prevent non-contacting users from extracting additional information. It is important to note here that decentralized matching does not preclude the usage of a central service for purposes other than matching, such as synchronizing tokens – in fact, to our best knowledge, all currently deployed systems rely on a central server for this purpose.

2.2.3. Proximity Approximation and Transmission Risk Calculation

Pure logging of contacts can lead to a high number of detected encounters which may suggest the user being close to lots of other users, while in reality the user may be separated from others by a wall. Thus, and as Bluetooth signal range can vary based on a number of factors, the sending and receiving signal strength and/or delay is often used to estimate the distance between users.

The *transmission risk*, which quantifies the probability of the virus being transmitted in an encounter, is aggregated for all encounters with confirmed infected users to calculate a risk score. The estimated distance can either be a component of this risk calculation, or used to filter encounters outright. Without considering proximity, false alarms will likely decrease the effectiveness of the system.

In certain scenarios, where the wireless signal is affected by nearby metal surfaces, distance measurement parameters may need to be calibrated differently or Bluetooth distance measurement might become completely unreliable, as shown in [LF20].

Not only the proximity is an important factor for transmission risk. On different days of an infection cycle, patients are more or less likely to infect others on contact. Also, the ventilation of the area is a factor for virus transmission. Some or all of these factors can be integrated into a transmission risk calculation, leading to more precise estimates and outcomes, in turn improving the effectiveness of the protocol.

2.2.4. Shifting and Padding of Transmitted Keys

To reduce server load, and/or due to manufacturer restrictions – e.g., for the newest version of Google/Apple Exposure Notification(s) (GAEN), only six calls for contact matching are allowed per day [Goo21e; App20] – several decentralized protocols transmit the keys to be matched as packages of all the keys generated in a certain time span. If there are always multiple users submitting keys during this time span, e.g., every hour, keys cannot be linked without doubt. If not, all the uploaded keys of a user are publically linkable.

To combat this issue of potential linkability, a technique called *shifting* is employed. If the number of keys inside a package is too small, the package is instead shifted to the next time period and merged with the newly-submitted keys, providing the necessary buffer. This, in turn, prolongs the time between submission of a key and notification of potentially exposed contacts, reducing effectiveness.

As the case of shifting, which limits the effectiveness, appears often at the beginning of the rollout or if the infection rate is low, another mechanism is introduced for additional protection. Every submitted key is *padded* with a fixed or random amount of other fake keys. To ensure the padding keys cannot be discerned from real keys by techniques such as statistical analysis, the metadata of the padding keys must be carefully chosen. For example, the Corona-Warn-App (CWA), which utilized these features in the past, set the metadata as being the same as the real keys.

Shifting and padding parameters must be carefully chosen to avoid linkability, which is a big part of the reason these techniques are applied in the first place. Section 3.3.3 shows an example of a scenario in which user keys are linkable in a small anonymity set.

2.2.5. Background Services on iOS and Android

To preserve energy and computing resources, the two major mobile platforms iOS and Android both have strictly-enforced rules about apps running in the background. On both platforms, once the visible part of an app (the *foreground*) is minimized, it is subject to being killed by the operating system at any time to free up resources (processor time and memory).

On iOS, if an app requires background resources, there are multiple specific Application Programming Interfaces (APIs) to run certain tasks in the background. In contrast to Android, all of these background tasks are hard-limited by time or a running upload/download operation. If an app exceeds its time limit, or is finished with its network operation, it is immediately terminated [App21a]. Regular background tasks are assigned a dynamic amount of time by the system, while specific types with defined limits are available for processing (high power, not time critical) or refresh (for regular updates) tasks. Another mechanism for background activity on iOS are notifications, which can either be triggered by the Apple Push Notification service or by a system service local to the device.

Critically, none of these strategies allow for an application to decide by itself when it wants to run, it always has to be triggered by an external source. Moreover, discovery of other Bluetooth devices by third-party applications is not possible if the screen is locked or the application is in the background[TCN20b]. Thus, implementing tracing apps on iOS without using the GAEN framework, which is integrated into the operating system, is considerably more difficult than on Android.

On Android, background activity is also restricted, however there are ways to mitigate this issue. Historically, the `IntentService` class allows for an application to run in the background [Goo20c]. With Android 8.0, new limits have been introduced, so background services are only allowed to run using Firebase Cloud Messaging, broadcasts, or for VPN services. This more recent restricted behavior is similar to Apple's background task system. Another mechanism introduced at this time, scheduled jobs, can run at a time decided by the application – which is fundamentally different from iOS, as the app can decide itself when jobs are scheduled, instead of the operating system.

In addition, on Android there are other ways to keep applications running while another application is open: *permanent notifications* and so-called *foreground services*. By providing feedback of running apps to the user, privacy and power consumption concerns are made clear, so the apps are allowed to run without occupying the whole screen.

Due to the restrictions imposed by operating systems, which can only be alleviated on Android, as well as the higher world-wide market-share of Android devices [Sta21] the development and benchmarking efforts in this thesis concentrate solely on the Android operating system.

2.3. Tradeoffs Between Efficiency and Privacy

Several civil rights groups, associations and researchers have analyzed and rated the tradeoffs made between efficient and privacy-preserving tracing. Major examples include the Germany-based Chaos Computer Club (CCC) and USA-based Electronic Frontier Foundation (EFF) and American Civil Liberties Union (ACLU), which are among the most prominent Non-Governmental Organizations (NGOs) fighting for privacy and against digital surveillance in their respective countries. All three organizations have published articles or papers [Neu20; COC20; Gil20] about criteria, safeguards and principles in digital contact tracing during the month of April 2020. In this section, the open letters which these groups have sent to app developers and governments are summarized and compared to provide context on societal requirements and privacy issues with certain tracing approaches.

First, the apps must be effective against COVID-19, but only against the disease, and not be misused for other purposes. This includes, but is not limited to, law enforcement, as is possible with attendance lists [Brö20]. Furthermore, [Neu20] requires that if the app lacks efficiency, it is shut down. That this efficiency must be measured and made public is postulated by [Gil20], which also provides suggestions for metrics such as number of exposures detected.

A further point the articles concur on is that all apps must be voluntary and free, in that there are neither fees charged nor incentives given for using the app. [Neu20] makes a statement about additional data besides information relevant for tracing to be collected for epidemiological purposes, which must require an additional confirmation, while others state that these systems "shouldn't be coupled"[Gil20].

For systems to be trustworthy, they must be independently auditable and verifiable. This is the reason why all three papers require publication of app source code. In addition, [Neu20] and [Gil20] require usage of reproducible builds for distribution of the app. These two articles also demand verifiable privacy not based on "organisational measures, 'trust' and promises"[Neu20], but instead on a sound technical design and algorithms.

On the server side, again all papers agree that a system must not have omniscient central servers, which rules out most centralized contact tracing approaches. Even so, the language used and the amount of protection required varies greatly – while [COC20] and [Gil20] require the "strongest possible technical and legal safeguards"[Gil20], [Neu20] goes as far as saying that neither server operators nor parties with access to communications metadata must be able to link the infection status of a user with personally identifiable informations, such as IP addresses. Fulfilling this requirement would make the usage of peer-to-peer techniques such as torrents, onion routing or blockchains necessary. Given the limited resources of the mobile devices targeted by tracing applications, as well as the amount of extra bandwidth generated, this wording of the requirement is not currently fulfilled by any tracing application known to the author, although a system has been theorized which is usable with a blockchain instead of a central server [ABIV21]. Additionally, even if the server side implementation of a certain protocol has its source code freely available, what is actually deployed may not match the published code. This makes all protocols based on a central server, whether it is for contact matching (centralized protocol) or just for data exchange (decentralized protocol), which is currently the only deployed option, inherently based on trust that this central server does not misuse data.

As for the data collected and stored on the device, [Neu20] allows only the duration of the encounter to be stored, while [COC20] allows the signal strength, as well as possibly the device types for better interpretation, and the date, not the time, "if public health officials think this is important to contact tracing". Taking again the example of GAEN, the date is required for verification of encounter validity, while the duration and signal strength are combined for risk estimation (see Section 3.1.4 for details), thus not complying with either policy. [Gil20] makes no such statements, but all papers require the data to be deleted after it is no longer needed for tracing. In addition, [Neu20] and [Gil20] call for encryption to be used for the local data on the device to prevent extraction.

[Neu20] and [COC20] mandate the usage of rotating identifiers (see Section 2.2.1) and explicitly state that the server may not profile the users. Notably, these requirements are absent from [Gil20], which alongside the others states that if rotating identifiers are used, they must not be linkable either to each other or to other personally identifiable information. [Neu20] adds finer-grained statements about the data collected and sent about devices not being enough to deanonymize anything.

Finally, some points not addressed by [Neu20], but by the other papers, include non-discrimination of certain population groups, including but not limited to infected and high-risk people in the context of COVID-19. In addition, the app must have a mechanism to stop functioning, and must not hinder any other efforts against the pandemic, such as testing and research on treatments.

To summarize, the technical requirements stated by [Neu20] could be called a technical "gold standard", which can prove difficult to achieve especially in the limited time span where tracing is effective. [COC20] and [Gil20] focus more on the societal impact of applications, deviating from the CCC article's hard and fast rules to not make unreasonable claims, but rather demanding a best effort from developers and government agencies. That said, all of these articles provide criteria on which all protocols with their tradeoffs can be measured.

2.4. The TraceCORONA Protocol

The TraceCORONA project was started in early April 2020 with its tracing protocol being the first step of development. This protocol uses active contact establishment and a decentralized matching (see Section 2.2.2). As the co-implementation of this protocol in a

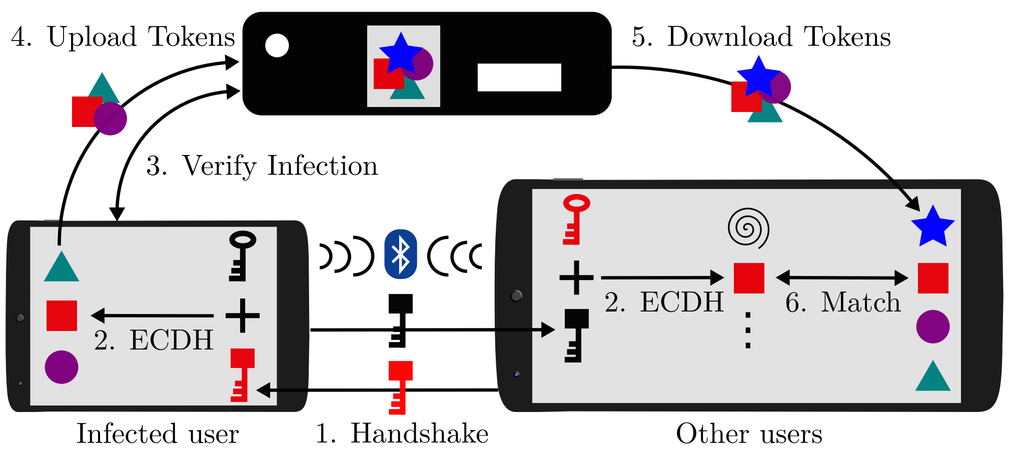


Figure 2.3.: Steps of the TraceCORONA protocol, inspired by [NMS20, Fig. 4] . Note that hashing of encounter tokens is omitted for visual clarity.

prototype application is a major part of the thesis, it is described in detail in this section. This description is based in part on [MNS20; NMS20].

Figure 2.3 gives a detailed overview of the steps and parties involved in contact tracing with TraceCORONA. The two large mobile phones signify two users of the system who meet each other. Later, the left user turns out to be infected and declares this status in the TraceCORONA app. The following sections each explain one of the steps enumerated in the figure.

2.4.1. Bluetooth LE Handshake

Every device running TraceCORONA has an active Elliptic Curve Diffie-Hellman (ECDH) key pair and a randomly-generated *rollingID* value. While tracing is active, the devices continuously advertise their *rollingID* through BLE advertisements with a fixed service UUID. On a regular basis, devices also scan for BLE advertisements with this UUID.

Figure 2.4 shows the process started when an advertisement is received by a scanning device. At first, the receiving device checks if it has seen the advertised rollingID already. If this is the case, the device just saves the metadata of this advertisement (timestamp and signal strength) to the database as a *Scan* (see “Rediscovery of same device” in Figure 2.4).

Otherwise, the initial handshake is performed with the encountered device. A GATT connection is established and a characteristic read request is sent with a fixed characteristic UUID. The other device then sends back their public key (key with a rectangular head in Figure 2.3). Afterwards, a characteristic write request is sent with the same UUID with the initiator’s own public key, the signal strength and transmit power level included in the received advertisement (Received Signal Strength Indication (RSSI), rssiTx) as well as the rollingID for cross checking.

At the end of the handshake, both devices save an *Encounter*, which can have multiple *Scan*s, as well as the initial *Scan* for this encounter. Devices change their ECDH key pair and *rollingID* after a fixed period of time. If the devices again encounter each other, a fresh encounter record is generated.

2.4.2. ECDH Token Establishment

Before up- or download of encounter tokens, the gathered encounter data (*EncounterTokenParameters*) are converted into *encounter tokens* by means of the ECDH algorithm. Through this process, a secret is established between the two parties of the encounter.

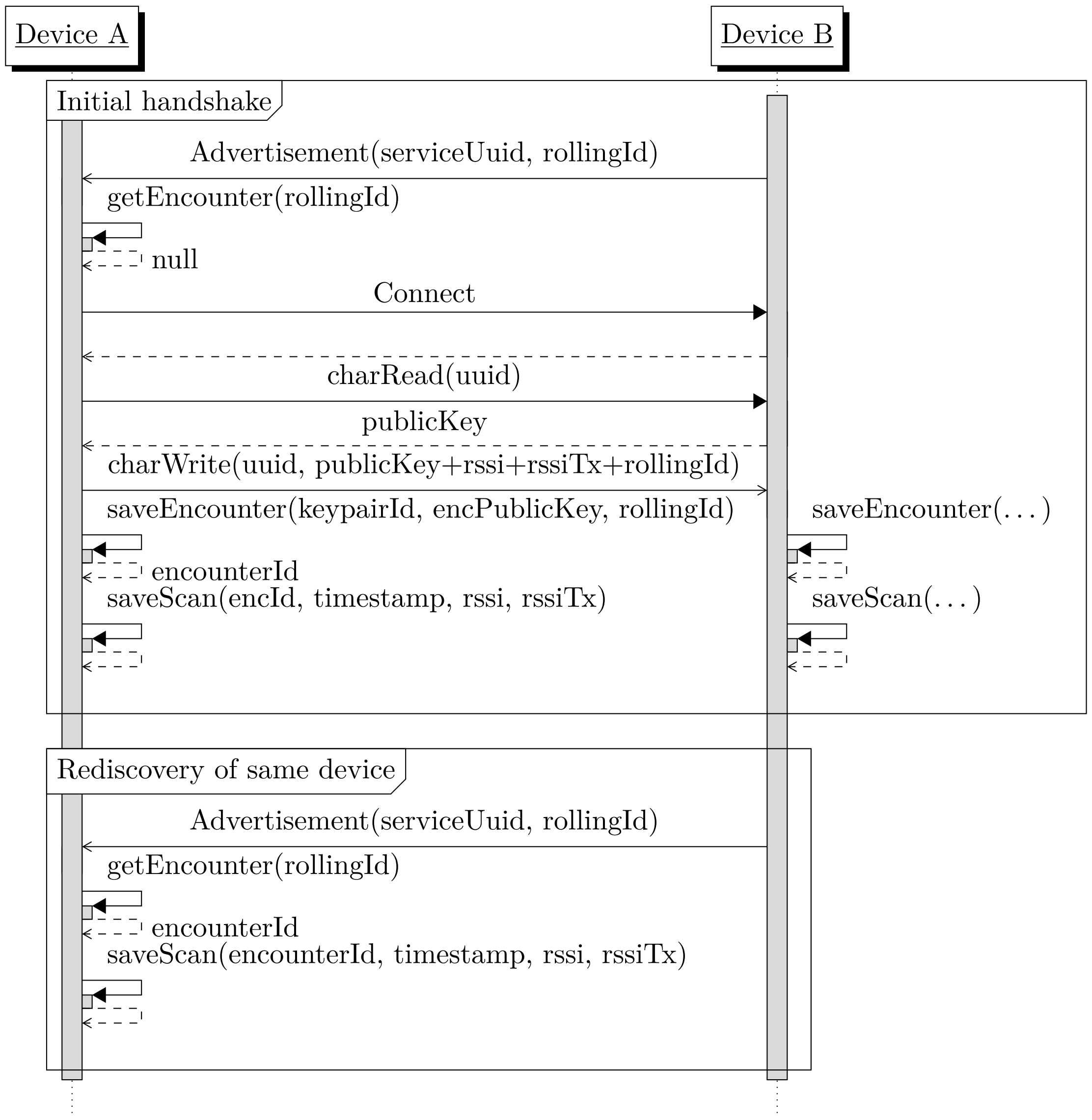

Device A
Device B
Initial handshake
Advertisement(serviceUuid, rollingId)
getEncounter(rollingId)
null
Connect
charRead(uuid)
publicKey
charWrite(uuid, publicKey+rssi+rssiTx+rollingId)
saveEncounter(keypairId, encPublicKey, rollingId)
saveEncounter(. . .)
encounterId
saveScan(encId, timestamp, rssi, rssiTx)
saveScan(. . .)
Rediscovery of same device
Advertisement(serviceUuid, rollingId)
getEncounter(rollingId)
encounterId
saveScan(encounterId, timestamp, rssi, rssiTx)

Figure 2.4.: Sequence diagram of Bluetooth LE contact establishment in TraceCORONA

This secret token is later used for decrypting end-to-end-encrypted metadata and must never leave the device. Therefore, tokens are also hashed using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [KE10] to yield *tokenHash*es, which are used for comparison purposes.

2.4.3. Infection Verification

Once a user tests positive, they receive a Transaction Authentication Number (TAN) from their health authority to be able to notify the people they encountered in during the tracing period, i.e., the last 2 weeks. Figure 2.5 shows the communication protocol used when sharing encounter tokens with the server under "Encounter upload after positive test". In the first step, the TAN entered by the user is submitted and the device receives a *nonce* value from the server.

2.4.4. Encounter Token Upload

In the remaining part of the upload process, a random key is generated for each encounter token. This key is used to encrypt the infection state (0 for users which have received a positive test result) and the nonce received from the server. The random key itself is then encrypted using the encounter token as a key. The upload message sent to the server consists of a list of all collected encounter *tokenHashes* with their corresponding *keyEncryptions* and *stateAndNonceEncryption* parts. In addition, the client sends the random key for verification on the server side. It will not be distributed to clients later.

Once the server receives the upload message, the nonce and infection status values are decrypted and checked against the database of valid nonces. If the check succeeds, the server sends back a confirmation to the client and publishes the upload message, separated into single tokens and without the unencrypted random keys, to other users.

2.4.5. Encounter Token Download

On a regular basis, or when triggered by the user, the app downloads the published messages from the server. This flow is depicted in Figure 2.5 as "Encounter download for matching". The client submits the time when they last downloaded token messages and the server only sends the newer messages. In response, the messages are streamed to the client device.

2.4.6. Matching

As a final step after download of encounter token messages, each encounter token hash is compared to the device's own database. If a matching entry is found, this confirms the encounter between the device with an infection state and the own device. To gain information about the infection state of the encounter, the device then retrieves the corresponding unhashed encounter token from the database, uses it to decrypt the random key, and in turn decrypt the infection state and the nonce. Based on the gathered metadata about the encounter, a risk scoring algorithm (cf. Section 2.2.3) is used to quantify the risk for the exposed user.

Observant readers might wonder what the nonce was transmitted to the client for. If the tracing remains limited to a single step, which is the current state of implementation, it is indeed unused and can be omitted by the server. In this case, this is the end of the protocol.

In addition, TraceCORONA has the ability to recursively trace indirect infections – if the second user from Figure 2.3 decides to publish their contacts, the message is generated

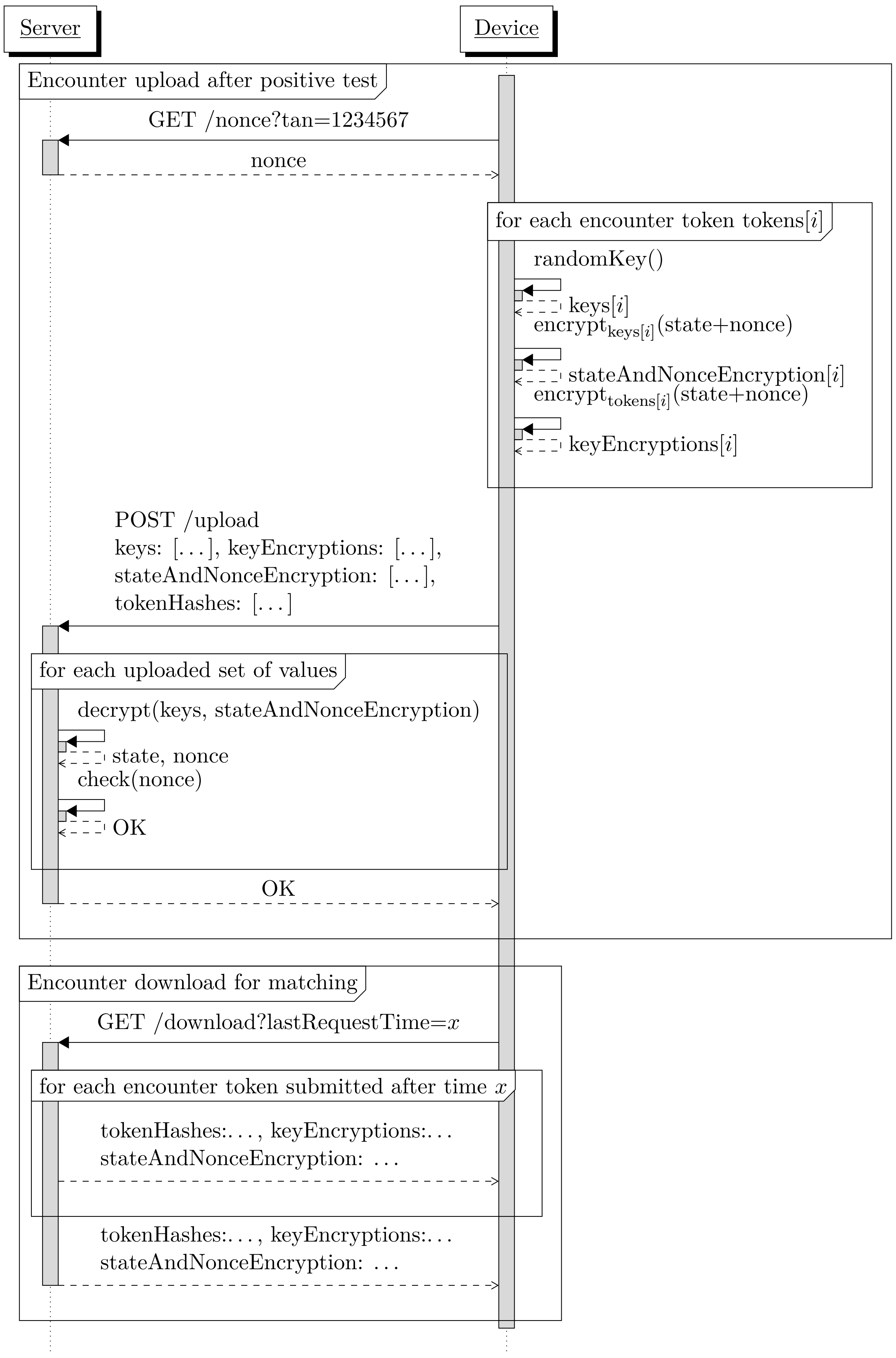

Server
Device
Encounter upload after positive test
GET /nonce?tan=1234567
nonce
for each encounter token tokens[i]
randomKey()
keys[i]
encrypt$_{\text{keys}[i]}$(state+nonce)
stateAndNonceEncryption[i]
encrypt$_{\text{tokens}[i]}$(state+nonce)
keyEncryptions[i]
POST /upload
keys: [. . .], keyEncryptions: [. . .],
stateAndNonceEncryption: [. . .],
tokenHashes: [. . .]
for each uploaded set of values
decrypt(keys, stateAndNonceEncryption)
state, nonce
check(nonce)
OK
OK
Encounter download for matching
GET /download?lastRequestTime=x
for each encounter token submitted after time x
tokenHashes:. . . , keyEncryptions:. . .
stateAndNonceEncryption: . . .
tokenHashes:. . . , keyEncryptions:. . .
stateAndNonceEncryption: . . .

Figure 2.5.: Sequence diagram of server-device communication in TraceCORONA

in a similar fashion. The already-matched keys are omitted, as this would lead to an unnecessary back-and-forth loop. The newly-generated message retains the nonce value as a proof the user is part of this potential infection chain (it can only be decrypted with a matching encounter token), but increments the infection state by 1. Once users download this message, the users with an indirect encounter (i.e., who were in contact with a person who has been in contact with a confirmed infected person) can also receive warning messages. This can be repeated as often as desired. We will not further explore this possibility in the remainder of the thesis, focusing on single step tracing as is the norm today.

3. Related Work

After introducing the technical and ethical basics of Bluetooth-based contact tracing, in this chapter the existing approaches and work in the field are described and analyzed.

3.1. Existing Contact Tracing Approaches

A large amount of research and development during the COVID-19 pandemic has lead to various universities, research institutes, and companies developing several approaches, protocols, and applications for contact tracing. After giving a brief overview of the beginnings of the field, the remainder of the section introduces some of the different protocols existing at the moment categorized by type of contact matching (cf. Section 2.2.2), describing the concrete choices made for the key tradeoffs introduced in Section 2.2 and briefly touching on the history and usage of each protocol.

3.1.1. Beginnings of Mobile Contact Tracing

In the years 2009 to 2012, the FluPhone project conducted experiments on the usage of mobile apps for epidemiology purposes. These included contact tracing using the Bluetooth Basic Rate/Extended Data Rate (BR/EDR) technology. Designed to run on feature phones using Java Mobile, the app recorded encounters by scanning for Bluetooth devices and recording the device address. This data was then sent to a server in full, which makes FluPhone a centralized approach [Yon09]. Rolling Bluetooth addresses (see Section 2.2.1) were not implemented in older versions of Bluetooth before Bluetooth Low Energy (BLE or Bluetooth LE) was introduced, so this was an obvious choice for a unique identifier. A study was conducted to simulate outbreaks of different diseases, including the SARS disease related to COVID-19, using this technology [Yon11].

The first occurrence in the news of a mobile app-based contact tracing approach related to COVID-19 was in early February 2020, when Chinese state news agency Xinhua announced the development of a “novel coronavirus close contact detection app” [hua20a]. There is not much information known about the implementation of the application, only that it uses phone numbers and national ID numbers as identifiers, using a centralized approach linked with databases from other government agencies to retrieve data. This makes it a centralized contact tracing approach, however not using any wireless technology for tracing. Later press releases by Xinhua in the month of February include a digital attendance list system on the Shanghai public transport system [hua20b]. Because all approaches implemented in the country are proprietary to China and details are sparse, we are not going to be exploring them further.

First ideas about using Bluetooth technology in contact tracing were published in the end of February, when the CoEpi project (see Section 3.1.3) was created on GitHub [LLCR20] and beginning of March, when two interviews with scientists [Gor20; SF20] independently yielded the basic idea. Throughout the month, development began on the first modern Bluetooth contact tracing protocols and applications. Some of these protocols are described in the following subsections.

3.1.2. Centralized Approaches

TraceTogether (BlueTrace)

The first BLE contact tracing app rolled out by a government is the TraceTogether app used in Singapore. It was launched on March 20, 2020 and uses a protocol called BlueTrace [Gov20a]. A reference implementation for this protocol was made public shortly after under the GNU General Public License as OpenTrace [Ope20], however the official TraceTogether application remains closed-source. In addition, a white paper [BKT+20] containing a detailed description of the protocol and considerations has been published.

Before starting, users are required to register with their phone number, which is subsequently linked to a user identifier (UserID) on the server. Temporary identifiers (TempIDs), which contain symmetrically encrypted UserID and timestamp values, are generated centrally and transmitted to the users' phones. This ensures only the server is able to map TempIDs to UserIDs, and in contrast to locally generating encryption keys, is more energy efficient on the phone. Every Temporary ID is valid for 15 minutes to prevent profiling attacks based on observation (see Section 3.3.1), and to make replay attacks more difficult. Scanning is performed 15 to 20 percent of the time, while advertising takes place 90-100 percent of the time. Finally, provisions are made for interoperability between countries [BKT+20].

The protocol is deployed in Singapore in a way that is able to cooperate with manual contact tracers, called "human-in-the-loop" [BKT+20]. Due to the SARS outbreak in 2003, Singapore has an extensive contact tracing force and expertise, which is why this approach works for the country. In other countries, where SARS was not widespread, modern manual contact tracing has not been deployed before and the resources to double-check and follow up on every infection for effective tracing are not available. In general, the protocol requires users to place full trust in the government, as all the data are collected centrally and linked to the phone number, which is a personal identifier. If the protocol were to be deployed stand-alone, without involving manual contact tracing, the phone number would not be required and notifications could be sent in the app instead.

StopCovid/TousAntiCovid (PEPP-PT/ROBERT)

In Europe, the public research institutes Inria and Fraunhofer AISEC, among a consortium of institutes, companies and universities began development on the PEPP-PT project in late March 2020. As the first complete proposal for a centralized protocol was published by the core group, on April 19 a significant portion of supporters dropped out of the consortium to pursue decentralized tracing approaches instead [Var20]. This lead to the creation of the DP-3T project, described in Section 3.1.3. Afterwards, PEPP-PT was developed into ROBERT, the protocol used by the app called StopCovid. The app, later renamed to TousAntiCovid, was officially deployed in France on June 2 [Dil20]. A full description of the protocol was published shortly before the app release [PF20].

This protocol is a balance between the fully-centralized and deliberately linked BlueTrace protocol and the decentralized solutions, mixing parts of both to provide more privacy. In this solution, an initial registration is performed, assigning an identifier without collecting additional data, but employing a *Proof of Work* mechanism to make abuse harder. The device regularly downloads so-called *Ephemeral Bluetooth Identifiers* from the server, which are then exchanged over Bluetooth with nearby devices. If a user is tested positive, the app uploads the so-called HELLO messages, which were exchanged in the presumed period of contagion, to the server. Notification of clients is handled by a regular check-in from the device, which can be combined with the download of new ephemeral identifiers. There is also a provision for federation between country servers by transmitting a country code alongside the HELLO messages [PF20].

The authors propose a unique way of masking the so-called "one-entry" attack, where a malicious actor only activates tracing when meeting a specific person. By introducing a random chance that an intentional false positive warning is generated, the attacker cannot be sure their target is actually infected [PF20]. We explore this attack among other similar ones in Section 6.1.4.

3.1.3. Decentralized Approaches

CoEpi/Covid Watch (CEN/Temporary Contact Numbers (TCN))

The CoEpi project was started by US-based developers on February 23, 2020 as one of the first proposals to use Bluetooth LE for contact tracing [LLCR20]. Development on the mobile applications and server-side code started on March 22 and has stagnated after August. The protocol of CoEpi, initially called CEN (Contact Event Numbers), was renamed to TCN on April 8, which marked the beginning of the TCN Coalition [TCN20b]. By today, the TCN Coalition is a part of the Linux Foundation under the name Linux Foundation Public Health [Lin20]. In contrast to other state-sponsored work, the development of this project has always been public and released under the MIT License. The TCN protocol was also worked on and co-developed by the Covid Watch organization, starting on March 11 [TCN20a].

In this approach, for a given period of time the phone creates a set of keys called Report Authorization Key (RAK) and Report Verification Key (RVK), rotated up to every 6 hours. By utilizing a *cryptographic ratchet* algorithm[1], which includes the SHA-256 hash of the previous key and the RVK, a number of Temporary Contact Keys (TCKs) are generated. The TCKs are then concatenated with the sequence number and hashed again to generate the Temporary Contact Numbers (TCNs), which are broadcast over BLE. TCKs and TCNs should be rotated in sync with the Bluetooth device addresses to prevent linkability. This protocol implements the most versatile broadcasting algorithm of the analyzed work, allowing for Android and iOS in both foreground and background states to communicate. In case two iOS devices in a background state want to communicate, an Android device must act as a relay between the two devices, actively mediating communications between both.

The TCN protocol is an early decentralized tracing protocol, containing interesting approaches which can be used for further development of other protocols. The Linux Foundation Public Health as well as Covid Watch now use the Google/Apple Exposure Notification(s) (GAEN) protocol (introduced in Section 3.1.4) for the deployed versions of their apps and the development of TCN seems to have stalled.

[1]Like a mechanical ratchet, cryptographic ratchets are algorithms whose steps can only be advanced, not reverted back from a future step.

DP-3T

Researchers from the Swiss universities EPFL and ETHZ, among others, uploaded the first version of their white paper on the DP-3T standard to GitHub on April 3, 2020 [TPH+20a]. Here, the researchers propose a decentralized standard for contact tracing using Bluetooth LE. This standard was later expanded into three protocols with varying tradeoffs made between privacy, amount of bandwidth and processing power needed. The third protocol is identical to GAEN, if its variable window is chosen to be 24 hours, and so is not going to be elaborated further in this section. However, the authors criticize this choice: "We recommend a time window of 2 or 4 hours depending on the bandwidth availability in the region." [TPH+20b]

In the original design, later renamed to "low-cost", the phone generates a seed out of the previous day's seed and, using a pseudo-random generator, generates the amount of ephemeral rolling identifiers needed to cover all time intervals of the day. Every ephemeral identifier is 16 bytes long due to limitations of the BLE protocol with regards to passive advertisement and scanning, which is used to establish contacts. The use of a cryptographic ratchet is comparable with TCN, but instead of every time interval, the ratchet is advanced daily. If an infection is confirmed, the seed of the predicted starting day of the infection (e.g. 14 days before discovery) is uploaded and distributed to other users and a new random seed is used from then on, creating a fully new identity. By uploading a seed linking all exposures together, by design infected users' recordings are linkable across all days of an infection, which can be up to 14 days.

To fix this issue, the DP-3T team developed a second design, called "unlinkable". Ephemeral identifiers are generated from random seeds, which are saved to the phone, and encountered phones store a hash of the ephemeral identifier and the number of the time interval in which the identifier was scanned. In the case of an infection, the random seeds are uploaded. All random seeds are then encoded into a Cuckoo filter, which is distributed to devices. This design requires more bandwidth, storage and computational power, yielding a vastly improved privacy for infected users.

Overall, the DP-3T team presented designs focusing on very different goals: one on minimum resource usage with reduced privacy, and one with more resource usage than all other protocols and very good privacy protection. After the publishing of the DP-3T whitepaper, Google and Apple developed GAEN and were certainly inspired by the DP-3T designs [EL20].

Pronto-C2

A more recent development in the space of decentralized digital tracing protocols, Pronto-C2 [ABIV21], employs a similar key-based approach to TraceCORONA, in that Elliptic Curve Diffie-Hellman (ECDH) keys, instead of identifiers as with other protocols, are shared by a central server. The main difference between the two protocols is the way in which this key exchange is performed: While TraceCORONA uses a local Bluetooth connection, Pronto-C2 uses a separate public mapping of Bluetooth identifiers to ECDH inputs, where each device then retrieves the current mapping.

In addition, the communication between the server and devices uses extra steps for Pronto-C2: An additional authentication service is introduced to reduce the risk of DoS attacks, while mixing servers are used to improve anonymity towards the central server. For infection verification, blind signatures are distributed by a separate server after verification through the health authority and associated with the uploaded keys, while in TraceCORONA the infection status is encrypted with the encounter token and verified by using a nonce value obtained from the verification server.

The paper evaluates performance metrics similar in type to the ones we are going to calculate for TraceCORONA: With 5000 new infections per day, a 15-minute length of keys and RSA-2048 used as a signature algorithm, each user has to download 177 MB and upload under 350 KB of data per day. In addition, each device belonging to a non-patient performs under 3500 exponentiations for the Elgamal cryptography, and if a user is tested positive, the device needs to perform circa 13500 of these operations.

3.1.4. Google and Apple Exposure Notifications

Currently the most widely used protocol for BLE contact tracing is implemented by the Google/Apple Exposure Notification(s) (GAEN) framework. It combines features implemented by the DP-3T "low-cost" and "unlinkable" designs to save bandwidth while still keeping linkability to a day. Temporary Exposure Keys (TEKs) are generated every day, and a number of Rolling Proximity Identifiers (RPIs) are derived from it by encrypting a so-called RPI key with the number of the time interval. Metadata, such as the signal strength, is transmitted alongside the RPI [AG20b; AG20a]. In the case of an infection, the TEKs, which now become diagnosis keys, are synchronized via the application and submitted for checking to the operating system Application Programming Interface (API), which internally computes the RPIs and matches these to the encounter records.

Infection verification and synchronization of keys is left up to the client application. Due to familiarity and the availability of open source code and analysis tools, we are using the German Corona-Warn-App (CWA) as an example of a GAEN app. It implements the infection verification via a system based on Transaction Authentication Numbers (TANs). A TAN is needed to authenticate against the main backend when uploading diagnosis keys and can be obtained by one of two flows. In the case of an "integrated laboratory" the test laboratory is connected to the verification server using the *Laboratory Information System* and allows retrieving a TAN if a positive test result is obtained via a QR code containing a unique identifier. Otherwise, the TAN is generated by an employee of the health authority and given to the user via a phone call. This flow is called "teleTAN" [Sti20].

After an infected user has registered their positive test result, they are asked to optionally provide information about the date when symptoms of the disease began to appear, called Days Since Onset of Symptoms (DSOS) [Hoe20], to be included with every key upload. Previously, based on this information, a value called the Transmission Risk Level (TRL) was calculated for each submitted TEK before key upload, in even earlier versions of the application a fixed sequence of levels going back from the upload date was used. After keys are transmitted to the server side, the keys with metadata (time of validity and TRL) are packaged once every hour and distributed to other users. Once other clients have downloaded the keys, the TRL, as well as the measured signal strength, are used by GAEN to estimate the risk of an encounter (cf. Section 5.3.2) [Cor20a]. Depending on the probability of infection, warning messages are displayed to the other users [Wol20].

In Europe, national tracing apps based on GAEN can federate their diagnosis keys between countries to enable travelers to use their national contact tracing app throughout the continent. This is facilitated by the *European Federation Gateway Service* operated by the German app team. We only focus on the national tracing functionality in this thesis.

3.2. Analysis of Contact Tracing Approaches

Many efforts focus on documenting, analyzing, and improving already deployed tracing solutions further. Some of these efforts focus on providing an overview and survey of existing work, while others focus on a specific implementation and aim to track efficiency, gain insights, and ultimately make concrete proposals for improvements. In this section, both categories are going to be discussed: projects aiming to list and compare different approaches, and projects focusing specifically on the German Corona-Warn-App and the underlying GAEN framework, which has been the main focus of our research so far.

3.2.1. Lists, Comparisons, Surveys, and Indexes

Throughout the years, there have been several attempts at keeping up-to-date lists of apps related to contact tracing and other topics around the COVID-19 pandemic. An early article, giving detailed information about every app and focusing mostly on privacy aspects, is [SBG+20]. It stopped receiving regular updates in mid-May 2020. Still ongoing efforts are [ORJ20], which rates apps based on five criteria laid out by the American Civil Liberties Union (ACLU) (also see Section 2.3) and the Wikipedia article [Wik20].

In addition to listing different apps, yet another category of work covers various aspects of the protocols used in more detail. [Alb20] is an extensive survey of utilized frameworks and requested permissions in nearly 500 COVID-related iOS apps, not only focusing on tracing apps, but also pure informational applications. It concludes that half of all apps contain at least one framework made by Google. [AAA+20] analyzes the permissions, privacy policy, reviews, and whether the app is using Transport Layer Security (TLS)-secured connections for 26 tracing apps. They conclude that several of the applications are using unclear language in the privacy policy, require too many permissions, and/or, in five cases, even fail to use TLS. Based on reviews, users are aware of these concerns. Finally, [AMX+20] is a detailed survey of architectural features, attacks, protocols, and user concerns about contact tracing.

3.2.2. GAEN and Corona-Warn-App Analysis Tools

On May 17, one month before the final release of the Corona-Warn-App, a project by developer Huebler, was launched to create a framework for experimenting with and analyzing the data format of the app [Hue20b]. This led, among others, to the discovery of privacy issues related to key linkability (cf. Section 3.3.3). In addition, this tool was used in multiple dashboards to visualize different parameters related to app usage, which allows for analysis of the app's effectiveness [Pfi20; Böh20]. Later, on October 4, 2021, the CWA team published a "key figures dashboard" containing statistics on app downloads, registered tests, sharing behavior (how many users actually share their diagnosis keys) and issued warnings [Hou21].

3.3. Attacks on Contact Tracing

Apart from privacy concerns in regular usage and operation of contact tracing apps, malicious actors can mount several attacks, which may lead to deanonymization and false positives, thus decreasing privacy and effectiveness. The paper [BDF+20], co-authored by us, presents and demonstrates two basic attacks common to most Bluetooth tracing protocols, such as GAEN. Further research has yielded more weaknesses, allowing for injection of fake alerts [AFV21b; IVV21b] or linking of rolling identifiers among multiple days [Hue20a]. Google has published a FAQ document responding to several attack vectors [Goo20e].

3.3.1. Profiling Using Bluetooth Sniffing

The basic functionality of Bluetooth LE-based tracing makes it necessary to send rotating identifiers out. Combining received identifiers together with timestamps and locations of reception allows for the creation of movement profiles, as long as the identifier stays the same. This can be combined with the linking of rotating identifiers described in Section 2.2.1 for the creation of movement profiles of infected people.

For executing an attack in practice, there are multiple options on how to obtain the data. One can deploy a network of fixed-position Bluetooth sniffers, as demonstated in [BDF+20]. A case study in this paper estimates the number of tracing stations needed to collect coarse-grained information about all people working in a city of 160000 inhabitants during their commute to work at 395 to 465. Another option is to utilize compromised smartphones of other users by using malware or injecting code through a vulnerable or modified framework [DR20].

The paper [ABIV21] introduces more variations on this attack to highlight design decisions of the introduced protocol. Based on the network of Bluetooth sniffers, called Paparazzi attack, there is the Orwell attack, where server data are also in the hands of the attacker, as well as the Matrix attack, which allows sending of BLE data in addition to receiving. As these attacks require more data than the original sniffing attack, they can be seen as an extension or potential workaround against mitigations introduced for the original attack.

A different angle on Bluetooth sniffing is presented in [NAE+21]. Instead of gathering movement profiles of users the aim of this work is to gather photographs of infected individuals. To achieve this purpose, in practice the attacker uses a directional antenna and receives RPIs from passers-by using GAEN-based applications. Once the signal strength is high enough, a snapshot from an attached camera is taken and the picture saved together with the RPI. Later, these simulated encounters are processed one-by-one by the matching algorithm like real encounters would, which allows the device to know the infection status of every photographed individual. The same attack, but performed more manually with a paparazzi spying on celebrities is proposed in [Vau20].

3.3.2. Fake Alert Injection

Users receive alerts if an encounter meeting certain criteria is recorded. Normally, this encounter is recorded directly from a phone or other token running the same protocol which is actually in the physical vicinity. There are several ways of injecting fake encounters into most contact tracing systems.

A wormhole attack, demonstated in [BDF+20], captures the BLE advertisements sent out by a legitimate app, and forwards them to one or more remote attacker devices, which re-broadcast the advertisements, acting as a beacon of their own. If an already infected user or to-be-infected user is on the side of the wormhole where messages are being captured, people who were not in contact with the infected user are going to register a risky encounter. The attack was tested using both the DP-3T sample application and CWA and works in both cases.

To receive a token belonging to an infected or soon-infected person is a challenge which has been studied by the authors of [AFV21b]. The paper presents several approaches on how a person with malicious intent could buy TEKs from infected app users without trusting them: using JSON Web Tokens, or using decentralized oracles. Both attacks were demonstrated with the Italian Immuni app and the SwissCovid app. The authors note that a part of the attack, which allows proof of ability to upload keys, is likely also possible on CWA due to a similarity in the protocol.

Actually injecting fake recent exposures into phones is not trivial, as the TEK of the current day is kept inaccessible inside the GAEN framework. In order to still generate a fake warning message, [IVV21b] proposes several methods to manipulate the victim device's time. Either setting the time manually, which requires physical access, or using rogue NTP servers, mobile base stations, or GNSS senders. Then, replay attacks can be carried out using the TEK which was valid on the day the phone is now set to.

In [ABIV21], an additional attack resulting in fake alerts is theorized (Matteotti attack). Here, the attacker colludes with the server and health authority and is able to place BLE receivers. The hypothetical motivation and outcome is a government trying to stifle political opposition by sending members of parliament into quarantine [ABIV20].

3.3.3. Linkability of Temporary Exposure Keys

When the CWA was rolled out, the number of users as well as the infection rates were low. This, combined with the always identical profile of TRLs, a fixed number assigned to the key dependent on the amount of time passed since the positive test, enabled using data analysis to link together TEKs of up to 12 days without doubt in special cases [Hue20a].

Figure 3.1 shows an example of a diagnosis key file from the time period when padding was active and TRL values were not yet varied based on symptom indication. One can clearly see that three users uploaded their diagnosis keys. If an attacker had recorded one user's movement as in [BDF+20], they would have a linkable movement profile over 11-13 days. In this case, it is extremely unlikely that they would have recorded multiple users from this package with an anonymity set of three users distributed over Germany.

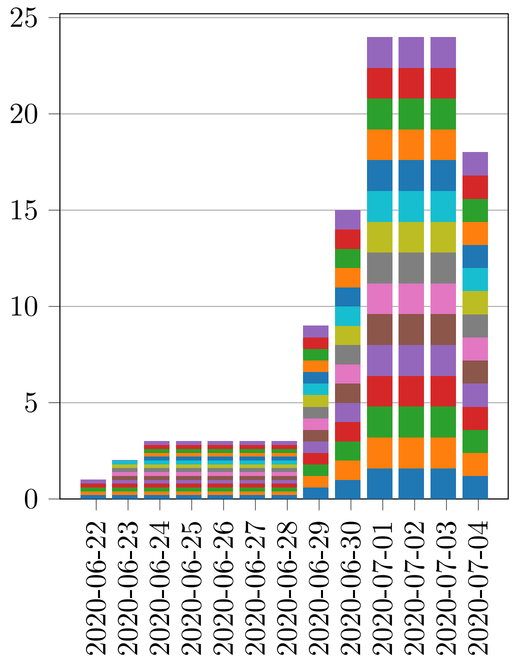
A worst case of linkable users, which could have happened in theory, was described in [Hue20a]. With a multiplier of 10 and a minimum package size of 140, 13 keys from user A (which was the maximum amount as the key for the current day was not uploaded) and 1 key from user B (this means the app has been used for one day before uploading keys). This would allow to link 12 keys of user A without doubt.

However, this problem eventually solved itself as more and more infections were reporting via the app, so today no padding is applied and shifting is only performed during times when most recipients of warnings would be asleep anyway.

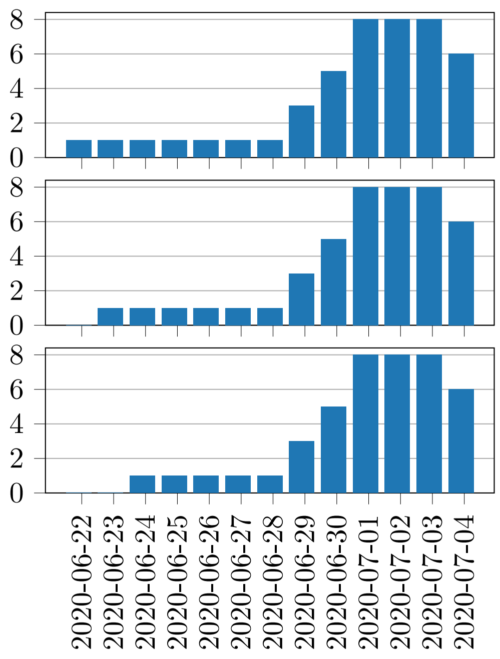
3.3.4. Server-Side Attacks and Gossip Attack

In addition to attacks which can be performed by users, the paper [ABIV21] also theorized a class of attacks performed entirely on the server side, by a malicious operator colluding with the health authorities. These include the Brutus attack, where the verification mechanism is abused to link the rolling identifiers or pseudonyms of users to personally identifiable information, and the Bombolo attack, where additional information such as number of contacts and information about which people have met each other is extracted on the server side. As stated in Section 2.3, only [Neu20] considers this class of attack in their guidelines.

The final "attack" presented by [ABIV21] is the scenario of a user being able to gather evidence about a contact with an infected person. First theorized in [Pie20, p. 9], conversely, this "weakness" of a scheme could be turned into an advantage: it would enable an additional confirmation of encounters to third parties such as testing labs.



(a) Raw data



(b) Sorted by user

Figure 3.1.: Transmission Risk Levels of CWA diagnosis keys from July 5, 2020

4. TraceCORONA Implementation

Following the last chapters, in which the existing work has been detailed, the focus now shifts to the TraceCORONA protocol and the prototype implementation of the TraceCORONA app developed by a team of researchers at TU Darmstadt as well as the author of this thesis. In this chapter the app is described in detail, commenting on architectural and design decisions. After providing general information about the implementation, the chapter begins with a high-level overview of the app's architecture, after which the packages and functional units of the app are explored in detail, one after the other.

4.1. General Information

The Android platform was chosen for our prototype implementation due to the difficulty of running applications in the background on iOS (cf. Section 2.2.5), as well as the wider variety of tools available for development and testing on the platform compared to iOS. Android is also more widely used than iOS, and other mobile operating systems have too low of a market share to make testing a prototype implementation worthwhile [Sta21].

Due to familiarity with the platform, the TraceCORONA app is written as a native Android application using Kotlin as a language. It has a total of 4983 lines of Kotlin code, along with 1832 lines of XML code used for user interface files, constant values and graphics.

4.2. Architecture and Packages

Figure 4.1 shows a simplified package diagram[1]. The Android Application Programming Interface (API) implements the architectural style of Model-View-ViewModel, introduced by Microsoft in 2005 [Gos05], which is also the basis for the architecture of our app. Following this principle, the "model" is implemented in the `database` package, with subpackages for the respective data types. The user interface is implemented in the `view`, `viewmodels` and `listadapters` packages, which we group in the diagram for sake of clarity. Utilizing these central parts of the app, the core business logic and functionality is implemented in the `tracing`, `encounterdownload`, `infectionverification` and `encounterupload` packages. Supplemental logic such as interfaces to cryptographic libraries and server connection functionality is contained in the `util` package. Finally, code such as views and logging facilities used only for debugging purposes are contained in `debugging`, which we omit in the figure.

[1]A full package diagram for an older version can be found in Section B

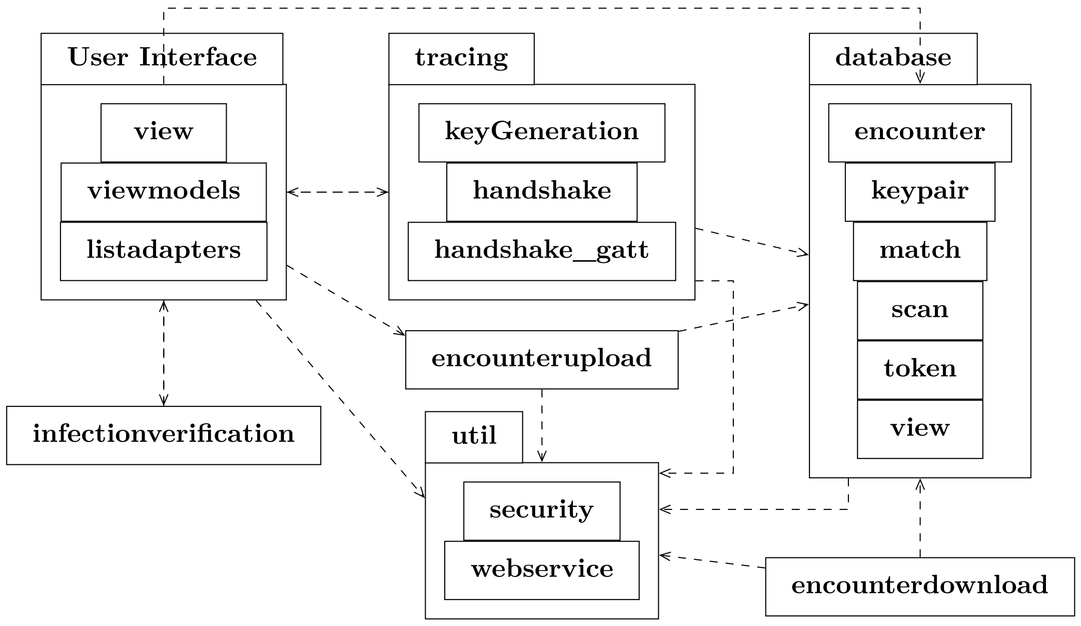


Figure 4.1.: Simplified package diagram of TraceCORONA app

4.3. Database

Design

The current database scheme is depicted in Figure 4.2. We briefly explain the design decisions leading to this version of the database. Initially, the only tables used were `Encounter`, `Scan`, and `EncounterMatch` with their respective relationships. `token` and `tokenHash` were immediately calculated after an encounter and saved directly to `Encounter`, with `tokenHash` serving as a primary key. `Encounter` also contains the `rollingId` of the encountered device, used to distinguish recently-encountered devices, where a handshake has already been performed, from new devices (cf. Section 2.4.1).

One or more `Scan` entities are present for every encounter, generated on every repeated Bluetooth encounter with the other device and containing a timestamp, Received Signal Strength Indication (RSSI) signal strength data, and a distance value (for now unused and always 0). Separately storing these results can be used for more fine-grained classification of encounters by using historic data.

If an encounter matches a token marked as infected through the server, the matching process generates an `EncounterMatch` containing the `nonce` of the respective message and the decrypted `chainStatus`.

In these early versions, as the keys and tokens were generated and calculated on the fly, no saving of keypairs and encountered public keys was necessary. Later, the database for TraceCORONA was redesigned and optimized, yielding the final database scheme seen in Figure 4.2. To preserve battery life, `Keypairs` are pre-generated before usage, and so need to be stored in the database. When a device is encountered, the received public key and rolling ID is stored in the database. Before encounter up-/download and matching, tokens and token hashes are generated, transforming `EncounterTokenParameters` entities into `EncounterTokens`.

Implementation

The database for the TraceCORONA app is implemented using SQLite [Hip21] and the Room object-relational mapper [Goo21j], a standard solution for mass data storage on

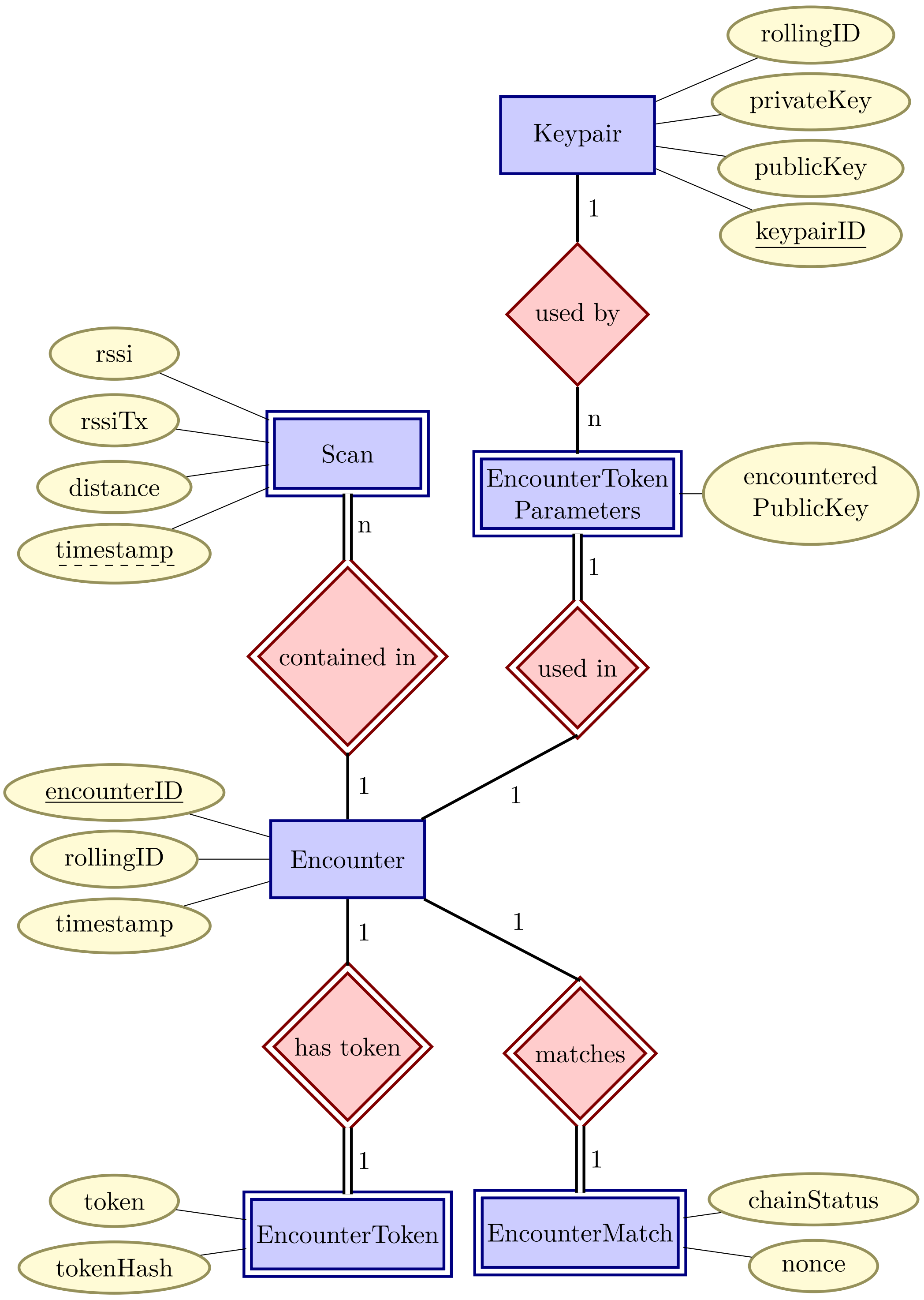


Figure 4.2.: TraceCORONA Entity Relationship diagram

Android. This allows for more straightforward access and interpretations of data than a proprietary format, as well as easier debugging.

Following the conventions for Room, we implement the database entities/tables as Kotlin data classes, while queries are implemented as functions in the corresponding Data Access Object (DAO). Apart from regular getters and setters, storing data in SQLite allows us to use database queries to retrieve aggregate values, such as the duration of all encounters in a certain interval, without manually filtering data in code. Another feature provided by Room are live queries using the *LiveData* interface of Android, which are used to automatically update the user interface when the database changes.

Every DAO is then wrapped in a *repository*, which supplements the SQL queries with functionality such as converting time to keypair ID (cf. Section 4.5) or converting the `EncounterTokenParameters` into `EncounterTokens`.

4.4. User Interface

Design

To achieve a usable and presentable interface quickly, and due to the robust integration into the Android framework, the interface design of the TraceCORONA app is based on Google's Material Design system [Goo21g]. It consists of a number of single-purpose screens linked together by an overview using card components, which are well-suited to group related information and buttons [Goo21d].

A tutorial is launched on the first start of the app only and provides an introduction on the functionality and usage to the user. Once the tutorial is completed, the user enters the main screen (see Figure 4.3(a)). Here, contact establishment (cf. Section 4.5) can be enabled with the main floating action button [Goo21c] "Start Tracing".

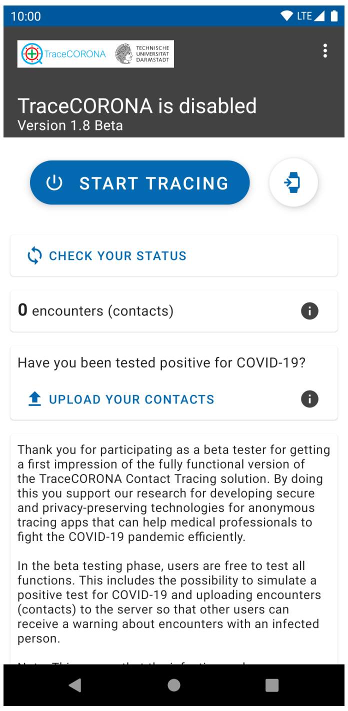
After touching "Check Your Status", the user is taken to the status screen (see Figure 4.3(c)), where the encounter download and matching (cf. Section 4.6) is triggered and a message according to the user's exposure status is displayed. If at least one encounter is present in the database, a "More Statistics" button is shown, with which the user can see a list of daily encounter count and duration. Lastly, the "Upload Your Contacts" button launches the Transaction Authentication Number (TAN) entry screen (see Figure 4.3(b)) for infection verification and subsequent encounter upload (cf. Section 4.7).

Implementation

The Material Components for Android library [Goo21i] is used to extend the Android SDK and Android Jetpack support libraries' built-in functionality. The user interface is implemented using activities, with the only part using a fragment-based interface being the `WelcomeActivity` for the tutorial.

Every activity represents a screen as described in the Design part. To prevent loss of information when the activity is destroyed, e.g., if the device orientation changes, *view models* are used to provide and persist data. `MainActivity` and `AlertsActivity` (see Figure 4.3(a) and Figure 4.3(a)) use a dedicated view model to store information only needed for the specific screen, while separate view models are used for information from the database. These handle the database connection and wrap functions and values of the repositories (cf. Section 4.3).

LiveData values provided by the view models reduce the boilerplate code, as the *Observer* pattern can now be used in the activities to update the user interface if values change in



TraceCORONA is disabled
Version 1.8 Beta

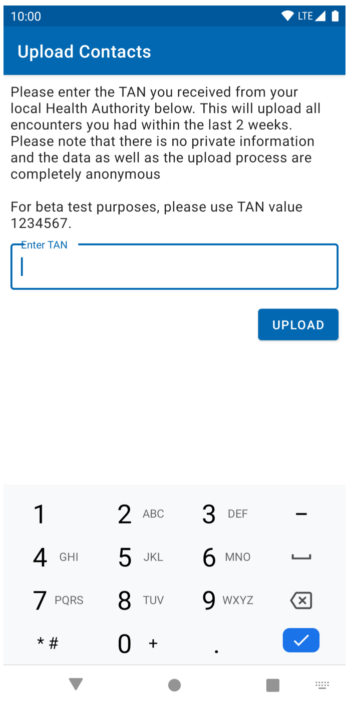
0 encounters (contacts)

Have you been tested positive for COVID-19?

Thank you for participating as a beta tester for getting a first impression of the fully functional version of the TraceCORONA Contact Tracing solution. By doing this you support our research for developing secure and privacy-preserving technologies for anonymous tracing apps that can help medical professionals to fight the COVID-19 pandemic efficiently.

In the beta testing phase, users are free to test all functions. This includes the possibility to simulate a positive test for COVID-19 and uploading encounters (contacts) to the server so that other users can receive a warning about encounters with an infected person.

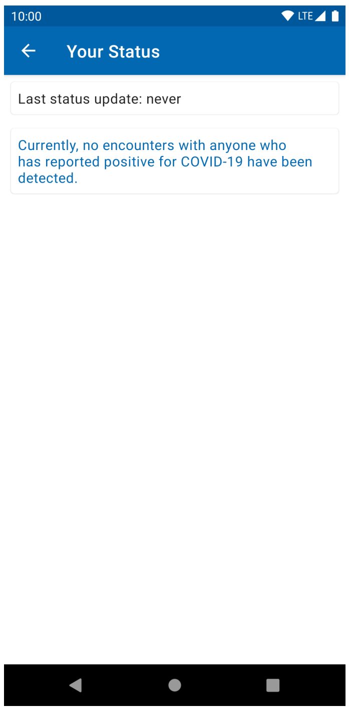
(a) Main screen (`MainActivity`)



Please enter the TAN you received from your local Health Authority below. This will upload all encounters you had within the last 2 weeks. Please note that there is no private information and the data as well as the upload process are completely anonymous

For beta test purposes, please use TAN value 1234567.

(b) Upload Contacts (`UploadContactsActivity`)



Last status update: never

Currently, no encounters with anyone who has reported positive for COVID-19 have been detected.

(c) Status screen (`AlertsActivity`)

Figure 4.3.: Screenshots of TraceCORONA user interface

another part of the app, e.g., the database. Additionally, a separate instance of *Shared-Preferences* is used for the user interface to persist values such as if the tutorial has been shown yet or when the matching process was last run.

To implement lists of values, we use the *RecyclerView* provided by Android Jetpack, which requires us to implement *list adapters*. These in turn use the view models to manage the layout and content of lists.

4.5. Contact Establishment

The `tracing` package contains the core logic used for Bluetooth contact establishment and logging. It is organized into three subpackages: `handshake`, `handshake.gatt` and `keyGeneration`. The root, as well as the former two of these subpackages form a multi-layered structure handling the different layers of Bluetooth Low Energy (BLE or Bluetooth LE) communication. The top level contains the classes `TraceWorkManager` and `TraceService`, which are responsible for regular key rotation and scanning/advertising intervals. One level below, `handshake` handles the connection process with scanning and advertising, while the nested `gatt` package is used for key exchange through the Bluetooth Generic Attribute Profile (GATT) protocol after initial discovery.

Prerequisites

Before enabling contact establishment, several other requirements have to be fulfilled. The application checks if Bluetooth is enabled and prompts the user to enable it otherwise. Another check is performed for the permission to read the fine device location, required by Android for usage of Bluetooth LE, as Bluetooth beacons may reveal the device location. For the application to run in the background without risk of being interrupted by Android's battery optimization, the user has to explicitly exempt this app from being optimized in the device settings.

The final requirement to use the TraceCORONA app is support for Bluetooth LE advertising in the device's Bluetooth chipset. Multi-advertisement support, i.e., the ability to advertise for multiple services and with multiple message parameters at the same time, is recommended so that other devices and apps can function as normal while TraceCORONA is advertising. The Android developer documentation even states that support for multi-advertisement should be queried "to check whether LE Advertising is supported on [the] device"[Goo21b]. However, devices exist where advertisement is supported, but multi-advertisement is not, e.g., recent Nokia phones per our tests. A toggle was added to the app to override this check, still, advertising may be unreliable when used with these phones.

Communication

Bluetooth communication is handled according to the protocol design described in Section 2.4.1. For the discovery phase, where no connection is active, Advertisements are handled by the `Advertiser` class and scanning is handled by the `Scanner` class. Once a device is discovered, the `Discovery` class then checks if the device is known and triggers the connection process for the handshake if not.

For this key exchange, the scanning side uses `GattClient`, which provides an `android.bluetooth.BluetoothGattCallback` to control the connection to the `GattServer` provided by the advertising device. The exchanged keys and parameters are subsequently saved to the database as an `Encounter` with attached `Scan` and `EncounterTokenParameters` entities.

The two `WorkManager` classes, despite their name, inherit from the `androidx.work.Worker` class with the companion object providing a `startWorker` function for initialization, which communicates with `androidx.work.WorkManager` for scheduling. The `TraceWorkManager` restarts the `TraceService` every 30 minutes to regenerate the BLE device address and switch to a new keypair and rolling ID.

Key Generation

Key generation is a mostly separate part, being similar in architecture to the top level package with the `KeyGenerationService` being started by the `WorkManager`, which then actually generates keys and saves them to the database. Keys are generated every two days by default, and for further optimization `WorkManager` allows to restrict the execution to times when a battery charger is connected. Keypairs consist of a public and private key, and are supplied by the Elliptic Curve Diffie-Hellman (ECDH) provider included in the Bouncy Castle crypto package [Leg20], wrapped in the `util.security.ECDHUtil` class.

Every keypair is assigned an ID, also used for selecting the correct keypair planned for the current time window: the first keypair has ID 0 and subsequent keypairs count up. The timestamp of the first tracing activation is saved to the `SharedPreferences`, and the time window for every key is hard coded to 30 minutes. Thus, the ID of the active keypair can be calculated as $\text{ID}_{\text{Keypair}} = (t_{\text{current}} - t_{\text{1st tracing}})/30\,\text{min}$.

4.6. Encounter Download and Matching

All server communication in the app is handled by the Retrofit 2 [Squ20] and Gson [Goo21f] libraries. Once the user initiates the encounter download process by opening the alerts screen, the `encounterdownload.DownloadService`, inheriting from `util.Webservice`, is started and establishes a connection to the TraceCORONA server. If a connection to the server is possible, the encounters are streamed from the web server to the app, where they are asynchronously matched to the locally recorded encounters.

Matching is performed by querying the database for `EncounterToken`s with each downloaded encounter hash. When a match is found, the encrypted random key (see Section 2.4.6) is used to decrypt the state and nonce information contained in the encounter data. These data are subsequently saved to the database as an `EncounterMatch` entity.

Every 41 seconds, a timeout is enforced: if no data was received for the last 40 seconds, the connection is closed by the client. The streaming system allows for operations on the server and client side to be parallelized, as no full file has to be downloaded, then extracted.

4.7. Infection Verification and Encounter Upload

On the TAN entry screen (`UploadContactsActivity`), after the user has entered their TAN and confirmed with the "Upload" button, the `infectionverification.NonceTanService`, again inheriting from `util.Webservice`, is called to send the TAN to the server and retrieve the nonce value as described in Section 2.4.3. If this query succeeds, the upload process is started and on failure an error message is displayed to the user.

The next steps are performed in the `encounterupload.UploadService`. This service, which is the final descendent of `util.Webservice`, begins by generating the upload message as described in Section 2.4.4. After the upload message is generated, the `UploadService` sends it to the server. Finally, the status of the local device is set to "Infected" in the app-wide `SharedPreferences` key-value store.

5. Comparative Design and Implementation Analysis

The aim of this chapter is to establish claims about the performance, privacy, security, and efficiency claims of the contact tracing protocols Google/Apple Exposure Notification(s) (GAEN) and TraceCORONA, highlighting similarities and differences in both protocols. These claims are going to be verified by means of formal and practical analysis in the following chapters. As the GAEN specification only describes the device-side communication and processing, we use the German Corona-Warn-App (CWA) as a real-world example for an implementation.

Figure 5.1 shows a high-level schematic overview of a decentralized tracing system. The following sections each describe and analyze a single part of this overview, as described by the section numbers in the figure.

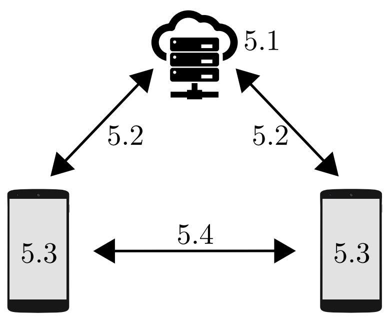


Figure 5.1.: Schematic overview of decentralized contact tracing

5.1. Server Side

As both GAEN and TraceCORONA are decentralized tracing protocols (cf. Section 2.2.2), the server side for both protocols has the purposes of:

(i) verifying positive test results

(ii) receiving identifiers used for matching encounters, verifying their authenticity and saving them

(iii) distributing these identifiers among devices

Purpose (i) is achieved in both systems by issuing a Transaction Authentication Number (TAN) for every positive test result. As this mechanism is not part of the TraceCORONA protocol and prototype, as well as highly country-specific, we are not going into further detail on this point. For more details on the CWA's implementation, refer to Section 3.1.4.

Receiving and Verifying Keys/Tokens

The functionality for achieving Purpose (ii) is handled in the CWA by the submission service [Cor20b]. After receiving a TAN-authenticated submission message containing a number of Temporary Exposure Keys (TEKs), the key metadata are sanity-checked. The Days Since Onset of Symptoms (DSOS) value, used for calculation of the infection risk, is converted into a Transmission Risk Level (TRL) value, or vice versa, for compatibility between old and new versions of the application and GAEN framework. Previously, before October 18, 2020, padding was applied with the same metadata as the uploaded keys (see Section 2.2.4), however this is disabled in the current deployment, presumably to save bandwidth. The results are saved into a PostgreSQL database twice: for distribution to CWA users and upload to the European federation gateway.

The raw data saved into the `diagnosis_key` table (used for distribution) consists of 17 B of key data (16 B of key + 1 B byte overhead [The21a]) and a minimum of 113 B of metadata (more if multiple countries were visited by the patient). On a local install of the CWA server, the total table size for 67.315 records of automatically generated fake keys is 13 074 432 B bytes. When adding an additional 9607 fake keys, the size grows to 14 352 384 B. From the difference, we can estimate the average real size of a diagnosis key stored in the database at $\approx$ 133 B. Although federation between countries is not further explored, the `federation_upload_key` table is nearly identical, only adding a so-called `batch_tag` for organization purposes.

In TraceCORONA, as explained in Section 2.4, the submission of keys is a two-step process. Before keys can be sent from a device, the TAN is exchanged for a randomly-generated nonce. Once the server receives the upload message, it decrypts the encrypted part consisting of the infection state and the nonce, verifies the nonce is genuine, and then saves the full messages into a MongoDB database. Further techniques to prevent key linking such as padding or mixing of messages are not currently implemented in the prototype, but could be added later.

Distributing Keys/Tokens

Fulfilling the third purpose, distribution of identifiers among devices, is a challenging step due to the number of app users present. Here, the two implementations employ vastly different approaches. In the CWA system, the format for transmitting TEKs is specified by the GAEN framework: they must be packed as a ZIP file containing the keys in a binary file format and a signature. These files are generated hourly and daily by the back end and then served through a Content Delivery Network, which distributes the load away from central database servers and towards dedicated infrastructure.

In contrast, the TraceCORONA server back end streams keys to client devices: once a device sends a request to download new keys beginning from a certain point, data are continually sent from a database query until the device closes the connection after a certain timeout. This architecture yields the benefit of being able to control certain parts of the response data or metadata on the server side for each client device, which can open up optimization possibilities. In addition, newly-submitted upload messages are directly distributed without any architecturally-induced delays, which is more suitable for testing and demonstrations. On the other hand, this approach cannot easily be adapted to utilize an external content delivery network, as is the case with CWA, and therefore leads to a higher server-side processing load.

5.2. Server-Device Communication

In addition to the server side software, the communications between the server and device play an important role for constraints and tradeoffs of the tracing system. The two parameters relevant in this part are the bandwidth, measured in bit/s, and the *traffic volume*, measured in bits or bytes (B). For the contact tracing protocols analyzed here, bandwidth is mostly relevant for estimating the impact on the internet and server caused by simultaneous communications, and traffic volume is relevant on the server and client sides, as it is commonly used as a billing unit for servers and mobile data plans[1].

For decentralized contact tracing, there are two major traffic-generating operations: uploading encounters to the server and downloading encounters from the server. In the two applications analyzed here all other operations are either unnecessary for the tracing itself, such as fetching statistical data for display to the user, or specific to the local implementations, such as verification of infection. We estimate the traffic generated as a function of the user count n, the rate of infected users λ, and additional variables related to the data shared by the system. The total traffic volume and bandwidth scales up with the amount of users and/or infected users.

After estimating and comparing the bandwidth required for both systems in the first sections, in Section 5.2.5 we highlight another aspect of server-to-device communication: preventing passive attackers from gaining information about user status.

5.2.1. Generalized Worst Case

To put an upper bound on the traffic volume required by users, we take a look at a generalized “worst case protocol” first. In this protocol, both sides record each encounter with a size of s_{enc}. For longer encounters, these data are repeated for every time slot. In case of an infection, all encounter data are distributed to all other users. In the worst case, every user encounters every other user, i.e., every additional user can generate $n-1$ additional encounters. Once an infection is registered, this increased number of encounters is then distributed to all devices in the system, leading to n transfers for every encounter. Again assuming the worst case – everyone encounters everyone and is then afterwards simultaneously diagnosed as positive – yields the maximum traffic volume a distributed contact tracing system can produce per time slot, referred to as $\tau_{\max}$ in Equation 5.1.

$$\tau_{\max}(n) = \underbrace{n}_{\substack{\text{Transmissions}\\ \text{per Enc.}}} \underbrace{n(n-1)}_{\substack{\text{Number}\\ \text{of Enc.}}} \underbrace{s_{\text{enc}}}_{\substack{\text{Size of}\\ \text{Enc. Data}}} \tag{5.1}$$

Leaving out the factor n for the transmissions yields the traffic volume per user, while dividing by the length of the time slots yields the average bandwidth used by this system. Therefore, this generalized distributed contact tracing system's traffic per time slot is in $O(n^3)$.

Of course, if everyone is infected, tracing contacts does not make sense anymore. Therefore, we introduce the rate of infected users, λ, which is the amount of newly-infected users in this time slot divided by the amount of total users. If a system is encounter-based, we additionally need the average encounters for the uploaded time period per person, $\mu_{\text{enc}} = \mu_{\text{enc/day}} \cdot n_{\text{days}}$. These variables will be used in the equations describing the systems in the next sections.

5.2.2. GAEN

In GAEN, to reduce the amount of traffic generated, the system uses keys rotated daily (TEKs) to then generate the identifiers used in communication between devices (Rolling Proximity Identifiers (RPIs)). Thus, for server-device communication the size of these TEKs and the corresponding metadata is relevant. As mentioned above in Section 5.1,

[1]Note, however, that the traffic of government-supported tracing apps is often zero-rated by mobile service providers, i.e., users are not billed [Hol20; CJG20].

Constant	Value
$m_{\text{CWA, down}}$	33 B
$t_{\text{CWA, down}}$	52 B
$\tau_{\text{CWA, up, 14 keys}}$	488 B

Table 5.1.: Corona-Warn-App encounter token upload and download payload size

the GAEN specification includes an exact file format to be used for download of TEKs from the server [Goo20d; App21b]. For efficient encoding of data this binary file uses the Protocol Buffers library [Goo21h], which allows for defining custom binary protocols. For every key this format includes the key data (16 B) as well as the metadata about when the key was active (4 B + 1 to 2 B), the type of diagnosis (1 B), and the TRL/DSOS values (see Section 3.1.4, 1 B + 1 B). As every field includes a 1 B header, this adds up to 31 to 32 B per key.

When generating messages for download by the client application, the keys are packaged into an export message. This includes a time window for the keys in the message (8 B+8 B), a region string (2 B), batch values for splitting the file into multiple parts (1 to 5 B + 1 to 5 B), information strings about the signature of the file (2 B + 3 B + 19 B), and the keys (31 to 32 B each, see above). Again adding the sizes together, considering strings and embedded messages up to 127 bytes in length have 2 B total overhead, the total length of an export message is at least $t_{\text{CWA, down}} + n_{\text{keys}} \cdot m_{\text{CWA, down}} = 52\,\text{B} + n_{\text{keys}} \cdot 33\,\text{B}$.

Messages used for uploading of keys, on the other hand, are not standardized between implementations. Here CWA also uses a format based on Protocol Buffers. In addition to up to 14/15 keys[2] in the TEK format described above, every upload message includes strings for the visited countries (2 B each) and the origin country (2 B), a flag controlling if the message contents are to be federated between different national tracing applications (1 B), as well as a padding string[3] to reduce side channels based on message sizes (28 B per missing key to pad out to 14 or 15 keys[2]). Adding these numbers, as well as headers, yields an upload message size of minimum $405\,\text{B} + n_{\text{countries}} \cdot 4\,\text{B}$ for one key valid for under $\approx 21\,\text{h}$ and maximum $484\,\text{B} + n_{\text{countries}} \cdot 4\,\text{B}$ for 14 keys valid a full day each. For comparison purposes, we assume $n_{\text{countries}} = 1$, which results in $\tau_{\text{CWA, up, 14 keys}} = 488\,\text{B}$.

Notably, these numbers should be identical or very close due to the padding applied during upload. However, at the time of writing, two bugs in the padding behavior were discovered by the author. The amount of keys to be padded to is not consistent between the Android and iOS apps – 15 keys for Android and 14 for iOS – therefore making the message size platform-dependent in practice. In addition, the estimate for the additional length one key adds is set at 28 B, which, due to using variable-length encoding, is too low for the actual values occurring in the data structure, and was estimated above at 33 to 34 B per key when considering the Protocol Buffers header for the embedded message. This significantly reduces the usefulness of the padding, as messages with fewer keys are still smaller. Both bugs were reported to the developers by the author on April 22, 2021 [Roo21], but no response or change in the relevant code segments occurred during the period of the thesis.

Table 5.1 summarizes the results of our payload size calculations for Corona-Warn-App. If fake key padding (cf. Section 2.2.4) is utilized, it must be considered, as it nearly multiplies

[2]The Bluetooth-based contact tracing feature uses up to 14 keys, while the more recent QR code based check-in feature extends this by one. For the sake of comparison we use 14 keys and ignore the QR code check-in.
[3]Not to be confused with fake key padding (cf. Section 2.2.4).

Constant	Value (original)	Value (no encoding)	Value (optimized)
$m_{\text{TC, down}}$	181 B	80 B	24 B
$t_{\text{TC, down}}$	0 B	0 B	0 B
$m_{\text{TC, up}}$	143 B	96 B	28 B
$t_{\text{TC, up}}$	73 B	0 B	0 B

Table 5.2.: TraceCORONA encounter token upload and download payload size

the generated traffic for key download. However, due to this form of padding not being used in the CWA at the moment, it is left out here.

5.2.3. TraceCORONA

In contrast to GAEN, TraceCORONA does not use derived keys, instead using Elliptic Curve Diffie-Hellman (ECDH) to generate a new token per encounter, which is rotated every 15 minutes. Therefore the upload and download messages must include all hashed encounter tokens. In the TraceCORONA prototype, messages are encoded using JavaScript Object Notation (JSON) and Base64 due to ease of use.

One encounter token takes 16 B and is encoded as 25 B. Additionally, the upload message contains the encrypted state and nonce values (48 B, encoded as 68 B), the key and encrypted key (16 B/25 B each) for every encounter, resulting in a payload size of $m_{\text{TC, up}} = 143\,\text{B}$. The JSON labels take up an additional $t_{\text{TC, up}} = 73\,\text{B}$ per packet.

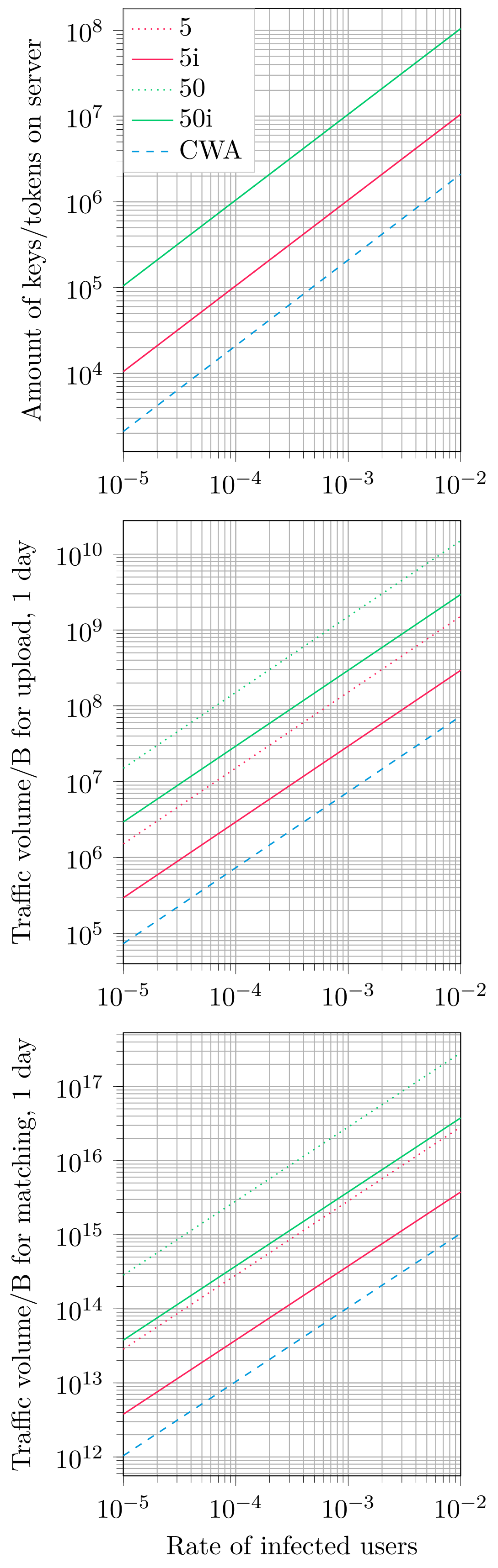
Downloads contain similar information, only omitting the plaintext key used by the server. Due to the streaming implementation (cf. Section 4.6), every encounter token takes a total of $m_{\text{TC, down}} = 181\,\text{B}$, of which 63 B are used for JSON labels and could therefore be reduced. The inclusion of the nonce data as well as the inefficient encoding leads to the download size of original TraceCORONA being $11765/533 \approx 22$ times larger for a period of 13 days and 5 encounters per day, and this is without considering the Hypertext Transport Protocol (HTTP) headers repeated for each streamed key.

If TraceCORONA instead uses a system where multiple keys are grouped into a single download package, similar to GAEN, and leaves out some unnecessary metadata sent with the key, the traffic can be reduced considerably. We estimate the metadata can be reduced to 4 B, consisting of infection state, date of encounter (as days since epoch) and calibration data. Together with the token hash and encrypted key for metadata encryption, this would then consume $m_{\text{TC, down}} = 24\,\text{B}$ for every token, which is 13.3 percent of the original size. For uploads, this would reduce the upload size to $m_{\text{TC, up}} = 28\,\text{B}$ per key. As the usage of an encoding like Protobuf is not strictly necessary for TraceCORONA, we do not assume that one is used here, while for GAEN it is mandatory.

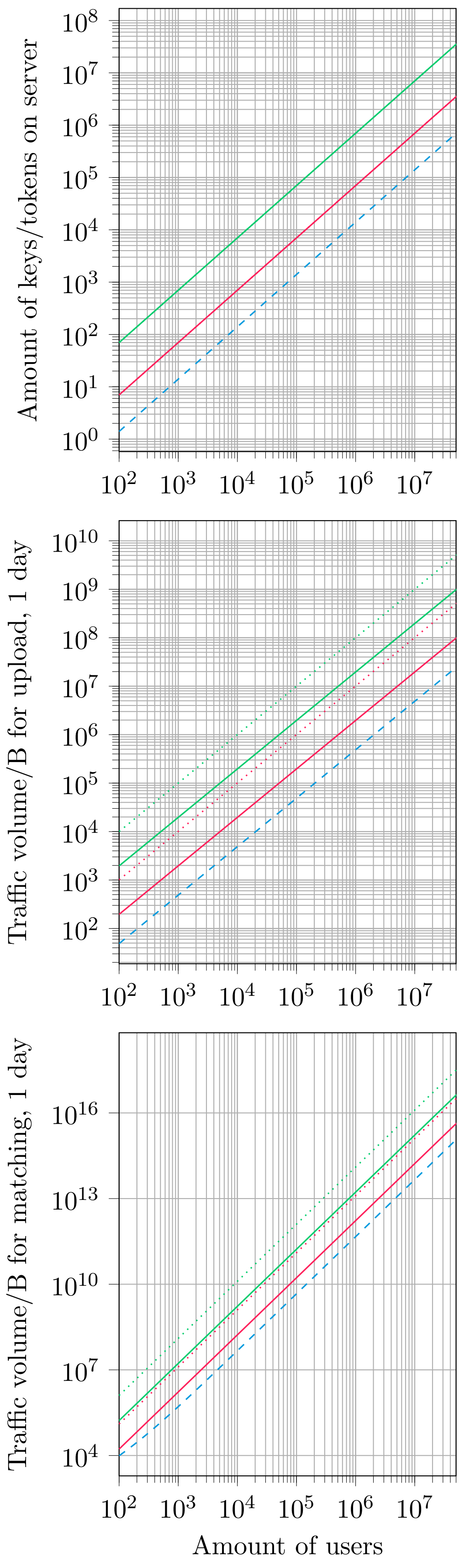
Table 5.2 contains the results of our payload calculations as described in the paragraphs above.

5.2.4. Comparison of Generated Traffic

After describing the transmission format used by the two protocol implementations, we compare the traffic generated by the two systems. For this purpose, the previously-determined parameters are interpolated to realistic daily traffic volumes for a nation-wide contact tracing system. As described in Section 5.2.1, three parameters are used as variables: the total amount of users of the system (n), the rate of infected users over the analyzed time period (λ) and the average encounters per person and day ($\mu_{\text{enc/day}}$). We



(a) Varying rate of infected users, 15 000 000 users



(b) Varying amount of total users, rate of infected users = 0.001

Figure 5.2.: Comparison of TraceCORONA (raw, 5 or 50 average encounters/person/day) and Corona-Warn-App (CWA) traffic per day, based on theoretical numbers, logarithmic scale for both axes

consider one low contact scenario with $\mu_{\text{enc/day}} = 5$ and one high contact scenario with a value of $\mu_{\text{enc/day}} = 50$.

Three values are compared: the amount of keys/tokens stored on the server, as well as the total traffic volume used for the upload and matching processes. The plots in Figure 5.2 are split into two columns, where either the rate of infected users or the amount of users is varied. We plot two variants of TraceCORONA's traffic volume, one with the proposed reduced metadata size and ignoring JSON and Base64 encoding (5i and 50i, solid lines) and one with the original size (5 and 50, dotted lines). Lines are colored according to the legend, where the numbered lines stand for the TraceCORONA results with the respective $\mu_{\text{enc/day}}$ value and the dashed CWA line stands for the CWA results.

The amount of keys or encounter tokens is estimated by the functions

$$\begin{aligned} n_{\text{tokens, TC}}(\mu_{\text{enc/day}}) &= \mu_{\text{enc/day}} \cdot n_{\text{days}} \\ n_{\text{tokens, TC, total}}(n, \lambda, \mu_{\text{enc/day}}) &= n \cdot \lambda \cdot n_{\text{tokens, TC}}(\mu_{\text{enc/day}}) \\ n_{\text{keys, CWA, total}}(n, \lambda) &= n \cdot \lambda \cdot n_{\text{days}} \end{aligned}$$

and is plotted in the first row of Figure 5.2. As expected, due to the vastly different approaches of TraceCORONA and GAEN, the amount of cryptographic identifiers stored on the server is 2-3 orders of magnitude higher for TraceCORONA. Note that this calculation does not change for the improved versions of TraceCORONA, therefore the dotted and straight lines are on top of each other.

The upload traffic volume in both systems is calculated by the following equations. Note here that CWA always uses the same upload size for padding, so even if users only upload one week of diagnosis keys, the upload traffic does not change significantly:

$$\begin{aligned} \tau_{\text{TC, up}}(n, \lambda, \mu_{\text{enc/day}}) &= n \cdot \lambda \cdot (m_{\text{TC, up}} \cdot n_{\text{tokens, TC}}(\mu_{\text{enc/day}}) + t_{\text{TC, up}}) \\ \tau_{\text{CWA, up}}(n, \lambda) &= n \cdot \lambda \cdot \tau_{\text{CWA, up, 14 keys}} \end{aligned}$$

Finally, when examining the download traffic volume, a reason for keeping the number of keys or tokens to a minimum is revealed – the traffic increases quadratically with the number of users. This phenomenon is apparent in the function used for estimation, as the amount of keys/tokens is already dependent on the amount of users, and every user downloads every key/token:

$$\begin{aligned} \tau_{\text{TC, down}}(n, \lambda, \mu_{\text{enc/day}}) &= n \cdot (m_{\text{TC, down}} \cdot n_{\text{tokens, TC, total}}(n, \lambda, \mu_{\text{enc/day}}) + t_{\text{TC, down}}) \\ \tau_{\text{CWA, down}}(n, \lambda) &= n \cdot (m_{\text{CWA, down}} \cdot n_{\text{keys, CWA, total}}(n, \lambda) + t_{\text{CWA, down}}) \end{aligned}$$

Both for upload and matching we can see that CWA is the most efficient system in terms of traffic. However, the difference to the improved version of TraceCORONA is under one order of magnitude in the low contact scenario, so TraceCORONA does not generate unreasonable traffic. In the high contact scenario, the difference is under two orders of magnitude. Nevertheless, 50 encounters per day is a large number to be reached with contacts away from home. The prototyped version of TraceCORONA generates a much larger amount of traffic (1-3 orders of magnitude higher than CWA), while theoretically allowing for recursive tracing. This function was not implemented in any decentralized contact tracing system known to us.

Real values which can be used for interpretation of these results are provided by CWA download numbers [Hoe21], incidence values [Wor21] and surveys such as [MHJ+08] or [Rot20] respectively. Note that the definition of encounters is different between the studies and the actual criteria applied on devices. For TraceCORONA, on one hand signal strength and duration are present as metadata to filter encounters, on the other hand long encounters can be registered as multiple encounters due to periodic key rotation.

Also note that this theoretical comparison excludes headers and verification data such as TANs. As TraceCORONA's streaming implementation incurs additional increased overhead from packet frames and headers, the discrepancies between the current version of TraceCORONA and GAEN are likely larger in practice. To verify our results and to generate a more complete picture we conduct real measurements and compare the results with this analysis in Chapter 7.

5.2.5. Plausible Deniability and Playbooks

One more aspect belonging to the category of server-to-device communication is the notion of masking the infection status of a person from an observer to the communication. Although the contents of the communication are secured by Transport Layer Security (TLS), without proper precautions the metadata can be used to infer the infection status of individual users. To combat this, systems may send fake upload messages appearing to confirm an infection and sending the ephemeral IDs. Additionally, when a device is retrieving test results from the server, the responses should have the same size and thus be indistinguishable for the attacker. As this aspect is not implemented into TraceCORONA yet, the analysis in this section highlights CWA's approach to this problem.

Fundamentally, this area involves two security properties of a protocol: plausible deniability and observational equivalence. Plausible deniability is the concept of the attacker not being able to confirm that a user has a certain property, i.e., knowledge of a secret. In the case of contact tracing, we can view the possession of a positive test result as the secret that the user wants to keep, but still be able to transmit to the server. Observational equivalence is a related notion in which an attacker is not able to distinguish two processes in a communication protocol, i.e., a fake transmission and a real one. Currently, the CWA does not use plausible deniability, the feature was disabled before release. If enabled and configured so that the feature is actually effective, i.e. every user would have a random chance to send a fake upload, the bandwidth usage of the system would increase by a large amount.

In CWA, every communication with the server follows an identical pattern enforced by the so-called playbook system. First, two requests are made to the verification server, then one request is made to the submission server. All fake requests indicate their nature to the server in a HTTP header and are consequently answered with a realistic response. However, for the submission service one detail is not respected: the check-in system added later adds two headers to the response [Cor21c] which are not present in the response to the fake message [Cor20c]. This bug was reported to SAP on October 4, 2021. Therefore, an attacker can distinguish a fake submission in two ways: first, the submission payload size is always 423 B, as opposed to the variable size described in Section 5.2.2, and second, the response size is smaller than for real submissions.

Overall, the criteria of plausible deniability and observational equivalence are not fulfilled by the current version of CWA. It is possible, however, to update the application and server side to effectively mitigate the possibility of network observers revealing infected users.

5.3. Device Application

The aspects of the systems discussed in the previous sections all involve the server side. In this and the next section we focus on the facets of the device applications, as well as the communications between them.

App Name	Country	Framework	Derived from
Stopp Corona	Austria	Native	
Coronalert	Belgium	Native	CWA
Stop COVID-19	Croatia	Native	
eRouška	Czechia	Native	
HOIA	Estonia	Native	
Koronavilkku	Finland	Native	
Corona-Warn-App	Germany	Native	
Immuni	Italy	Native	
Apturi COVID	Latvia	Native	
COVID Alert	Malta	Native	SwissCovid
CoronaMelder	Netherlands	Native	
ProteGO Safe	Poland	Native	
#OstaniZdrav	Slovenia	Native	CWA
Radar Covid	Spain	Native	
SwissCovid	Switzerland	Native	
NHS COVID-19 App	UK	Native	
COVID Alert	Canada	React Native	
CovTracer-EN	Cyprus	React Native	MIT PathCheck
Rakning C-19	Iceland	React Native	
COVID Tracker	Ireland	React Native	
StayAway COVID	Portugal	React Native	
SmitteStop	Denmark	Xamarin	
Smittestopp	Norway	Xamarin	

Table 5.3.: Frameworks used by open-source official national tracing apps

5.3.1. Multi-Platform Frameworks

Mobile Operating System (OS) such as iOS and Android enforce a strict design and layout of application packages. Apart from using the official native Software Development Kit (SDK) of the platform, there are additional frameworks available for sharing code between platforms, with the currently most widely-used being Facebook's React Native, Microsoft's Xamarin and Google's Flutter. We surveyed a number of official European contact tracing applications that published their source code. Among 23 open-source national contact tracing applications implementing the GAEN Application Programming Interface (API), 16 were using platform native SDKs, 5 were using React Native and 2 were using Xamarin.

We believe this low adoption of multi-platform frameworks to be related to performance considerations, as well as unavailability of GAEN in these frameworks, requiring native code for integration. Additionally, a main benefit of multi-platform frameworks is the implementation of web and/or desktop applications for PCs in the same codebase, which is not relevant to COVID tracing apps where usage is confined to cellphones.

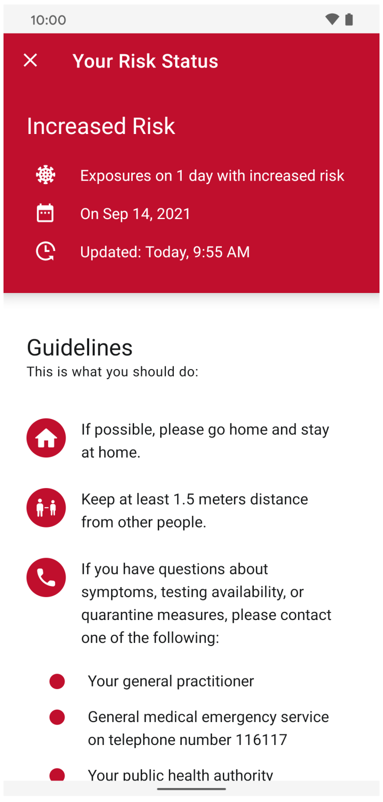
The TraceCORONA prototype, as stated in Section 4.2, is implemented as a native Android app. Therefore, for device performance comparisons between CWA and TraceCORONA, as in Chapter 7, the framework is not a variable to be accounted for.

5.3.2. Risk Score Calculation

An aspect not implemented in the current prototype of the TraceCORONA app is the classification of encounters by distance and time to minimize false positives if infection is unlikely. GAEN provides a system to classify *exposureWindows*, which is the term for single encounters inside a rolling window, by report type, infectiousness, attenuation or distance, and time.

As stated in Section 5.2.2, every TEK is assigned a DSOS value and a report type by the server. The DSOS value is converted into an infectiousness value of “none”, “low” or “high” by the GAEN framework, and the report type is used unchanged as either “unknown”, which leads to the key not being used for calculation, or one of four predefined values. In total, 8 different states which lead to the key being counted can be encoded by these two values. Additionally, GAEN allows weighting the time of the encounter. The time between repeated scans of the same RPI is multiplied with a value corresponding to attenuation buckets, i.e., depending on if the value falls between two defined thresholds, which are provided in advance by the app, yielding a *weightedSeconds* value. These three values can then be used either by GAEN’s built-in summary features or retrieved for manual calculation by the app.

Recent versions of GAEN and CWA implement the same logic for calculation of risk scores. The weightedSeconds value is multiplied with weights based on infectiousness and report type, which are again provided in advance by the app, resulting in the risk score. Every exposureWindow is therefore assigned a risk score, which allows for filtering of less risky encounters when computing the daily risk score. It is simply the sum of all exposureWindow scores on one day.


10:00

Your Risk Status

Increased Risk

Exposures on 1 day with increased risk

On Sep 14, 2021

Updated: Today, 9:55 AM

Guidelines

This is what you should do:

If possible, please go home and stay at home.

Keep at least 1.5 meters distance from other people.

If you have questions about symptoms, testing availability, or quarantine measures, please contact one of the following:

- Your general practitioner
- General medical emergency service on telephone number 116117
- Your public health authority

Figure 5.3.: “High risk” message of CWA, from [Cor21a]

In CWA, however, DSOS, infectiousness and report type are “abused” to encode a TRL between 1 and 8. Report types values are mapped to double the enum value, providing the upper two bits, and infectiousness is used for the lower bit. Still, the algorithm remains the same with only the server providing different values in the diagnosis key packages. ExposureWindows with a risk score of 5 to 9 minutes are classified as low risk, while higher scores are classified as high risk. Similarly, days with less than 9 minutes of risk score are low risk and days with more are high risk. If there is at least one high risk day, the user is warned with a red message (see Figure 5.3) telling to self-quarantine and observe distance rules.

Taking the absolute values for the calculation in CWA [Cor21b; Cor21d] into account, the risk score of 9 minutes is reduced to ≈ 5.63 min in ideal conditions, i.e., a TRL of 8 and a Bluetooth Low Energy (BLE or Bluetooth LE) attenuation between 63 and 72 RSSI. For lower attenuation, i.e., lower distance, this number changes to 7.031 25 min and for higher attenuations it changes to ≈ 90 min. The absolute worst case of minimum time is ≈ 150 min for the lowest eligible TRL of 3 and a high attenuation.

5.3.3. Limits of GAEN

GAEN has a unique property among contact tracing systems: it is integrated into the operating systems it runs on. This property allows for unique optimizations and mitigations for some attacks. Regardless, it is possible on Android to bypass the built-in API entirely and run a regular app which implements the same functionality. Google does not allow such apps into the Google Play Store [Goo20e], but sideloading of apps is officially supported on Android.

In detail, Google claims to detect and remove “malware that rebroadcasts BLE RPIs” as well as “apps found to be explicitly capturing BLE RPIs” [Goo20e] to counter possible relay and profiling attacks.

To prevent abuse of the official API for status-reveal attacks (see 6.1.4), the current version of GAEN limits the number of times keys can be imported to 6 per day (note that the number of keys is not limited further). Additionally, the parameters for risk score calculation can only be updated once per week [Goo21e; App20].

5.4. Communication Between Devices

Lastly, as the core technology behind Bluetooth LE-based contact tracing, the communication of devices using BLE plays a crucial role in determining the effectiveness, security and privacy properties of a system. Thus, the protocols have introduced different mechanisms to ensure the system meets its design criteria. In this section, we compare the different parts of the local communications between tracing devices.

Fundamentally, the purpose of a BLE tracing protocol is to record encounters between users’ devices in a way that allows for warning if a user is diagnosed as positive. As described in Section 2.1.1 and Section 2.2.2, the two techniques in use for this are active, in use by TraceCORONA, and passive contact establishment, which is used by GAEN-based apps.

Before any contacts can be established, one device must be advertising by sending packets and another device must be listening, or scanning, for advertisement packets (see Section 2.2.1). In GAEN, advertisements are continually sent at low transmission power “around every 250ms”[Goo20b; Goo21a]. Scanning is only active for a short period of 4 seconds [Goo20b] every 3-5 minutes (based on logs of a Pixel 3a running CWA). This limits the resolution of the encounter time to these 3-5 minutes.

In both protocols, contact establishment begins with one device discovering each other through these advertisements. The advertisement payload contains a rolling identifier, which changes every 15 minutes for GAEN and every 30 minutes in the TraceCORONA prototype. The former includes a 4-byte Associated Encrypted Metadata (AEM) value, while the latter also includes the reported transmit power level in an unencrypted form. See Section 6.3 for more details on these fields.

In both systems, the receiving device now saves the connection data and metadata to a database and continues listening until the scan interval is over. If a rolling identifier is scanned again, these data are saved indexed by the identifier to allow for profiling of the encounter later. For GAEN, this is the whole process, as the system does not establish contact between devices. TraceCORONA additionally needs to establish an encounter token, which is triggered by the scanning device not having seen the rolling identifier before.

In this process, an ECDH key exchange is performed, of which the result is the encounter token. This leads to the advantage of people being able to redact certain encounters before uploading their keys, but can also lead to an issue where a large amount of encounter tokens generated on other devices can be uploaded to the server as well. A detailed description of this attack vector and its implications can be found in Section 6.4.

6. Security Analysis

After establishing knowledge about the two protocols by looking at specifications and code, we now focus our attention on attacks possible against users of the two analyzed tracing systems. At first, we provide an overview of all attacks for both systems which were analyzed either in related work or by us in Section 6.1. Then, we perform a formal analysis of Google/Apple Exposure Notification(s) (GAEN) and TraceCORONA in Section 6.2. At the end of the chapter, we provide in-depth descriptions and discussions of a a novel attack we uncovered during our research in Section 6.4.

6.1. Known Attacks

In Table 6.1, we list all attacks known from the literature (cf. Section 3.3) which are applicable to common contact tracing solutions. We have analyzed the impact of these attacks in the GAEN or specifically Corona-Warn-App (CWA) system, and the TraceCORONA system.

The results of this analysis are described in the following four sections, each corresponding to a category of attack, sorted by outcome and technique and marked by horizontal lines in Table 6.1. Afterwards, the results of the analysis are discussed in Section 6.1.5.

6.1.1. Profiling With or Without Bluetooth LE Communication

First, we examine most[1] attacks resulting in untargeted profiling of infected users, i.e., gathering information about infected users' location history or social behavior by observing their Bluetooth Low Energy (BLE or Bluetooth LE) communication or actively communicating with them. Along this line, there are multiple aspects to be discussed.

Bluetooth LE Profiling without Server Access

In GAEN, once the users upload their diagnosis keys, an attacker can link the ephemeral IDs over the course of one day. Therefore, GAEN is vulnerable to profiling with a fixed-position BLE scanner, as was demonstrated in [BDF+20]. This attack is also called Paparazzi attack by [ABIV21]. TraceCORONA does not upload ephemeral IDs or related secrets to the server, instead using Elliptic Curve Diffie-Hellman (ECDH), which protects the established encounter tokens from passive attackers.

[1]Attacks using device time modification are described in Section 6.1.3

Outcome	Attack	GAEN	TC	Alternative names/references
P	Profile infected users using fixed-position BLE sniffers	✗[a]	✓	Privacy [BDF+20], Paparazzi [ABIV21]
P	Profile infected users using fixed-position BLE transceivers	✗[a]	○	
P	Profile infected users using malware on other devices	✗[a]	○	De-anonymization [DR20]
AP	Link uploaded TEKs/encounter tokens by Transmission Risk Level (TRL)	○	✓	Section 2.2.4, [Hue20a]
AP	Link recorded beacons of infected users by unique metadata values	✗	✗[b]	Sec. 6.3, CVE-2020-24722 Iss. 2 [Mar20]
P	Profile infected users using fixed-position BLE sniffers + server data	○	✓	Orwell [ABIV21]
P	Profile infected users using fixed-position BLE transceivers + server data	○	○	Matrix [ABIV21]
AP	Deanonymize encounter data as colluding server + health authority	○	○	Brutus [ABIV21]
P	Count infected users' encounters as colluding server + health authority	✓	○	Bombolo [ABIV21]
P	Profile infected users as colluding server + health authority	✓	✓	Bombolo [ABIV21]
I	Inject targeted fake alerts using BLE sniffer + server access	✓	✓	Matteotti [ABIV21]
I	Replay (record and broadcast later)	✗[c]	✓	
I	Relay (record and immediately broadcast)	✗	✓	Security [BDF+20], False-Positive [DR20]
AI	Modify calculation of attenuation as man-in-the-middle (replay attacker)	○[d]	✓[b]	Sec. 6.3, CVE-2020-24722 Iss. 1 [Mar20]
AI	Modify calculation of attenuation as sender	✗	✗[b]	
AI	Inverse Sybil (pose as one user on multiple devices not able to communicate)	✗	✗	[ACK+21]
AI	Sybil (pose as multiple users at the same time using only one TAN)	○	○[b]	Section 6.4
AI	Buy/sell upload of TEKs/encounter tokens using various methods	○	○	[AFV21a]
AI	Modify time of device to inflate time of encounter	✗	✗	Master of Time [IVV21a]
I	Modify time of device to inject encounters with IDs outside of validity period	✗	✗	Belated Replay [IVV21a]
I	Modify time of device to inject encounters without knowing TAN	○	✓	KISS [IVV21a]
AP	Modify time of device to gather past/future ephemeral IDs	✗	○	My-Number: Past/Future [IVV21a]
AP	Modify time of device to make it broadcast the same ID for an extended time	○	✓	My-Number: Far Future [IVV21a]
R	Capture photos of infected users using BLE transceiver + directional antenna	✗	✗	Paparazzi [Vau20], [NAE+21]
R	Annotate encounters with location, personal data, possibly share this info	✗	✗	Nerd [Vau20], Militia [Vau20]
R	Malware/trojan horse acts like tracing app, reports results to attacker	✗	✗	Biosurveillance [DR20]

P: Profiling of infected users, I: Injection of fake encounters, R: Revealing infection status of single users, A: Aid in. . .
✗: design vulnerability – can only be corrected with major changes to protocol, ○: implementation detail – can be prevented in implementation, ✓: not vulnerable
[a] linkable for 1 day, [b] not yet implemented, [c] limited to 15 minute windows, [d] probabilistic Risk: Total(Criticality,Likelyhood of Occurrence)

Table 6.1.: Overview of attacks on GAEN and TraceCORONA (TC)

To better model TraceCORONA's security, we introduce a modified attack with the malicious observer being able to send and receive BLE data (transceiver). With only a transceiver and without access to the server, TraceCORONA has one remaining way to link observed encounters: the "nonce", originally meant to allow identified contacted users to recursively continue tracing (cf. Section 2.4) and therefore able to be decrypted by all contacted users, allows for linkage of encounters. Transmitting the nonce to clients is not integral to the functionality of TraceCORONA, which is why we classify it as an implementation detail to be changed in an eventual deployment.

Both of the previous attacks have assumed a network of fixed-position scanners/transceivers. [DR20] presents an attack scenario, the so-called de-anonymization attack, in which malware included as part of advertising Software Development Kits (SDKs) interacts with users of the tracing app. As phones are able to act as transceivers, this attack is equivalent to the fixed-position transceiver without server access, the only difference being that location of the device can change, which is recorded through other means. The impact is identical to the attack above.

Linkage of Diagnosis Keys/Encounter Tokens

Next, we look at attacks which allow to link multiple encounters or diagnosis keys to a specific user or small anonymity set. Of course, this alone does not result in profiling, rather aid with processing data gathered in another attack. The first attack, described in detail in Section 2.2.4, abuses the Transmission Risk Level (TRL) system of CWA and is only applicable in very specific cases in which a very small amount of users is uploading diagnosis keys. TraceCORONA, on the other hand, is not vulnerable to this attack, as there are no daily diagnosis keys and the transmission risk level could be encoded like the nonce currently is, making it impossible for observers to link encounters by a single user.

Another attack centers on the Received Signal Strength Indication (RSSI) calibration metadata sent with users' advertisements. These values can provide another data point for linkage of multiple tokens or keys, even if transmitted in an encrypted form only visible to the encountered party. When using less common Android devices with per-device calibration data, the chance that this device is either the only instance or part of a small group of a certain model in an attacker's dataset is high. Even though calibration metadata are not used in TraceCORONA yet, we argue that both TraceCORONA in a real world deployment and GAEN would be vulnerable to this attack, as radio calibration metadata is essential to reliable RSSI-based distance estimation. [Mar20] contains this attack vector as "Issue 2".

Profiling with Bluetooth LE and Server Access

A variant of the Paparazzi attack, where the attacker operates fixed-position BLE sniffers, is the Orwell attack [ABIV21]. Here, the server is additionally under control of the attacker. Again, GAEN is vulnerable to this attack, and with server access all keys are linkable over the whole uploaded period, i.e. users can be tracked over up to 14 days.

When the attacker is able to access the server data, we run into multiple attacks where both systems fail to protect users by default. In the so-called Matrix attack [ABIV21], where the attacker has server access and operates fixed-position BLE transceivers, all uploaded data from users is linkable in both systems, which allows the attacker to link their observed/exchanged identifiers. If the health authority distributing the Transaction Authentication Numbers (TANs) for upload by users colludes with this attacker, this data can even be linked to personal identifying information collected by the testing station or hotline. The latter case is called Brutus attack in [ABIV21].

As countermeasures against these attacks, blind signatures could be used to make uploaded data unlinkable, such as in Pronto-C2 [ABIV21]. Additionally, mix networks [Cha81] operated by Non-Governmental Organizations (NGOs) can be used to then prevent linkage by IP addresses. Technical solutions such as these serve to decouple the different trust actors and make an attack with operator level access less likely. Still, the upload process of contact tracing data requires trust in the authority operating the system – if all actors of a system collude, the user cannot prevent linkage of their data. Increasing the technical debt incurred by more complicated processes is likely to harm performance and introduce bugs leading to more serious issues than would be prevented here.

Non-Bluetooth LE Profiling with Server Access: Bombolo Attack

What would happen if the attacker does not use BLE communications at all, instead only aiming to gather metadata about the encounter behavior of users and having access to server-side data by operator and health authority? This attack is known as the Bombolo attack [ABIV21]. We have separated this attack into two in the table, as a full profile of the user akin to what a BLE attacker can achieve is not possible in either system.

In GAEN-based systems, performing a Bombolo attack only tells the attacker on which days tracing was active in the uploaded time period, i.e., the last 14 days, which we do not regard as useful in any way. TraceCORONA in its current implementation would leak the number of encounter tokens the user has collected, which could lead to a rough estimate on the contact behavior of the user. Still, we do not see this as an important flaw of the system, and the original paper states on this problem: “It is hard to imagine how such leakage could be exploited by mass surveillance attacks.” [ABIV21].

Nevertheless, securing TraceCORONA is possible by always uploading the maximum amount of encounter tokens if such a limit exists and padding the data with fake encounter tokens. This in turn would increase traffic requirements of the system and prolong on-device matching. Reducing the granularity of the attack as a compromise between wasted traffic and user privacy is possible by instead defining fixed quanta of tokens to pad to.

6.1.2. Fake Alert Injection

In this category we classify attacks which result in – or help with – alerting users about contacts with infected users which never happened, i.e., generating fake alerts. For our examined systems, these attacks always require an active attacker broadcasting a BLE signal.

Full Attacks

Two attacks which do not work either on GAEN or on TraceCORONA are the passive insertion of fake alerts by reporting people in contact with the user as positive (Matteotti attack [ABIV21]), as well as the replay attack, in which attackers record transmissions and later send them at the same or a different location. The former does not work, as both systems do not upload the transmitted ephemeral ID to the server in plain, but rather a value either derived from this ephemeral ID or used to derive the ephemeral ID. These are based on security primitives currently regarded as secure. The replay attack does not work, as the ephemeral IDs used in both systems change every 15 minutes, and in the case of TraceCORONA, a two-way handshake is required, which needs a private key inaccessible to the attacker.

However, for the case of an attacker immediately forwarding and rebroadcasting received messages at a different location, we proved in [BDF+20] that GAEN allows this attack. We dubbed this relay attack “Mind the Security GAP” and a version using malware on devices was theorized in [DR20] as the False-Positive Attack. TraceCORONA is not vulnerable to a one-way relay attack, but would require a sophisticated two-way relay attack, which presents a much higher technical hurdle due to the required real-time capability and the limited capacity of common off-the-shelf BLE devices.

Partial/Supplemental Attacks

The remaining part of this section covers attacks which on their own do not lead to injection of fake encounters, but can be combined to achieve this purpose. The first subcategory of these attacks concerns itself with the Associated Encrypted Metadata (AEM) field in GAEN Bluetooth LE transmissions. It contains calibration data used to more accurately estimate the distance between users through the measured signal strength. As this field uses AES-CTR, which is an unauthenticated encryption mode, relay attackers can modify this value. However, as it is encrypted, the attacker is limited to flipping bits, hoping to affect the value in a way which helps them achieve a higher risk score. This attack vector was reported by Marsiske as “Issue 1” of CVE-2020-24722 [Mar20]. Furthermore, a malicious sender could directly influence the value used in the message. The former attack can be fully prevented in TraceCORONA by transmitting the calibration data as part of the encrypted message currently containing the infection state and nonce. The latter attack, however, could only be prevented by using a centralized registration which verifies the device model and a trusted computing concept enforcing this registration, which is difficult, if not impossible, to implement in a manner usable by lower-cost and older smartphones.

Another way of maximizing the attack impact of fake alert injection is modifying the relationship between real devices and device identities. In peer-to-peer systems, which decentralized contact tracing systems are, abusing multiple identities is called a Sybil attack. One application of this principle, dubbed “Inverse Sybil attack” [ACK+21], has an attacker model of multiple devices not being able to communicate with each other and being able to assume the same identity, therefore being able to feign contact with more users, ultimately generating fake alerts. For GAEN, this attack can neither be mitigated nor prevented, while for TraceCORONA a mitigation in the form of encounter limits is possible. The proposed solution to the issue, creation of a hash chain advancing each time period or fixed number of encounters [ACK+21], does not prevent the use of a connection between devices to synchronize these values. In the real world, an internet connection can be used for this purpose, thus making the attacker model unrealistically limited.

A novel attack is our Sybil attack, where one device pretends to be multiple devices. This attack, fully detectable and easily preventable in GAEN, is not trivial to handle in TraceCORONA due to the unlinkability of encounter tokens. When successful, an attacker can create more encounters, leading to a higher risk score when assuming the risk calculation from GAEN, thereby reducing the time it takes to inject a fake high risk alert. However, its impact is limited drastically for most attack scenarios using an encounter limit. Other countermeasures include requiring a minimum time per encounter for it to be counted, counting multiple simultaneous encounters as one, or the usage of heuristics to detect the attack. A more in-depth analysis of this attack can be found in Section 6.4.

Lastly, to realize these latter fake alert injection attacks, an attacker has to upload their generated diagnosis keys or encounter tokens to the server. For this purpose, the paper [AFV21b] devises a number of schemes allowing unscrupulous users to sell their infection status for profit. These mechanics are not part of the core tracing protocol, rather falling under the responsibility of implementors. Thus, we classify this attack vector as an implementation detail. One countermeasure, at least making it more difficult for users to sell their positive test result, would be to tightly couple the release of the test result to the diagnosis key/encounter token upload. The inconvenience caused by this mitigation might outweigh its benefits, though.

6.1.3. Time Modification Attacks

Having analyzed attack vectors mainly targeting design flaws of the tracing app and protocol itself, we now focus on a security issue highlighted in [IVV21a], namely the possibility to influence the device time remotely. As time is an essential factor used in the tracing protocols at hand, modifying the time disrupts certain processes in tracing protocols and is particularly hard to detect.

Keeping with the theme of the last section, we first look at an attack allowing to inflate the risk score generated by a fake encounter. This attack, dubbed “Master of Time”[IVV21a], advances the device clock between two scans of an encounter. As both systems rely on the device time to measure the length of encounters, they are both vulnerable. The only way to prevent this attack is to use a timer which always advances independently of the operating system clock. This can either be a software timer, which consumes a whole CPU core and wastes power, or a hardware timer, which might not be accessible on mobile phones.

Two attacks used for injection of keys make use of the limited validity period of the ephemeral IDs used in tracing. Both attacks first modify the device time to a point in this validity period and then generate an encounter with this ephemeral ID. In GAEN, this would allow for a “belated” replay attack (after the ephemeral IDs were rotated) [IVV21a]. In TraceCORONA, where replay attacks are not possible, the risk score could be affected or daily encounter limits bypassed by an attacker. The second attack combines this flaw with a bug in the servers used by multiple countries, where Temporary Exposure Key (TEK) that were still valid could be downloaded as part of diagnosis key packages. This “KISS attack” [IVV21a] allows attackers to inject encounters without having the ability to upload encounters. It is a bug in the implementation and not relevant to TraceCORONA, as private keys are not transmitted to the server.

Two other attacks, dubbed “My-Number attack” [IVV21a] allow for exfiltration of past or future ephemeral IDs from the device. By setting the device time and waiting for the ephemeral ID to change, the device broadcasts an ID not meant for the current time frame, rather broadcasting one already used or scheduled to be used. This attack works on GAEN, as described in the paper, and can also be applied to TraceCORONA if keypairs and ephemeral IDs are generated in advance. To prevent this, the system can record if one of these records was used and never use an ephemeral ID twice. As it is not cryptographically bound to the time as with GAEN, this poses no problem.

Another attack is to set the device time to the far future, which generates an ephemeral ID scheduled to change at the end of its validity period. This period can last over 100 years, defeating the “ephemeral” part completely. If successful, the functionality and privacy of the app are broken completely. However, this attack relies on a logic flaw in the GAEN system and can be easily fixed.

6.1.4. Status Reveal Attacks, Digital Evidence

Finally, after discussing the impact of several issues where prevention is mostly possible is some way, we now enter the realm of attacks inherent to existing contact tracing flows. All of these attacks are possible on both discussed systems and can only be mitigated, leading to limitations in the platform.

The first class of attacks inherent to (decentralized) contact tracing are status reveal attacks. These allow the attacker to link certain personally identifiable information with the information that a person is infected. One attack, theorized as a “Paparazzi attack” [Vau20], not to be confused with the one above, and demonstrated in [NAE+21], uses a camera to capture a photo of a person and establishes an encounter at the same time. If the encounter token or ephemeral ID is later confirmed as belonging to an infected person, the attacker can link the photo with it. An approach to mitigate this attack was discussed in [PF20], in which random people are notified as being infected. This mitigation was originally devised for a centralized tracing system. Transferring it to a decentralized system poses some challenges, as people need to be compelled to share their encounter data for a seemingly arbitrary reason.

Another status reveal attack, dubbed the “Nerd attack” or “Militia attack” [Vau20], proposes the following scenario: A custom client for the contact tracing system is created that allows people to manually annotate their encounters with additional information about the encountered person. Later, when the risk score is calculated, the system can tell whom the risk came from. If this information is shared between users, it could even be used for suppression of infected people. To mitigate this attack, devices must prevent the access to contact tracing broadcasts to unauthorized apps. This needs to be enforced on the operating system level and would make the development of new BLE applications more difficult. Additionally, a jailbroken or rooted phone could bypass these blocklists, enabling dedicated users to still perform the attack.

The paper [DR20] proposes one more status reveal attack named “biosurveillance attack”: this time, unsuspecting people install a trojan horse software including a tracing component which does not display its results to the user. Rather, a risk score is then calculated for the attacker to know if their victim is at risk of infection. Again, this issue can only be fixed on the platform level, e.g., by the app store providers (cf. Section 5.3.3). The mitigation described in the last paragraph also applies for this attack, with all its flaws.

After analyzing all of the attacks that we consider as real attacks, we want to remark on the collection of *digital evidence* of encounters. Users can use a well-known digital evidence technique such as publishing a hash on social media or using a blockchain to store their encounters. Later, they can provide compelling evidence that their encounter was genuine and not faked after the fact. This scheme is possible on all decentralized tracing protocols, as all of them record a certain kind of evidence of encounter as part of their functionality. Therefore, the only “fix” for this property is to use a centralized tracing protocol instead. It is thematized by [Pie20] and called a “Gossip attack” in [ABIV21]. There, it is discussed that this could actually be a feature to prove the authenticity of a high risk warning, e.g., if it would lead to a free test and therefore an attacker would be motivated to fake a high risk status.

6.1.5. Discussion of Attack Risk

With all of the discussed attacks, there are different ways to judge their risk, impact or relevance when comparing different systems. In this section, we discuss different approaches towards summarizing the results of our attack analysis, and why we chose not to.

The industry standard way to rate vulnerabilities in software is the Common Vulnerability Scoring System (CVSS) [FIR19], in use in the Common Vulnerability Enumeration system, which two of our evaluated attacks are actually a part of. For our study, we argue that the system (CVSS v3.1 Base Score) is too coarse-grained in certain areas, while including metrics difficult to map to contact tracing systems. For example, both attacker models including Bluetooth LE sniffers and transceivers would fall under the category of an adjacent attacker with no privileges. How do we classify user interaction in these attacks? Is walking by the attacker a form of user interaction? Another attribute in the Base Score is Scope. This attribute is designed for software and hardware inside a device, but is hard to apply to a distributed system such as contact tracing. Overall, we believe the CVSS is not a good fit for our security rating.

Another system in use to classify the importance of security issues is the DREAD model [HL02]. It consists of five metrics, which are averaged together in the end. This rating is more fine-grained than the CVSS, while at the same time having a flawed calculation, which the author proposes to fix by applying rules more akin to CVSS [LeB07].

[HL02] proposes another, simpler model titled Risk$_{\mathrm{CO}}$, in which there are only two metrics: Criticality and Likelyhood of Occurrence, which are then multiplied to yield the final score. Criticality describes the potential damage of an attack, while the likelyhood of occurrence describes how likely the attack is to affect any user.

The main issue with all of these metrics is their subjective nature. CVSS comes the closest to an objective rating, while just estimating a numerical risk (or “high, average, low”) is very subjective. Different actors have reason to estimate attacks in a different light, and users may also differ in the rating of these attacks. Thus, we decided not to give a rating to the attacks, rather leaving this estimation to the reader.

Therefore, we can only conclude that TraceCORONA is not vulnerable by design to any more attacks than GAEN. In concrete numbers, GAEN is affected at least partially by 14 of the attacks listed in Table 6.1, while for TraceCORONA this number is 8.

6.2. Formal Analysis

After the previous section has established the applicability of known attacks, we now focus on the discovery of new attacks against the systems. For this purpose, we attempt to use the technique of formal analysis. Section 6.2.1 reviews our choice of tool and Section 6.2.2 discusses the results we are able to gather from the analysis.

6.2.1. Verifpal

In the field of formal analysis, a range of tools allow to verify a given network protocol's security and privacy properties by simulating attacks which are possible to model using the given tool's modelling language and logic. For comparatively analyzing the security of TraceCORONA and GAEN, we use the Verifpal [KNT20] tool. This tool, a relatively recent development in the field, focuses on being understandable rather than allowing to model every possible property of a protocol. As formal analysis is only a relatively small part of our work, we assume that the time needed to build basic proficiency with more advanced modeling tools was better spent understanding and comparing the protocols in depth.

The manual [Kob21] describes all features of the software intuitively. As queries, Verifpal can prove confidentiality of values, authentication of messages, freshness of cryptographic keys, unlinkability of IDs and equivalence of values. The latter two are relevant to our analysis, as unlinkability of identifiers is a concept especially interesting for contact tracing and ProVerif [Bla01], an older, more established tool, does not seem to support checking for equivalence of values.

Some limitations were encountered when modeling the protocol. In a Verifpal model, messages arrive in the same order every run, with checked primitives being the only way to model a deviation from the standard control flow by aborting the simulated flow. Additionally, there are only two attacker models to be checked: either the attacker is completely passive and can only observe communicated values, or the attacker follows the so-called

Dolev-Yao active attacker model and can modify, replay, withhold or generate any message. The former is more relaxed in ProVerif, as multiple processes run in parallel, while the latter property is identical.

Modeling all properties of a Transport Layer Security (TLS) connection, while minimizing the model size, which is required for execution to finish in a reasonable time, is not trivial. One model even crashed the Verifpal software after running for over two days on a computer with an Intel Core i7-5820K processor. In the full model, as in the real world, every connection begins with a certificate check, then a session key is established, which is later used for communication. A simplified version without establishment of a fresh session key for every connection allows to prove properties impossible to verify with the original model. However, here the server communication is vulnerable to replay attacks. Finding the right balance between modeling too much and having the simulation not finish, or modeling too little and getting false positives, is the main difficulty in the analysis process.

A modeling tool with extended features can aid in analysis of complex models such as the ones described in this section. A predefined notion of TLS communication, a channel where confidentiality, integrity, authentication, freshness, and perfect forward secrecy are ensured without having to use the authenticated encryption primitives and exchange keys manually allows to focus on the contents of the transmission rather than the structure. Additionally, a feature present in ProVerif, but completely absent from Verifpal by design are data types, which, when used for simulation, reduce the attacker's guesswork. In our largest model, there are 96 constants, which despite optimizations result in a large amount of useless deductions and unnecessary substitutions in results.

Apart from the features, one bug was encountered and reported to the developers. Initially, with version 0.26.0 of Verifpal, Diffie-Hellman key exchanges, which we used to approximate the ECDH key exchange in TraceCORONA, were broken completely and did not yield an equivalent value on both sides. This bug is fixed after our report in the current version 0.26.1, which is used for the remaining experiments.

We also describe some unexplained behaviors, for which it is not certain if they occur due to bugs or design limitations. For one, when a private value is first used for some operation, then afterwards publicly transmitted, the attacker can change the value and it will be considered changed even for the operation which appeared first. Another unexplained behavior is a failed authentication query despite all of the secrets and keys being fresh. In the process of testing the latter issue, an equivalent freshness query takes over one month to finish, which exceeded our time for this thesis.

Overall, Verifpal manages to provide an easy introduction into protocol verification. However, despite its advanced capabilities and a provided example model of low-cost DP-3T (described in Section 3.1.3), we are not able to prove full models of the protocols at hand.

6.2.2. Models and Results

In total, we build 3 models of our protocols leading to different results each. The following paragraphs therefore describe one model each. Models and queries are listed in full in Section A and the queries are enumerated the same as in the listing in this section.

`tracecorona.vp`

For TraceCORONA, the flow modeled here begins, after key generation, with `UserI` and `UserJ` encountering each other. `UserJ` initiates the handshake using Diffie-Hellman, which is functionally identical to ECDH. `UserI` tests positive and receives a TAN from the health authority. They then acquire a nonce and upload their encounter token hash generated earlier, as well as the StateAndNonce packet, to the server. `UserJ` downloads the encounter token hash and checks if it is identical to the recorded one (matching process), then proceeds to decrypt the state and nonce.

This model, as our first attempt at comprehensively modeling the protocol, includes a certificate authority and health authority, as well as separate ephemeral keys per connection. A full TLS connection is established for every modeled connection over the internet. The queries tested with this model include:

1. The confidentiality of the encounter token, which never leaves the device. Diffie-Hellman is vulnerable to man-in-the-middle attacks, and so Verifpal managed to disprove this hypothesis by inserting a bogus value into the key exchange. Especially this result made us discuss the Sybil attack in detail.
2. Another query we test with this model is the equivalence of the encounter tokens for both users. This is disproven the same way.
3. We try verifying the authentication of the upload message, i.e., if the attacker could modify or replay the upload message. This should be impossible due to the nonce being only transmitted over TLS. However, the verification process was not finished after one week on a computer with an Intel Core i7-8665U processor and we therefore gave up on this path.

`tracecorona-3people-simple.vp`

There are two problems with the previous model. If we guard the key exchange, i.e., mark it as read-only by the attacker, there is no way left for the attacker to interfere with the protocol. Also, other queries take too long to execute. Therefore, we develop a second model, in which there are three participants. `UserK`, our new participant, leaks all of their secret values to the attacker. To replace the TLS key exchange every connection, we pre-share one symmetric key each between server and users.

1. Using this model, we verify the confidentiality and equivalence of the encounter token between users I and J, which now exchange keys with a guarded connection.
2. Also, the private keys of both users are never leaked, which would allow the attacker to impersonate each user.
3. Again, the attacker is able to modify the exchange between J and K, as this connection is unguarded.
4. When testing authentication of the upload message, as expected the attacker is able to replay a previous message due to missing freshness guarantees.
5. Furthermore, the model verifies the confidentiality of the infection state and nonce value, which is transmitted end-to-end encrypted and should therefore not be readable for the attacker.

`coronawarn.vp`

For the Corona-Warn-App, the flow is similar to the 3-person variant of TraceCORONA. Instead of exchanging encounter tokens, Rolling Proximity Identifiers (RPIs) and AEM packets are sent to the other user. In contrast to the other models, we utilize a single guarded Diffie-Hellman key exchange to establish the session keys for network transmissions at the beginning of the model. Due to the properties of GAEN, several queries yield a different result.

1. The AEM key is not confidential after upload of the TEK. Inspired by this result, we further investigate the metadata transmitted in GAEN in Section 6.3.
2. We check that the AEM contents are the same if the attacker cannot modify the Bluetooth exchange. This should be the case, but an authenticated and encrypted value using a fresh key is still able to be modified by the attacker. This is another unexplained result.
3. Again, the authentication of the upload message is disproven. We believe this result to be a false positive, possibly due to a bug in the program.
4. Therefore, we instead query for the freshness of the generated secrets. At the same time, we try to prove that the attacker can link RPI from the same day. These queries were not finished after over one month of runtime on a computer with an Intel Core i7-2680QM processor.

To summarize, the results of the formal analysis with Verifpal are limited, in part due to the issues described above. However, even these analysis results lead to a further investigation of two attack vectors present in the protocols. These are described in the following two sections.

6.3. Metadata Attacks

In addition to the information of who was encountered, tracing systems have the ability to perform filtering of encounters based on signal strength measurements. For this purpose, Google maintains a database of calibration data for Bluetooth radios of different devices [Goo20f].

The appropriate values for transmission power and calibration confidence are included in GAEN advertising packets as the AEM [Goo20a]. Encryption is handled with an AEM key, derived in the same way as the RPI key, which is first used to encrypt the currently active RPI and then XORed with the AEM value. This behavior corresponds to the CTR mode of operation, but without actually counting up, as the size of the AEM is 4 bytes, smaller than the 16 byte block size of the used AES-128 cipher. No padding is used.

This encryption prevents a hypothetical attack in which users' phone models could be identified by comparing the transmitted values to the database. It is only possible for users diagnosed positive after diagnosis key upload [Mar20]. However, not using an authenticated encryption leaves the system vulnerable to another attack: metadata can be manipulated in a relay or replay attack scenario to influence distance measurements and therefore the risk score presented to other users [VV20; Mar20]. Google refused to switch to an authenticated mode of operation, as relay attackers would be able to falsify measurements anyway by using a high-powered antenna, and therefore a further amplification of this effect seems to have low impact in their point of view. Additionally, they believe that relay attacks are a problem of malware to be solved through app store policies, neglecting the fact that malicious parties may use their own Bluetooth radios [Mar20].

In TraceCORONA, the advertisement packet includes the transmission power level of the sender as reported by the Android Bluetooth Application Programming Interface (API). At the time of development, there was no database of Bluetooth calibration data available, making distance measurements unreliable at best. However, this also prevents a similar hypothetical vulnerability in TraceCORONA, where this fixed calibration data could be used for deanonymization. A future version of the protocol designed to utilize the calibration data should instead use an encrypted variable akin to the infection state and nonce to transmit this information.

6.4. Encounter Injection and Sybil Attacks

When analyzing the communication between devices in TraceCORONA, Verifpal alerted us that the Diffie-Hellman handshake was vulnerable to a man-in-the-middle attack. This inspired us to devise a scheme for simple encounter injection and encounter injection in combination with Sybil attacks[2]. After describing the basic idea in Section 6.4.1, we first describe our assumptions in Section 6.4.2 and then analyze the impact of Sybil attacks in detail in Section 6.4.3. Lastly, we discuss possible countermeasures in Section 6.4.4.

6.4.1. Idea

Bluetooth LE-enabled mobile devices often support a mode called "multiple advertisement", in which the device is able to act as if it were multiple distinct Bluetooth devices. This property can be used, for example in the "Mind the Security GAP" relay attack [BDF+20], to broadcast collected ephemeral IDs from other devices. For passive tracing protocols such as GAEN, this attack allows to fake a number of encounters, which can increase the risk level on the device.

An attack with a similar outcome, but different attack technique is possible on the TraceCORONA system. Instead of collecting ephemeral IDs, which is not possible by design on TraceCORONA due to the use of the ECDH key exchange, the attacker can generate multiple key pairs and therefore simulate many devices of their own. By encountering other users with this modified application running, the attacker is able to multiply the amount of encounters detected by other users.

However, merely generating encounters does not increase the risk score of an individual, rather these encounters have to be with a user who was later diagnosed with the virus. At this stage, the fact that token hashes are transmitted to the server individually, which is a design feature to reduce the traceability of users, becomes an issue. The attacker is able to send multiple encounter tokens per user to the server by just providing a single TAN, in turn multiplying the risk score generated by this single encounter.

In fact, this issue goes beyond a single device: if multiple devices share encounter tokens for upload, the reach of this attack is increased even further. Compared to the relay attack discussed above, however, the attacker needs to additionally obtain a valid TAN for encounter upload. Depending on the implementation of the system, methods as described in [AFV21b] may be utilized for this purpose.

Other types of tracing protocols are more resistant to this kind of attack. Tracing systems where broadcasted identifiers are uploaded can limit the amount of identifiers per upload to one per time frame. In practice, in CWA, upload messages are currently checked to contain a maximum of 14 keys [Cor21e], but not for multiple keys per time frame. Systems in which observed identifiers are uploaded can filter out duplicate keys.

6.4.2. Assumptions

After describing the basic idea and approach, we now explain a number of assumptions made in the following impact analysis and discussion.

Attacker Model

We assume the attacker possesses a legitimate TAN for upload of encounters/diagnosis keys. See Section 3.3.2 for related work in this domain. Furthermore, an attacker has full

[2]The term Sybil attack was coined in the paper of the same name [Dou02] referring to online peer-to-peer systems.

control over the device/devices used for the attack, these devices are able to communicate both ways with both victims and each other. This implies that an adversary can optimize all the relevant parameters dependent on the sender, which for GAEN is the signal strength and the transmission risk level. Physical or hardware limits are not considered for this attack.

Our analysis is based on the assumption that a real-world deployment of TraceCORONA would be utilizing the same risk scoring mechanism as GAEN, which is described in Section 5.3.2. If an attacker transmits with high power and manipulates the calibration metadata to fall into the highest attenuation bucket, which we assume to be more realistic than hitting the optimal target, they need to achieve a measured total encounter time of 7.03 min to trigger a high risk warning. We consider an attack as successful, i.e., a person as affected, if this warning is triggered.

Also, in GAEN a BLE scan is triggered every 3-5 minutes (cf. Section 5.4). The hypothetical deployment of TraceCORONA is assumed to also scan with this interval.

Encounter Limits

The design of TraceCORONA, in contrast to GAEN, allows for limiting the number of encounters able to be uploaded with a single nonce value. The amount of encounters allowed to be uploaded in the system is a tradeoff between functionality and security. If too little encounters can be uploaded, people at risk of infection may not be warned, but if too many encounters can be uploaded, the impact of encounter injection attacks such as the Sybil attacks rises. In this section, to be able to quantify the impact of the attack, we assume TraceCORONA implements a limit.

To find the correct number of encounters to limit to, there are several studies to be considered. Averages per day range from 9.9 contacts in a recent US study [Rot20] over 13.4 contacts in an European study [MHJ+08] to 16 average simulated encounters [DHHE07]. Notably, in [Rot20] retail employees averaged 89.4 encounters and manufacturing employees averaged 46.7 encounters. We consider two scenarios for limits of 160 encounters and 480 encounters over the whole 14 day tracing period, resulting in 11.4 and 34.3 average allowed daily encounters.

If the server could discern the day an encounter was generated on, the limit could be more granular. However, this would reveal more data – daily encounter numbers – to the server than is currently the case.

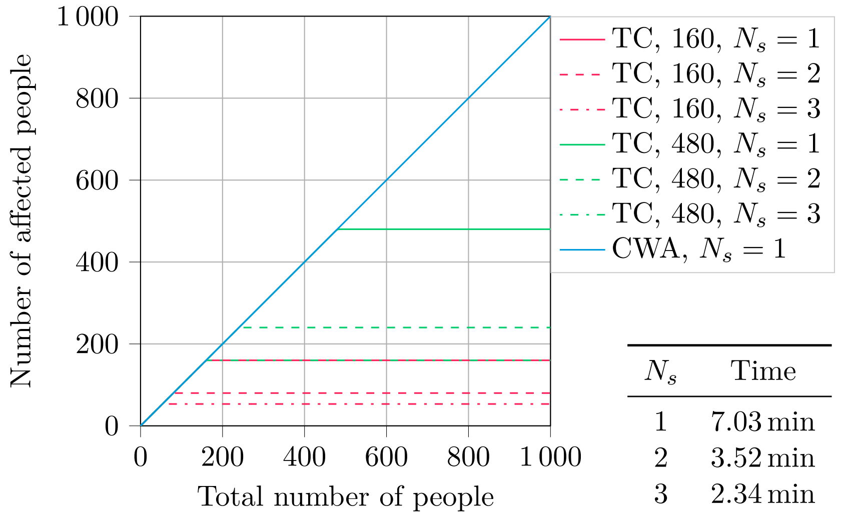
Attack Scenarios

To compare the impact of a Sybil attack between TraceCORONA and GAEN, we use two different scenarios:

- Mass scenario: A large number of people are present for longer than 9 min, e.g., a crowd of people or a busy lecture hall
- Targeted scenario: A small number of people are present for 3 to 9 min, e.g., a short bus ride or a smoke break

We set the cutoff at 9 min due to this being the minimum amount of time for 4 scans to occur on users' devices.

Note that in a benign mass scenario people present over an ephemeral ID change boundary would "consume" two encounter tokens. By switching off the tracing functionality before this change occurs, an attacker would be able to limit the count of encounters to one per person. In a "benign targeted" scenario, there is a chance of people not receiving a high risk warning even if one of the participants was later diagnosed positive.



N_s	Time
1	7.03 min
2	3.52 min
3	2.34 min

Figure 6.1 & Table 6.2: Impact analysis of encounter injection ($N_s = 1$, solid lines) and Sybil attack ($N_s > 1$, dashed and dash-dotted lines)

Applicability to GAEN

As mentioned above in Section 6.4.1, relay attacks on GAEN rely on the presence of infected users whose ephemeral IDs are captured. With TraceCORONA, all of the IDs need to be generated by the attacker themselves due to the usage of ECDH exchanges. This begs the question: Can an attacker use multiple IDs generated by themselves when performing an attack against GAEN?

On a conceptual level, this may seem impossible, as RPIs (ephemeral IDs in GAEN) include the validity time period as part of the key derivation process. In practice, however, RPI time periods are extended by 3 hours into the future and past to account for unsynchronized clocks on the device. In the August 2020 source code release of GAEN, we were not able to find any mechanism allowing to filter invalid RPIs if multiple RPIs in this grace period are detected at the same time. Confirmation of this possibility remains as future work, as this code is over a year old at this point and a detection mechanism may have been added in the mean time. Due to the theoretical possibility of detecting an attack without the possibility of restricting benign cases – by design, RPIs are linked through the diagnosis key – we assume Sybil attacks, i.e., simulating multiple devices with the same diagnosis key, not to be possible on GAEN in the following sections.

6.4.3. Impact Analysis

Now, we estimate the impact of the Sybil attack in the two scenarios highlighted above.

Mass Scenario

Figure 6.1 shows the potential impact of encounter injection and Sybil attacks. N_s is the amount of sybils: the solid lines represent a simple encounter injection attack, with the attacker only simulating a single device ($N_s = 1$), while the dashed and dash-dotted lines represent a Sybil attack with the attacker simulating two and three devices, respectively. The amount of affected people in Corona-Warn-App (CWA) with an encounter injection attack is unlimited, hence the graph shows an identity relation between total and affected. With TraceCORONA (TC), the amount of affected people is limited by the encounter limit, which is why after an identical impact for lower counts there is no further increase after the limit is reached.

Actually conducting a Sybil attack in the mass scenario does not make sense for the attacker, provided that an encounter limit is present. It only divides the amount of affected people by the amount of sybils used, i.e., by half with $N_s = 2$.

Targeted Scenario

Where the Sybil attack actually makes sense is the targeted scenario. The time required to trigger a high risk warning is shown in Table 6.2. With the attack active with $N_s = 3$, being recognized in two scans is guaranteed to lead to a high risk encounter, as the time required to trigger a high risk warning is lower than the minimum time between scans ($2.34\,\text{min} < 3\,\text{min}$). $N_s = 2$ represents a middle ground, where the attacker is more likely, but not guaranteed to achieve their goal in two scans and guaranteed in three, while without the attack ($N_s = 1$) three or even four scans are always needed.

As we assume only the simple encounter injection attack to be possible on GAEN, only $N_s = 1$ applies for this system: three to four scans are needed regardless of the attacker's behavior. Therefore, the Sybil attack does not have any effect on GAEN in this scenario.

However, this impact analysis shows the limited use case of the Sybil attack in TraceCORONA with encounter limits. If there are no encounter limits, the limitations of affected people and brevity of the encounter are related to hardware, i.e., how many devices a single Bluetooth sender is able to simulate in the given timeframe, and timing, i.e., how many of these devices are actually detected by users. Still, being able to affect a targeted group of people in a short time only soliciting a single TAN has the potential to cause disruption, for example if parliamentarians are prevented from joining a vote.

6.4.4. Countermeasures

After we established the attack impact of the simple encounter injection and encounter injection with Sybil attacks, we now analyze three approaches to prevent or mitigate the attack. All of the following countermeasures can be described as tradeoffs between security and effectiveness. Care must be taken to set the parameters in a way as to avoid preventing proper functioning of the tracing protocol in a benign scenario.

As analyzed above, the ability to limit encounter tokens per upload is the main countermeasure to be employed by TraceCORONA against encounter injection attacks by single individuals. To minimize the effect of the limit on tracing functionality, the app may sort encounter tokens by infection risk and upload the encounters believed to be most risky for the exposed person. This in turn allows the Sybil attack to be used for denial of service – preventing infected people from sharing their encounters – instead of encounter injection. For this outcome to occur, by using a large amount of sybils and a strong Bluetooth signal, victims' devices are forced to record many spoofed high-risk encounters, leading them to report less real encounters and hindering functionality.

Before March 2021, Corona-Warn only counted encounters with a normalizedTime of at least 5 minutes, and before February, only encounters with at least 10 minutes, with a total time to be reached of 15 minutes [Cor21b, old versions]. If we assume these minimum encounter times, the Sybil attack with over two sybils is not even effective in a targeted scenario. Due to the emergence of newer virus variants with a much higher infectiousness, shorter encounters are counted nowadays.

A different approach for mitigation of Sybil attacks is the heuristic detection of these scenarios. The following can be observed by a device under Sybil attack: (1) multiple devices with (2) a low attenuation and (3) a weak antenna ("close by") are encountered (4) simultaneously for (5) a short period of time. After detection of a scenario of this kind, either all encounters from this time period are ignored for risk detection or the user is warned of a potential disruption. On the other hand, a benign scenario triggering this behavior could be a short bus ride with multiple people later being tested positive and uploading their contacts at similar times. These scenarios might neither be discernible to the user nor the app, leading to false negatives if detected by the heuristic.

Alternatively, instead of heuristic detection, the risk score calculation may be altered to count all simultaneous encounters as a single encounter. Intuitively, when more than one infected person is present, the infection risk is higher due to more virus particles in the air. However, we were not able to find any study correlating this observation to a faster transmission of the virus.

In conclusion, the simple encounter injection attack is an inherent flaw of anonymous automated contact tracing, for which TraceCORONA allows to limit the number of affected users, while GAEN allows an unlimited amount of users in the attacker's range to establish encounters. When combined with the Sybil attack, the impact is increased under certain scenarios, with the limits imposed by TraceCORONA leaving the attacker to a tradeoff between affecting more people and affecting less people quicker. In GAEN, as well as the systems using it, checks for duplicate identifiers can be added to eliminate the possibility of a Sybil attack completely, while we are unsure if these checks are implemented at the current time.

7. Traffic Measurements

After the protocols of Google/Apple Exposure Notification(s) (GAEN) and TraceCORONA have now been analyzed theoretically in detail, we make measurements on the traffic generated by transmissions to and from the server. The results serve as a point of comparison between the two applications as well as backing up our analysis from Section 5.2.

The goal for our traffic measurements is to analyze the download process or matching process, which is performed on a regular basis by all users, as well as the upload process, performed only by users who have tested positive. We compare single and multiple (2 or 5) device systems to prove that traffic volumes can be interpolated for large amounts of devices. All numbers resulting from measurements in this chapter are averages of at least 3 benchmark runs.

Download and upload processes should be measured separately to allow for better comparison. To achieve this, our general flow for traffic testing is to first generate a certain amount of cryptographic identifiers (tokens or keys), then identify as positive and upload these tokens to the server, and in the second phase download the identifiers again. All of these steps are done with the same app instance, which is possible as both tested systems do not require any authentication or discern between clients when downloading keys to be matched. Before repeating the benchmark with a different number of encounters, the server side and the client side are reset by deleting the data, and the setup process is done again.

Section 7.1 describes the tools used for testing, followed by three sections describing each of the measured systems. Afterwards, in Section 7.5, we compare the results of Corona Contact Tracing Germany (CCTG) and TraceCORONA, before we verify the results of Section 5.2 and estimate the feasibility of a large-scale deployment of TraceCORONA in Section 7.6.

7.1. Instrumentation

To simulate each tracing system, we run the respective server side using the Docker Engine (version 20.10.8)[Doc21], which is used by both projects for easy deployment of applications, on a Linux computer. The server versions used are 2.5.0 for Corona-Warn-App (CWA) and 1.1.9 for TraceCORONA. To evaluate both apps fairly, we aim to recreate the connection parameters used in the infrastructure of the CWA, namely `https://svc90.main.px.t-online.de/`, `https://submission.coronawarn.app/`, and `https://verification.coronawarn.app/`. All servers use HTTP/1.1 exclusively, while the server behind the former domain offers TLSv1.2 only. The submission and verification servers are reachable with TLSv1.3 as well. nginx (version 1.21.3) is used [NGI21] as a reverse proxy for both TraceCORONA and CWA benchmarks. However, in our main measurements, the TLSv1.2 cipher suite `TLS_RSA_WITH_AES_256_GCM_SHA384` is used, as it does not provide perfect forward secrecy and allows us to decrypt the communication afterwards for analysis purposes. After the session key is established, it functions identically and as such the performance should be comparable to the original cipher suite.

When running benchmarks, all communication is captured on the server side using Tcpdump (version 4.99.1) [The21b]. We filter by the IP address of the host to avoid capturing server-internal communication. For sanity-checking the data, Wireshark (version 3.4.8) [Wir21] is used. This utility allows for usage of the RSA private key of the server to decrypt the TLS connection.

Depending on the test, the client side is run either on a number of emulated Google Pixel 4a devices (Operating System (OS) version RSR1.201211.001.A1) or a real Google Pixel 3a XL (OS version RQ2A.210505.002). All devices are running Android 11, with the operating system distributed by the manufacturer.

For interfacing with the Android devices and automating the benchmarks, we use Appium (version 1.21.0) [JS 21], which allows remote control of an Android device using a WebDriver-based [SB18] interface. In addition to having debugging access enabled, the devices have Magisk (version 23.0) [Wu21] installed for full access to application data without modifying applications, easy modification of the `/etc/hosts` file to forego running a DNS server, as well as allowing instrumentation of the GAEN framework.

On the computer side, the Appium server is interfaced by JavaScript code running in Node.js (version 16.6.2) [Ope21a] using the WebdriverIO (version 7.7.4) [Ope21b] library. We chose this language to minimize dependencies, as Appium also runs in Node.js. Additionally, the xml2js (version 0.4.23) [Kub21] and better-sqlite3 (version 7.4.1) [Wis21] libraries are used for parsing and modification of application files.

The benchmarks are orchestrated by a central script calling the necessary utilities (Docker, Tcpdump and Node.js), allowing for easy repeatibility. Single benchmark steps are separated into multiple Node.js programs to allow for reuse between benchmarks or even benchmarked applications (CCTG and CWA). Several functions are implemented in a separate library file, `helpers.js`, to further enhance the readability. As Appium provides a unified interface to the Android Debug Bridge (ADB) and UI Automator tools, we use it whenever possible.

Tcpdump is configured to output the captured packets into a pcap format file. We then separate the TCP/TLS connections (streams) using the Tshark utility included with Wireshark. Our measurement for the generated traffic volume is the sum of all Ethernet frame sizes of the relevant HTTPS connections recorded during the benchmark run. This extra step filters noise created by connections not essential to the measured communication, in our case CWA/CCTG trying to download extra diagnosis key files.

For the CWA backend, the demo mode of the distribution service is activated to disable the shifting algorithm (described in 2.2.4) which would prevent diagnosis key packets from being generated.

7.2. TraceCORONA

As TraceCORONA generates tokens only when actively encountering a device, the application database is downloaded after the initial keypair generation and fake encounter

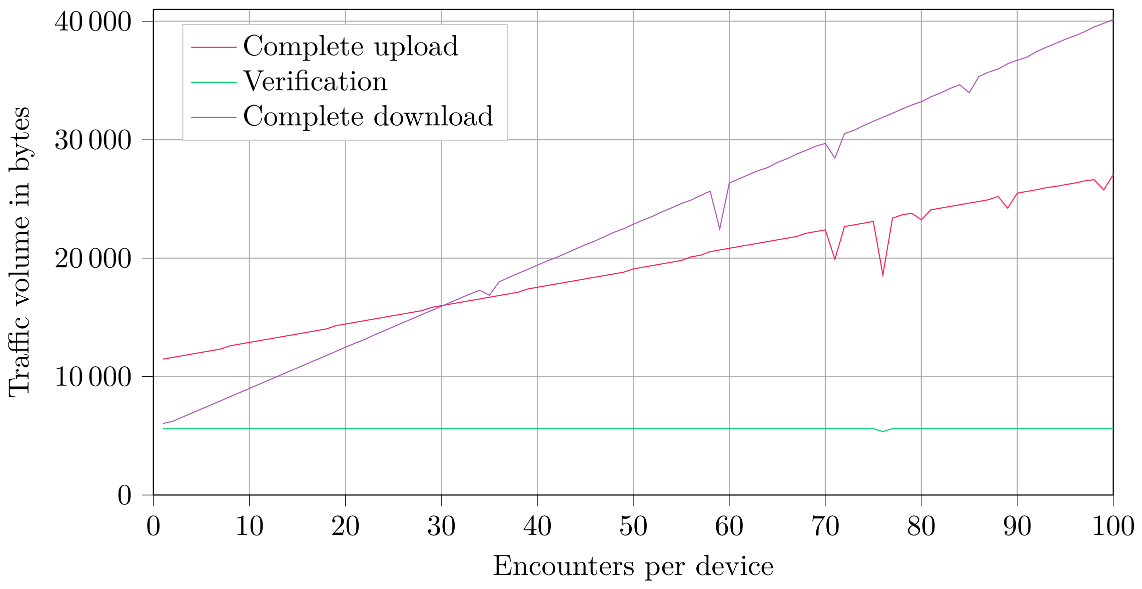


Figure 7.1.: TraceCORONA traffic volume per device, 1 device

values are inserted into the `EncounterTokenParameters` and `Encounter` tables. The data are converted by the application to encounter tokens before upload. This way, an arbitrary amount of encounters can be generated without emulating a Bluetooth device or programming an app for encounter simulation. Therefore, it allows us to perform tests on an emulator, eliminating device hardware as a factor.

TraceCORONA only initiates the matching process if prompted to do so by the user, i.e. when the “Check Your Status” button is pressed. This is a deliberate design decision to allow the user to decide for themselves when they want to view their status, instead of being displayed on the main screen as with other tracing apps. For our benchmark, this proves convenient as no spurious connections are measured in a typical benchmark run.

When looking at the results of the benchmark, plotted in Figure 7.1 and Figure 7.2, we are still able to observe discrepancies in the upload and download of tokens. Due to the status of TraceCORONA being a prototype application not developed for real-world usage, we did not further explore the reasons for these errors and instead assume these to be either caused by bugs in the application or disturbances in the benchmark. We were not able to benchmark more than 2 devices, as the errors became more serious, lending credibility to the theory that these are caused by load on the testing computer.

Leaving the errors out of the picture, though, the results show a near-linear increase in traffic volume with an increasing amount of encounters per device. For uploads, approximately every 10 encounters a small amount of overhead is added due to the data being split into one more TCP packet. As expected, verification requires a constant amount of bytes as well.

7.3. Corona Contact Tracing Germany

The other system in our benchmark, CWA, proves more difficult to instrument, as the application uses the GAEN Application Programming Interface (API), which is implemented by the closed-source Google Play Services and restricted by application signature checks. The en-13n script [BH20] was created to bypass this check. Unfortunately, this does not succeed on the recent version of Google Play Services required for current versions of the

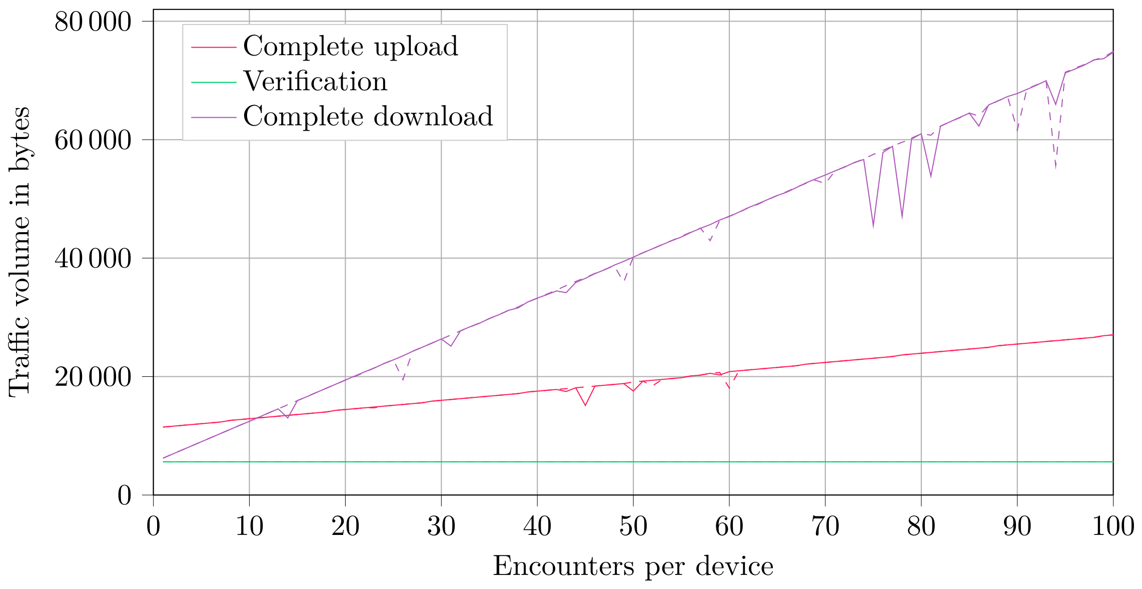


Figure 7.2.: TraceCORONA traffic volume per device, 2 devices. The straight and dotted lines each represent one device

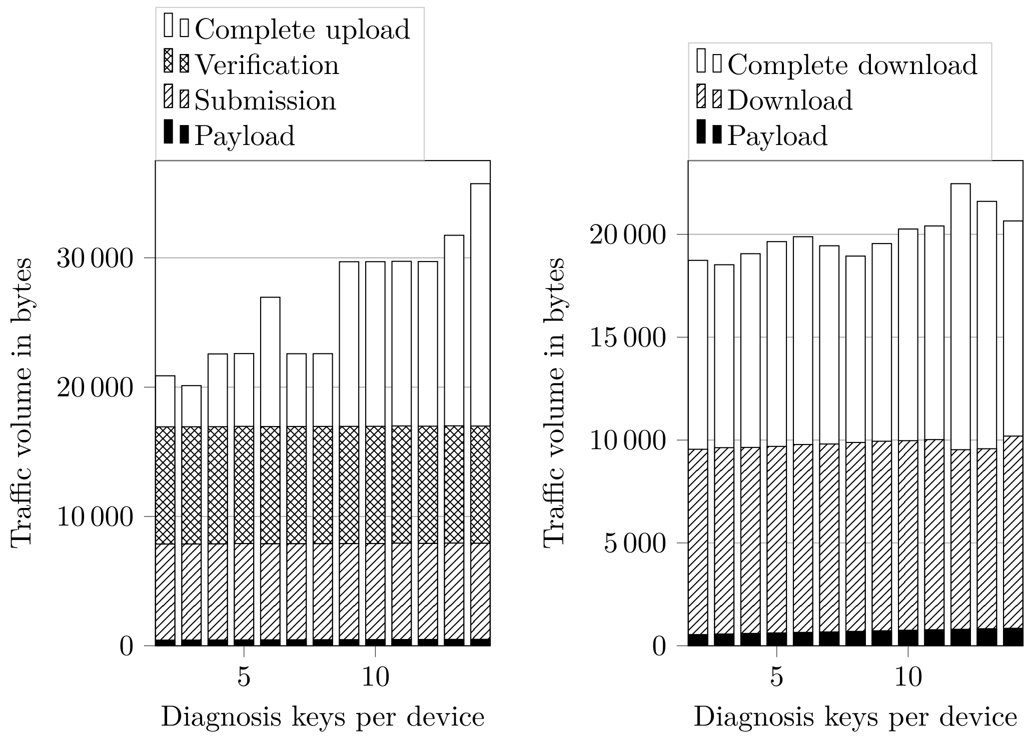
application. As a workaround, we therefore use the CCTG [con21] application (version 2.5.0.1) instead for this benchmark, which stems from the same code base, but contains an open-source reimplementation of the GAEN API. To ensure interoperability with Apple devices, the functionality of the API is fully specified. Therefore, traffic measurements should not be different, as the networking code is identical to the original CWA Android application. The tests for CCTG were performed on emulated devices.

Before starting the app for the first time, the date is set $n_{\text{keys}} - 2$ days into the past, where n_{keys} is the amount of Temporary Exposure Key (TEK), which later become diagnosis keys, to be generated in this benchmark run. For all the phones in the benchmark, the first-run tutorial is confirmed and the necessary permissions for Bluetooth Low Energy (BLE or Bluetooth LE) are granted, which is required for enabling the contact tracing functionality and generating TEKs.

Note that the minimum amount of diagnosis keys is 2, as one extra key is generated for an unknown reason. The maximum amount is 14, as the server will not accept more than 14 keys.

Once the app is started, tracing is turned off, the date is advanced by one day and tracing is turned back on. This procedure is repeated until the required amount of TEKs is generated. After generating the TEKs, the keys are uploaded. For this purpose, the teleTransaction Authentication Number (TAN) workflow is used: a 10-character TAN is entered (`SSS-WUE-TANG`) and the default profile of Transmission Risk Levels (TRLs) is selected, i.e., no Days Since Onset of Symptoms (DSOS) value is provided. A few seconds later, when the upload is finished, the test result is deleted to again enable the option to initiate the download and matching process. For this matching process to begin, the main activity of the app has to be reloaded. This is achieved by closing the app and relaunching it.

This process is repeated for all devices to be benchmarked. After the benchmark run is finished, all diagnosis keys are deleted from the server-side database and the object storage server is cleared. Then, the distribution service is run once, recreating the app configuration files to avoid errors.



(a) Upload process (b) Download/matching process

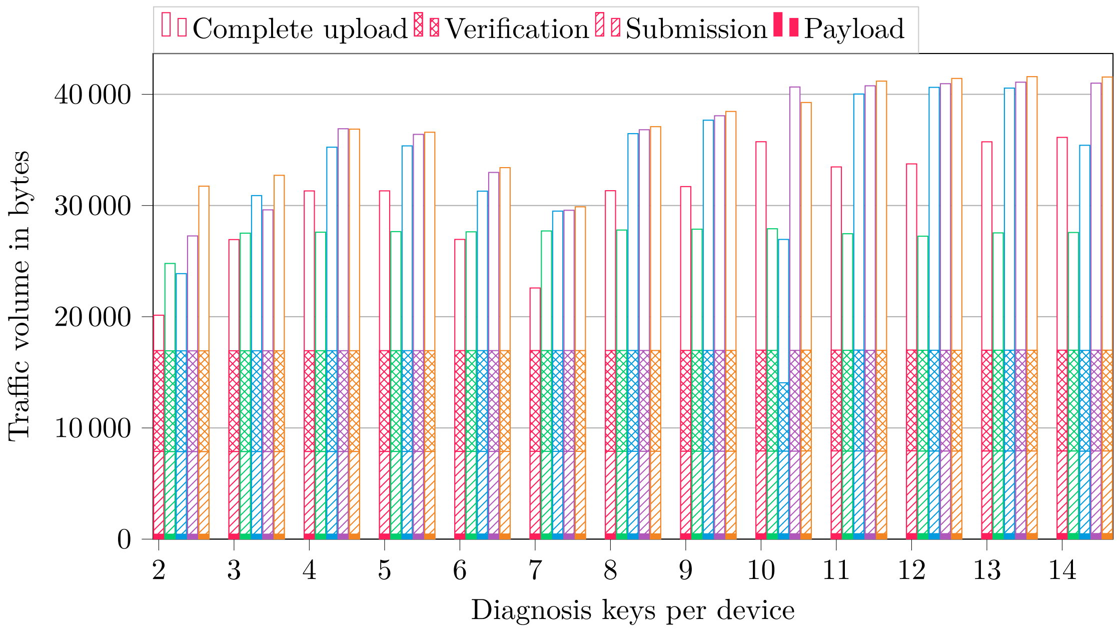
Figure 7.3.: Corona Contact Tracing Germany traffic volume per device, 1 device

When performing this test without changing any settings on the device, the download and matching process is initiated at seemingly random times. This is due to the server not providing any up-to-date diagnosis key packages, forcing the synchronization algorithm in the app to automatically retry for new packages again and again. To minimize the impact of these queries on our benchmark results and force devices to download only the current diagnosis key package, we turn on Airplane Mode at all times when internet access would lead to erratic measurements, i.e., when Tcpdump is capturing packets and the key upload process has not begun yet.

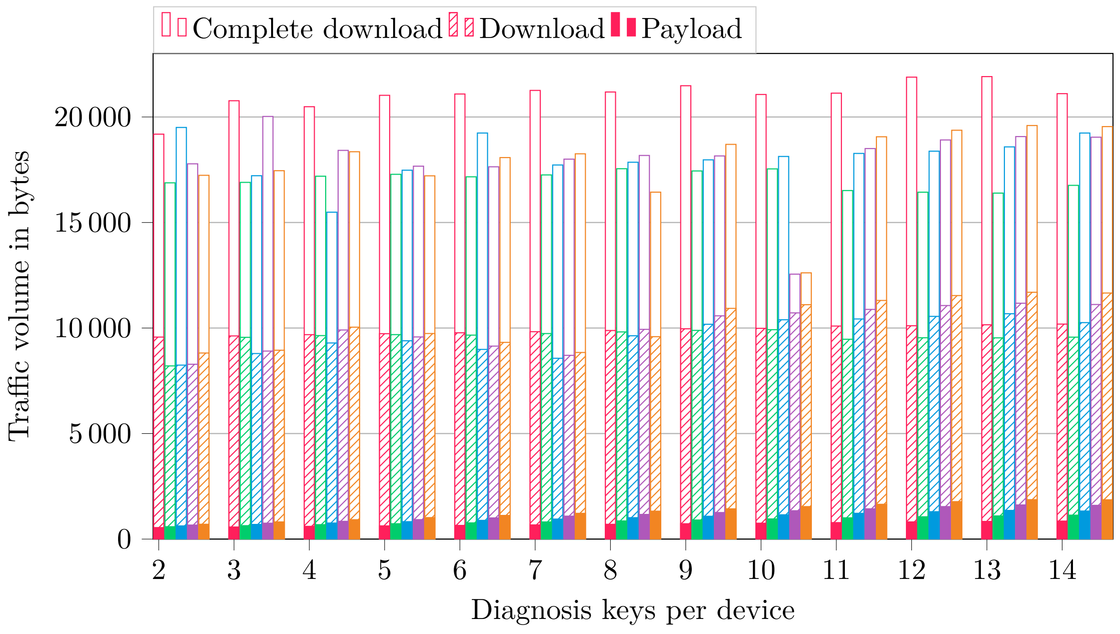
Figure 7.3 and Figure 7.4 show the raw measurement results, which is the traffic volume per device as measured on the server side. The solid part of the submission traffic bar is the size of the actual submission payload containing the diagnosis keys. Similarly, the solid part of the download traffic bar is the size of the diagnosis key package that is generated. Note, however, that this package is downloaded separately and not extracted from the captured packets in order to provide more reliable results. For some test results, the download still occurred while measuring the upload mechanism and was counted as part of the complete upload, so the *complete download traffic* does not necessarily correspond with the *download traffic*.

In general, the complete upload and download numbers do not exhibit any meaningful properties. Rather, the individual streams show that the verification process, as with TraceCORONA, always generates the same amount of traffic. The download overhead is constant in the 1-device test, but varies wildly in the 5-device test. This may be due to the previously-mentioned random download attempts, however we have not further analyzed these discrepancies, as the 1-device test numbers can be interpolated to multiple devices.

The submission process generates almost the same amount of traffic regardless of the number of diagnosis keys. In Section 5.2.2, we discovered a bug that prevents the generated traffic from being exactly the same every time. When isolating the upload payloads,



(a) Upload process



(b) Download/matching process

Figure 7.4.: Corona Contact Tracing Germany traffic volume per device, 5 devices – colors correspond to devices

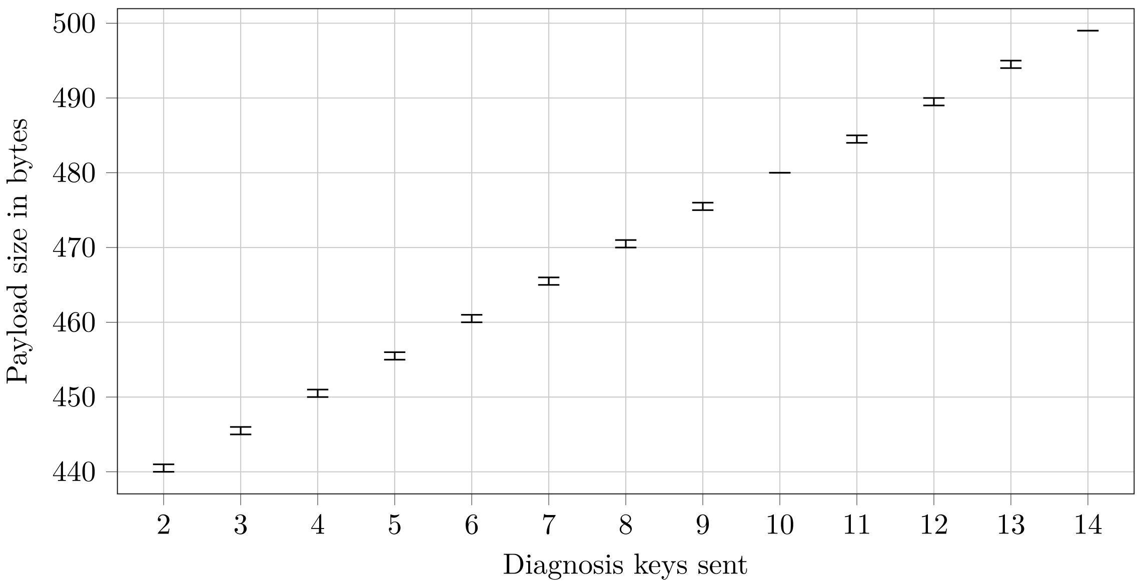


Figure 7.5.: Correlation of diagnosis key payload size with number of keys

measuring their size and comparing it to the number of diagnosis keys in each payload, as in Figure 7.5, we can see the payload size vary with the amount of transmitted keys. In fact, the payload size differs by only 1 byte for each amount of diagnosis keys. Note that instead of using the benchmark's amount of generated keys, we decode the payload using protoc and count the amount of transmitted keys, which eliminates the benchmark inaccuracies.

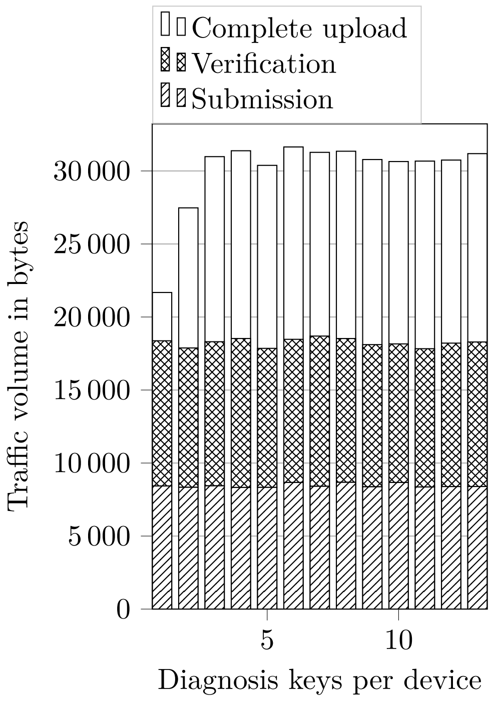
This confirms our discovery from Section 5.2.2 that the payload padding mechanism is broken and does not hide the number of uploaded diagnosis keys from a passive attacker observing the exchange. Due to the playbook system of CWA, messages are always sent in the same order, generating a distinct pattern of sizes even though the communication itself is encrypted.

7.4. Corona-Warn-App

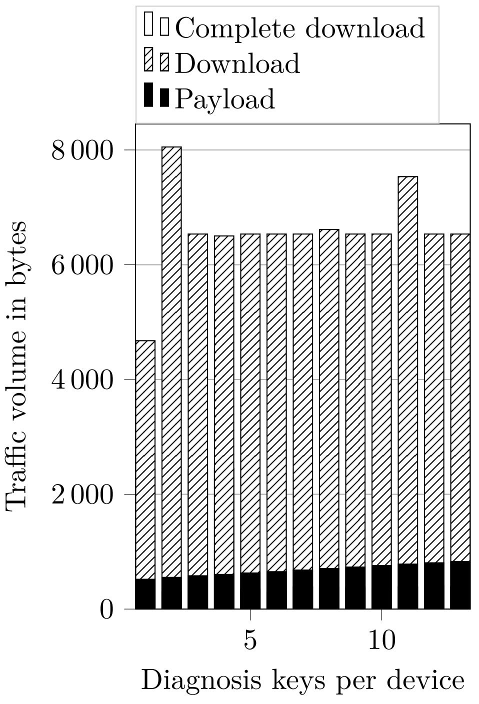
After conducting a number of measurements on the emulated CCTG app, we were able to get the original build of CWA running on the Pixel 3a phone. In this section, we measure the upload and download processes of CWA with a very similar tooling and methodology to the measurements of CCTG. This helps confirm the results from our original test of CCTG.

As the app uses certificate pinning, we used the tools *Frida* and *objection* to allow connection to our server. Additionally, as the app only allows to use TLS cipher suites providing perfect forward secrecy, we opted to use the same cipher suites as in the production version of CWA. We were unable to extract the pre-master secrets needed to decrypt the resulting TLS connections from nginx and opted to instead capture unencrypted traffic behind the reverse proxy. To verify the results gathered with CCTG, we re-ran the traffic measurements with one device, resulting in Figure 7.6.

Figure 7.6(a) shows the upload process. Notably, there is some variation in the submission and verification traffic volumes. This can be attributed to the device being connected over a shared Wi-Fi network and having to re-send packets. When comparing results to Figure 7.3, notably, the upload payload size is missing. This is due to the submission



(a) Upload process



(b) Download/matching process

Figure 7.6.: Corona-Warn-App traffic volume per device, 1 device

backend service using an encrypted connection even behind the reverse proxy, leaving us unable to gain access to the data. Unfortunately, we are unable to verify the payload padding bug due to this restriction. Overall, results are similar to the measurements performed with CCTG, but CWA actually submits one diagnosis key less to the server than CCTG. This is the expected behavior of the app, while CCTG includes one extra key that is valid for the current day.

In Figure 7.6(b), an attempt at measuring the download process is shown. CWA proved to be very temperamental about actually downloading new diagnosis key data and we were only able to get three measurements of it downloading in total. The measured download numbers in the graph show the app querying the server for diagnosis key packages immediately after the key upload, when keys were not yet distributed. Therefore, these measurements are unreliable.

Besides the one additional diagnosis key generated in CCTG, we observed no difference in traffic and functionality between the open-source reimplementation of GAEN from the microG project, which is used in CCTG, and the official version included in Google Play Services.

7.5. Comparison of TraceCORONA and Corona-Warn-App

After testing scripts and methodology for both TraceCORONA and Corona-Warn-App have been implemented, the gathered numbers are now used to compare the performance of GAEN/CWA and TraceCORONA with practical measurements. We repeat the comparison from Section 5.2.4, this time interpolating from our real measured numbers. The result of this interpolation is a more realistic overview of the traffic volume generated by a system over two weeks of operation.

In the first step, the upload and download measurements for TraceCORONA are regressed linearly: $\tau = m_{\text{TC, (up, down)}} \cdot n_{\text{enc}} + t_{\text{TC, (up, down)}}$. For CWA, in a similar vein, the

Constant	Value
$m_{\text{CWA, down}}$	34.628 B
$t_{\text{CWA, down}}$	9 038.821 B
$\tau_{\text{CWA, up, 14 keys}}$	16 987.5 B
$m_{\text{TC, down}}$	343.630 B
$t_{\text{TC, down}}$	5 817.037 B
$m_{\text{TC, up}}$	157.273 B
$t_{\text{TC, up}}$	11 314.727 B
n_{days}	14

Table 7.1.: Trend values and parameters for interpolation

average size of the matching process is regressed linearly, with the measured number of diagnosis keys contained in the payload (as opposed to the desired value) as a variable: $\tau = m_{\text{CWA, down}} \cdot n_{\text{enc}} + t_{\text{CWA, down}}$. In this comparison, uploads are assumed to always contain the full 14 diagnosis keys generated by the app over 2 weeks of operation. However, as CWA upload packets are padded, the difference in size is neglegible even on a large scale. Table 7.1 contains these values as extracted from the benchmark results.

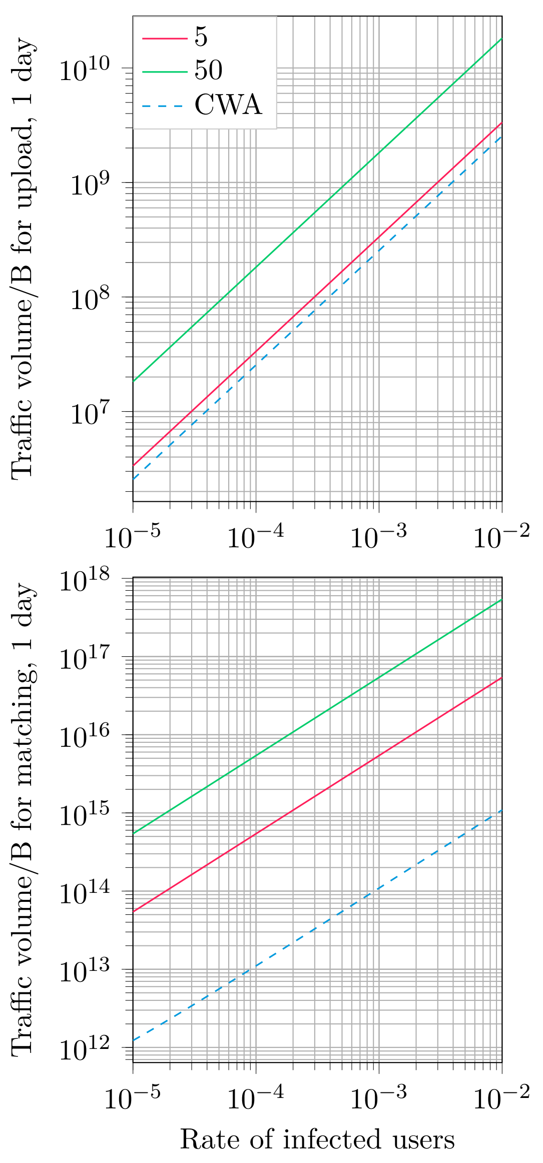
These values are then interpolated as per the equations described in Section 5.2.4. Again, TraceCORONA is analyzed in a low contact and high contact scenario. Figure 7.7 shows the results of this interpolation. Upload traffic is similar between both TraceCORONA in the low contact scenario and CWA, which is due to the test result retrieval and verification processes among with fake requests being included in the measurements. Download traffic is massively increased for TraceCORONA (2-3 orders of magnitude higher) due to the inefficient streaming mechanism and high amount of additional data being transmitted with every encounter token.

To summarize, these results show that the current version of TraceCORONA generates a traffic volume orders of magnitude higher than a GAEN-based approach such as CWA. We have shown in Section 5.2 that there is potential for bandwidth savings without loss of any current functionality. In a real-world deployment scenario of an encounter-based approach such as TraceCORONA, measures could also be taken to reduce the amount of keys distributed to each user such as separating the keys by region.

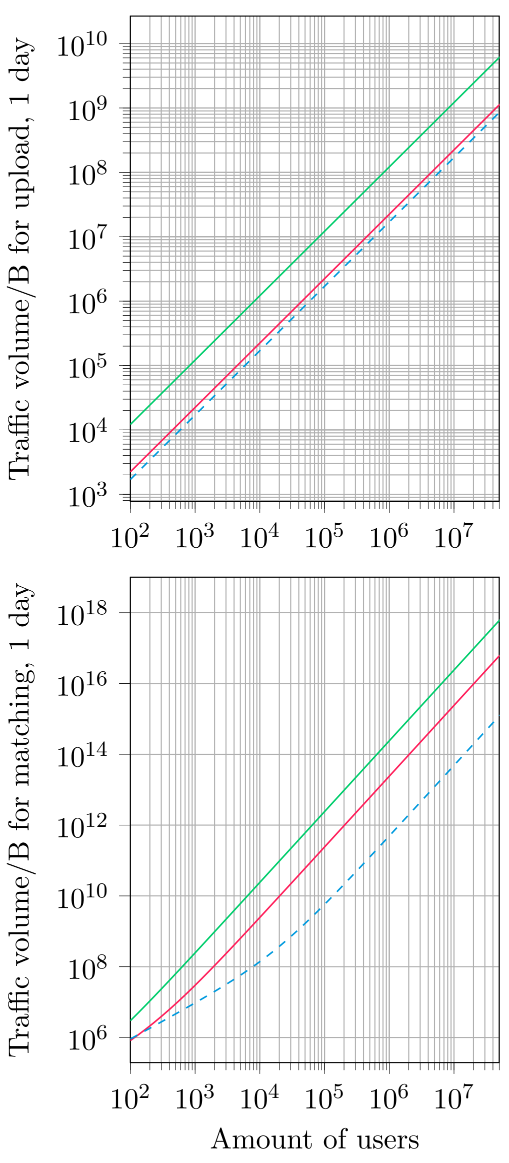
7.6. Comparison of Benchmarks and Theory

In two parts of this thesis, we have compared the estimated traffic for the two systems. We ask the question: Could TraceCORONA be used as a backend for a national tracing system? To evaluate this scenario, due to the amount of non-infected users which only download keys for matching purposes and do not upload keys, we can ignore the upload processes on a large scale, and are therefore only comparing the matching processes.

Figure 7.8 shows the results of our two comparisons side by side. Note that the dotted lines in Figure 7.8(a) are equivalent to the solid lines in Figure 7.8(b). Here we can see that both comparisons yield similar traffic volumes for the original TraceCORONA, with the values for our theoretical numbers being slightly lower, as expected due to leaving the overhead of underlying protocols out of the picture. Therefore, our estimates from Section 5.2 and our measurements from Chapter 7 are able to provide an overview of the traffic produced by both systems and serve as a means to estimate the feasibility of scaling up both TraceCORONA and GAEN.

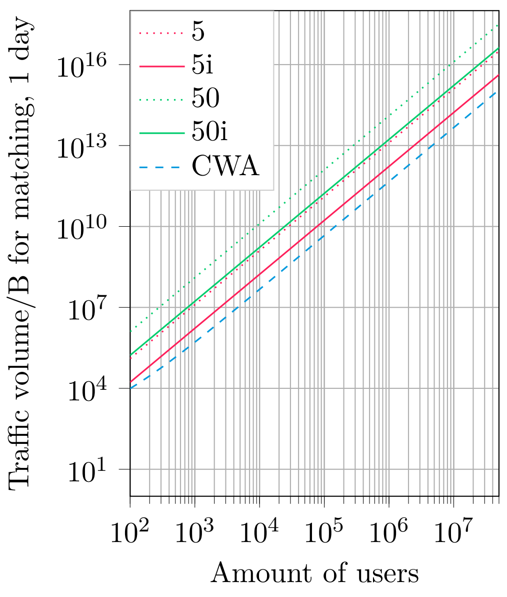


(a) Varying rate of infected users, 15 000 000 users

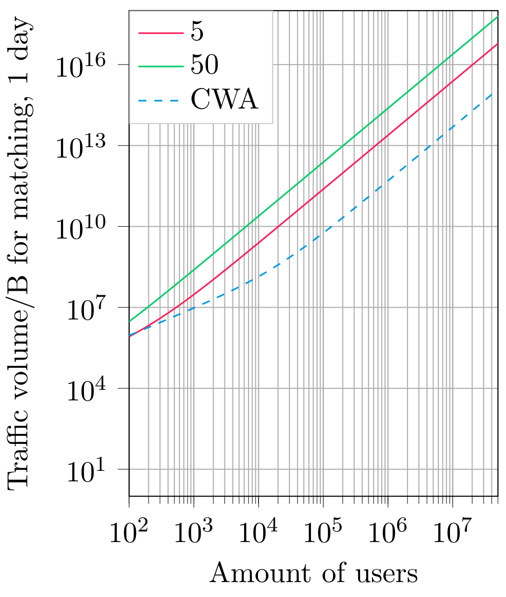


(b) Varying amount of total users, rate of infected users = 0.001

Figure 7.7.: Comparison of TraceCORONA (5 or 50 average encounters/person/day) and Corona-Warn-App (CWA) over a 14-day period, logarithmic scale for both axes



(a) Theoretical numbers, from Figure 5.2



(b) Interpolated measurements, from Figure 7.7

Figure 7.8.: Comparison of TraceCORONA (5 or 50 average encounters/person/day) and Corona-Warn-App (CWA) matching traffic per day, logarithmic scale for both axes, rate of infected users = 0.001

What can we gather from these results? TraceCORONA, although producing vastly higher traffic in the prototype, can be optimized to only need a small multiple of the traffic volume of CWA at high user counts. To further demonstrate this result, we compare TraceCORONA with CWA's situation on December 23, 2020, as recorded by [Böh20]. On this day, a record 57 201 diagnosis keys were submitted by infected users and between 24.2 and 24.9 million users had downloaded the app. CWA requires a total of $4.7 \cdot 10^{13}$ B of matching traffic volume, while the optimized TraceCORONA version requires between $1.7 \cdot 10^{14}$ B and $1.7 \cdot 10^{15}$ B for 5 and 50 average encounters per person per day respectively. Here, TraceCORONA produces between 3.6 and 36.4 times as much traffic as CWA. For comparison, the largest German internet exchange DE-CIX Frankfurt averages a daily traffic of $5.9 \cdot 10^{17}$ B based on 1-year statistics as of October 3, 2021 [DE-21].

8. Conclusion

The aim of this thesis was to further document and evaluate TraceCORONA, an alternative to the established Google/Apple Exposure Notification(s) (GAEN) tracing protocol. Based on in-depth analyses of both systems, we conclude that TraceCORONA can achieve a higher resilience against security and privacy attacks, while being feasible for an operator to implement in a large scale tracing system similar to Corona-Warn-App (CWA). Novel implementation issues in CWA and potential design weaknesses in TraceCORONA were discovered, discussed and reported.

Throughout the course of this thesis we conducted several different analyses, experiments and measurements. This also left behind a number of unanswered questions and pointers for future research on the topic. Below, we list ideas for future work:

- Implementation
  - **Implementation of proposed protocol extensions and improvements for TraceCORONA**: During our architectural and security analyses we proposed to improve the TraceCORONA protocol by removing the nonce from distributed token messages and instead duplicating or moving the transmission power to this field. Additionally, we proposed to implement risk scoring similar to CWA into the application. Implementing these proposals would move the TraceCORONA prototype towards being a more full-fledged protocol and tracing application.
- Architectural Analysis and Comparison
  - **Server-Device Encoding and Compression**: A large part of the bandwidth used by a tracing system is the fixed overhead in every packet sent to or from the system. Most of it is caused by the usage of HTTP/1.1 for transmission. Switching to the more recent HTTP/3 protocol could yield significant benefits that have the potential to partially mitigate the influence of a tracing protocol sending more data for purposes of unlinkability.
- Security Analysis
  - **Profiling by server-device communication**: In our analysis of the fake request scheme used by CWA, we discovered some discrepancies that could be used by attackers to negate the effect of this plausible deniability scheme. A proof of concept of this attack could be created to further demonstrate the issue at hand and possibly lead to its fix.

  - **Sybil attack on GAEN**: We applied a known attack from peer-to-peer systems to contact tracing systems: the Sybil Attack. However, we were unable to provide results on whether the current version of GAEN on Android and/or iOS is vulnerable to the attack as well. Despite it being preventable, the attack may have already caused false positives if it is possible.
  - **Proof of concept of Sybil attack**: Both for TraceCORONA and CWA or other GAEN-based systems, a proof of concept attack application could be developed to further demonstrate, test and refine the Sybil attack, similar to the work in [BDF+20].
  - **Formal analysis using different tools**: We chose Verifpal for our formal analysis. Instead, another tool such as ProVerif [Bla01] or Tamarin Prover [MSCB13] could be applied to the GAEN and TraceCORONA protocols to prove them as secure or discover yet-unknown attacks.

- Benchmarks
  - **Interpolation of results based on real numbers**: We have interpolated the measured traffic values from our apps based on a hypothetical scenario. However, the real usage numbers of CWA required to make more accurate estimates are available and could be used to estimate the actual bandwidth used by the system at all times, as well as the total traffic volume.
  - **Power measurements**: We evaluated the internet traffic caused by the operation of tracing apps. Another important tradeoff is the power consumed by the tracing process running in the background. As the TraceCORONA prototype is not optimized for power, a fair comparison with CWA was not possible at this time. If TraceCORONA is optimized for power in the future, comparative measurements can be conducted.
  - **Functional testing**: The functionality of TraceCORONA could be evaluated in a range of laboratory and real-world tests to test the claims made by Google that "forms of two-way interactions between participants' phones result in error-prone implementations" [Goo20e].

- Future Research Directions
  - **Re-implementations of GAEN**: Projects such as microG [mic21] have re-implemented the GAEN tracing protocol. Are these re-implementations equivalent in security, privacy and functionality or do they have flaws in this regard?
  - **Alternative devices**: Singapore introduced a centralized tracing system based on dedicated hardware [Gov20b]. Could the decentralized protocols developed for tracing apps work on non-smartphone devices which do not have a permanent internet connection and need to preserve energy and how would the properties change?
  - **Expand analysis to more tracing protocols/apps**: The focus of this thesis were the GAEN protocol widely used in different countries and the experimental TraceCORONA protocol among its prototype, as well as the German implementation of GAEN, CWA. Of course, other tracing systems both in the real world and in academia can be the focus of future research and have the potential to fix security issues and/or provide a better balance of security, privacy and efficiency than existing systems.
  - **Investigate usage of Private Set Intersection**: Established protocols for private set intersection are used in contact discovery. This goal seems to be

well-aligned at first glance with digital contact tracing. Could these protocols be used as an alternative approach for contact tracing and which benefits or drawbacks in terms of tradeoffs would private set intersection algorithms have?

List of Figures

1.1. Overview of the TraceCORONA system, from [MNS20] 2

2.1. Structure of the GATT hierarchy, from [Blu19, p.285] 6
2.2. Comparison of matching approaches 8
2.3. Steps of the TraceCORONA protocol 12
2.4. Sequence diagram of Bluetooth LE contact establishment in TraceCORONA 13
2.5. Sequence diagram of server-device communication in TraceCORONA 15

3.1. Transmission Risk Levels of CWA diagnosis keys from July 5, 2020 25

4.1. Simplified package diagram of TraceCORONA app 28
4.2. TraceCORONA Entity Relationship diagram 29
4.3. Screenshots of TraceCORONA user interface 31

5.1. Schematic overview of decentralized contact tracing 35
5.2. Comparison of TraceCORONA (raw) and CWA, based on theoretical numbers 40
5.3. “High risk” message of CWA, from [Cor21a] 44

6.1. Impact analysis of encounter injection ($N_s = 1$, solid lines) and Sybil attack ($N_s > 1$, dashed and dash-dotted lines) 60

7.1. TraceCORONA traffic volume per device, 1 device 65
7.2. TraceCORONA traffic volume per device, 2 devices 66
7.3. Corona Contact Tracing Germany traffic volume per device, 1 device 67
7.4. Corona Contact Tracing Germany traffic volume per device, 5 devices . . . 68
7.5. Correlation of diagnosis key payload size with number of keys 69
7.6. Corona-Warn-App traffic volume per device, 1 device 70
7.7. Comparison of TraceCORONA and Corona-Warn-App 72
7.8. Comparison of TraceCORONA and CWA matching traffic volume, theoretical and measured 73

B.1. Full TraceCORONA package diagram 112

List of Tables

5.1. Corona-Warn-App encounter token upload and download payload size . . . 38
5.2. TraceCORONA encounter token upload and download payload size 39
5.3. Frameworks used by open-source official national tracing apps 43

6.1. Overview of attacks on GAEN and TraceCORONA 48
6.2. Impact analysis of encounter injection and Sybil attack 60

7.1. Trend values and parameters for interpolation 71

List of Listings

1. tracecorona.vp 99
2. tracecorona-3people-simple.vp 103
3. coronawarn.vp 107

Acronyms

ACLU American Civil Liberties Union 10, 22

ADB Android Debug Bridge 64

AEM Associated Encrypted Metadata 45, 51, 56, 57

API Application Programming Interface 9, 21, 27, 43–45, 57, 65

ATT Attribute Protocol 5, 6

BLE/Bluetooth LE Bluetooth Low Energy . v, vii, 1, 5–7, 12, 17–23, 31, 32, 44, 45, 47, 49–51, 53, 58, 59, 66

BR/EDR Basic Rate/Extended Data Rate 5, 17

CCC Chaos Computer Club 10, 11

CCTG Corona Contact Tracing Germany 63–65, 69, 70

CVSS Common Vulnerability Scoring System 53, 54

CWA Corona-Warn-App . 2, 3, 9, 21–24, 35, 36, 38, 39, 41–45, 47, 49, 56, 58, 60, 63–65, 69–71, 75, 76

DAO Data Access Object 30

DSOS Days Since Onset of Symptoms 21, 35, 37, 44, 66

ECDH Elliptic Curve Diffie-Hellman 12, 20, 32, 39, 45, 47, 55, 58, 59

EFF Electronic Frontier Foundation 10

GAEN Google/Apple Exposure Notification(s) v, 2, 3, 9, 11, 19–23, 35–37, 39, 41, 43–45, 47, 49–52, 54, 56–65, 70, 71, 75, 76

GATT Generic Attribute Profile 5, 6, 12, 31

HKDF HMAC-based Extract-and-Expand Key Derivation Function 12

HMAC Hashed Message Authentication Code 12, 85

HTTP Hypertext Transport Protocol 39, 42, 75

JSON JavaScript Object Notation 23, 39

NGO	Non-Governmental Organization	10, 49
OS	Operating System	2, 43, 64
PDU	Protocol Data Unit	5, 6
RPI	Rolling Proximity Identifier	21, 23, 37, 44, 45, 56, 57, 60
RSSI	Received Signal Strength Indication	12, 28, 49
SDK	Software Development Kit	43, 49
TAN	Transaction Authentication Number	14, 21, 30, 33, 35, 36, 41, 49, 55, 58, 66
TCN	Temporary Contact Numbers	1, 19
TEK	Temporary Exposure Key	21, 23, 24, 35–38, 44, 52, 56, 66
TLS	Transport Layer Security	22, 42, 54–56
TRL	Transmission Risk Level	21, 24, 35, 37, 44, 49, 66
UUID	Universally Unique Identifier	6, 12
WPAN	Wireless Personal Area Network	5

Bibliography

[AAA+20] M. A. Azad *et al.*, “A First Look at Privacy Analysis of COVID-19 Contact Tracing Mobile Applications”, *IEEE Internet of Things Journal*, pp. 1–1, 2020, ISSN: 2327-4662. DOI: `10.1109/JIOT.2020.3024180`.

[ABIV20] G. Avitabile, V. Botta, V. Iovino, and I. Visconti, “Towards Defeating Mass Surveillance and SARS-CoV-2: The Pronto-C2 Fully Decentralized Automatic Contact Tracing System”, 2020/493, 2020-09-27. [Online]. Available: `https://eprint.iacr.org/2020/493` (visited on 2021-03-12).

[ABIV21] ——, “Towards Defeating Mass Surveillance and SARS-CoV-2: The Pronto-C2 Fully Decentralized Automatic Contact Tracing System”, presented at the Workshop on Secure IT Technologies against COVID-19 (CoronaDef) 2021, Virtual, 2021-02-21. [Online]. Available: `https://www.ndss-symposium.org/wp-content/uploads/coronadef2021_23013_paper.pdf`.

[ACK+21] B. Auerbach *et al.*, “Inverse-Sybil Attacks in Automated Contact Tracing”, in *Topics in Cryptology – CT-RSA 2021*, ser. Lecture Notes in Computer Science, K. G. Paterson, Ed., vol. 12704, Cham: Springer International Publishing, 2021, pp. 399–421, ISBN: 978-3-030-75538-6 978-3-030-75539-3. DOI: `10.1007/978-3-030-75539-3_17`.

[AFV21a] G. Avitabile, D. Friolo, and I. Visconti, “TEnK-U: Terrorist Attacks for Fake Exposure Notifications in Contact Tracing Systems”, 2020/1150, 2021-04-15. [Online]. Available: `https://eprint.iacr.org/2020/1150`.

[AFV21b] ——, “Terrorist Attacks for Fake Exposure Notifications in Contact Tracing Systems”, in *Applied Cryptography and Network Security*, K. Sako and N. O. Tippenhauer, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 220–247, ISBN: 978-3-030-78372-3. DOI: `10.1007/978-3-030-78372-3_9`.

[AG20a] Apple and Google. “Exposure Notification – Bluetooth Specification v1.2.2”. (2020-04), [Online]. Available: `https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf` (visited on 2020-12-22).

[AG20b] ——, “Exposure Notification – Cryptography Specification v1.2.1”. (2020-04), [Online]. Available: `https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf` (visited on 2020-12-22).

[AG20c] ——, “Privacy-Preserving Contact Tracing”. (2020-05), [Online]. Available: `https://www.apple.com/covid19/contacttracing` (visited on 2021-10-05).

[Alb20] J. Albright. “The Pandemic App Ecosystem: Investigating 493 Covid-Related iOS Apps across 98 Countries”, Medium. (2020-10-28), [Online]. Available: `https://d1gi.medium.com/the-pandemic-app-ecosystem-investigating-493-covid-related-ios-apps-across-98-countries-cdca305b99da` (visited on 2020-12-03).

[AMX+20] N. Ahmed *et al.*, “A Survey of COVID-19 Contact Tracing Apps”, *IEEE Access*, vol. 8, pp. 134 577–134 601, 2020, ISSN: 2169-3536. DOI: `10.1109/ACCESS.2020.3010226`.

[App20] Apple. “detectExposures()”, Apple Developer Documentation. (2020), [Online]. Available: `https://developer.apple.com/documentation/exposurenotification/enmanager/3586331-detectexposures` (visited on 2021-09-19).

[App21a] ——, “Choosing Background Strategies for Your App”, Apple Developer Documentation. (2021), [Online]. Available: `https://developer.apple.com/documentation/backgroundtasks/choosing_background_strategies_for_your_app` (visited on 2020-11-27).

[App21b] ——, “Setting Up a Key Server”, Apple Developer Documentation. (2021), [Online]. Available: `https://developer.apple.com/documentation/exposurenotification/setting_up_a_key_server` (visited on 2021-04-21).

[BDF+20] L. Baumgärtner *et al.*, “Mind the GAP: Security & Privacy Risks of Contact Tracing Apps”, in *Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2020)*, 2020-12-29, pp. 458–467. DOI: `10.1109/TrustCom50675.2020.00069`.

[BH20] K. Bobrowski and M. Huebler, *Kbobrowski/en-i13n*, 2020-07-24. [Online]. Available: `https://github.com/kbobrowski/en-i13n` (visited on 2021-08-19).

[BKT+20] J. Bay *et al.*, “BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders”, White Paper, 2020. [Online]. Available: `https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf`.

[Bla01] B. Blanchet, “An efficient cryptographic protocol verifier based on prolog rules”, in *Proceedings 14th IEEE computer security foundations workshop, 2001.*, 2001-06, pp. 82–96. DOI: `10.1109/CSFW.2001.930138`.

[Blu10] Bluetooth SIG, “Bluetooth Core Specification Version 4.0”, 2010-06-30. [Online]. Available: `https://www.bluetooth.com/specifications/archived-specifications/` (visited on 2020-11-20).

[Blu19] ——, “Bluetooth Core Specification Version 5.2”, 2019-12-31. [Online]. Available: `https://www.bluetooth.com/specifications/bluetooth-core-specification/` (visited on 2020-11-20).

[Böh20] M. Böhme. “Diagnosis Key Analysis (dka)”. (2020), [Online]. Available: `https://micb25.github.io/dka/index_en.html` (visited on 2020-12-03).

[Brö20] M. Bröckling. “Corona-Kontaktlisten - Wenn die Polizei dich nach dem Restaurantbesuch anruft”, netzpolitik.org. (2020-07-07), [Online]. Available: `https://netzpolitik.org/2020/corona-kontaktlisten-wenn-die-polizei-dich-nach-dem-restaurantbesuch-anruft/` (visited on 2020-12-04).

[Cha81] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms”, *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981-02, ISSN: 0001-0782, 1557-7317. DOI: `10.1145/358549.358563`.

[CJG20] A. Comninos, D. Johnson, and A. Gillwald. “Has South Africa’s COVID Alert contact tracing app been zero-rated?”, Research ICT Africa. (2020-09-25), [Online]. Available: `https://researchictafrica.net/2020/09/25/has-south-africas-covid-alert-app-been-zero-rated/` (visited on 2021-04-13).

[COC20] A. Crocker, K. Opsahl, and B. Cyphers. “The Challenge of Proximity Apps For COVID-19 Contact Tracing”, Electronic Frontier Foundation. (2020-04-10), [Online]. Available: `https://www.eff.org/deeplinks/2020/04/challenge-proximity-apps-covid-19-contact-tracing` (visited on 2020-11-27).

[con21] C. C. T. G. contributors, *Corona Contact Tracing Germany*, 2021-08-18. [Online]. Available: `https://codeberg.org/corona-contact-tracing-germany/cwa-android` (visited on 2021-08-19).

[Cor20a] Corona-Warn-App Server Developers. “CWA-Server Distribution Service”, GitHub. (2020-12-10), [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/main/docs/DISTRIBUTION.md` (visited on 2020-12-22).

[Cor20b] ——, “CWA-Server Submission Service”, GitHub. (2020-12-22), [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/main/docs/SUBMISSION.md` (visited on 2021-04-08).

[Cor20c] ——, *FakeRequestController.java*, SAP, T-Systems, 2020-10-01. [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/c5002cb222763787628bf35ea04b9b0c18e36852/services/submission/src/main/java/app/coronawarn/server/services/submission/controller/FakeRequestController.java` (visited on 2021-10-03).

[Cor21a] Corona-Warn-App Developers. “Screenshots”, Open-Source Project Corona-Warn-App. (2021-09-22), [Online]. Available: `https://www.coronawarn.app/en/screenshots/#` (visited on 2021-09-24).

[Cor21b] Corona-Warn-App Server Developers, *Risk-calculation-parameters-1.15.yaml*, SAP, T-Systems, 2021-04-15. [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/main/services/distribution/src/main/resources/main-config/v2/risk-calculation-parameters-1.15.yaml` (visited on 2021-09-29).

[Cor21c] ——, *SubmissionController.java*, SAP, T-Systems, 2021-08-20. [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/c5002cb222763787628bf35ea04b9b0c18e36852/services/submission/src/main/java/app/coronawarn/server/services/submission/controller/SubmissionController.java` (visited on 2021-10-03).

[Cor21d] ——, *Trl-value-mapping-1.15.yaml*, SAP, T-Systems, 2021-03-25. [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/c5002cb222763787628bf35ea04b9b0c18e36852/common/persistence/src/main/resources/trl-value-mapping-1.15.yaml` (visited on 2021-09-29).

[Cor21e] ——, *ValidSubmissionPayload.java*, SAP, T-Systems, 2021-08-12. [Online]. Available: `https://github.com/corona-warn-app/cwa-server/blob/0af3a26cc81f54879bf2cd9e2336f1fbf7edccda/services/submission/src/main/java/app/coronawarn/server/services/submission/validation/ValidSubmissionPayload.java#L96` (visited on 2021-10-03).

[DE-21] DE-CIX Management GmbH. "DE-CIX Frankfurt traffic statistics", DE-CIX. (2021-10-03), [Online]. Available: `https://de-cix.net/en/locations/frankfurt/statistics` (visited on 2021-10-03).

[DHHE07] S. Del Valle, J. Hyman, H. Hethcote, and S. Eubank, "Mixing patterns between age groups in social networks", *Social Networks*, vol. 29, no. 4, pp. 539–554, 2007-10, ISSN: 03788733. DOI: `10.1016/j.socnet.2007.04.005`.

[Dil20] R. Dillet. "France releases contact-tracing app StopCovid", TechCrunch. (2020-06-02), [Online]. Available: `https://social.techcrunch.com/2020/06/02/france-releases-contact-tracing-app-stopcovid-on-android/` (visited on 2020-12-18).

[Doc21] Docker. "Container Runtime with Docker Engine", Docker. (2021), [Online]. Available: `https://www.docker.com/products/container-runtime` (visited on 2021-07-04).

[Dou02] J. R. Douceur, "The Sybil Attack", in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2002, pp. 251–260, ISBN: 978-3-540-45748-0. DOI: `10.1007/3-540-45748-8_24`.

[DR20] P.-O. Dehaye and J. Reardon, "Proximity Tracing in an Ecosystem of Surveillance Capitalism", in *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, ser. WPES'20, New York, NY, USA: Association for Computing Machinery, 2020-11-09, pp. 191–203, ISBN: 978-1-4503-8086-7. DOI: `10.1145/3411497.3420219`.

[EL20] D. Etherington and N. Lomas. "Apple and Google update joint coronavirus tracing tech to improve user privacy and developer flexibility", TechCrunch. (2020-04-24), [Online]. Available: `https://techcrunch.com/2020/04/24/apple-and-google-update-joint-coronavirus-tracing-tech-to-improve-user-privacy-and-developer-flexibility/` (visited on 2021-09-21).

[FIR19] FIRST.Org, Inc. "CVSS v3.1 Specification Document", FIRST — Forum of Incident Response and Security Teams. (2019-06), [Online]. Available: `https://www.first.org/cvss/v3.1/specification-document` (visited on 2021-09-27).

[Gil20] D. K. Gillmor, "Principles for Technology-Assisted Contact-Tracing", White Paper, 2020-04-16. [Online]. Available: `https://www.aclu.org/report/aclu-white-paper-principles-technology-assisted-contact-tracing`.

[Goo20a] Google, *BluetoothMetadata.java*, Google, 2020-08-25. [Online]. Available: `https://github.com/google/exposure-notifications-internals/blob/aaada6ce5cad0ea1493930591557f8053ef4f113/exposurenotification/src/main/java/com/google/samples/exposurenotification/data/BluetoothMetadata.java` (visited on 2021-08-30).

[Goo20b] ——, *ContactTracingFeature.java*, Google, 2020-08-25. [Online]. Available: `https://github.com/google/exposure-notifications-internals/blob/aaada6ce5cad0ea1493930591557f8053ef4f113/exposurenotification/src/main/java/com/google/samples/exposurenotification/features/ContactTracingFeature.java` (visited on 2021-10-02).

[Goo20c] ——, “Create a background service”, Android Developers. (2020-06-02), [Online]. Available: `https://developer.android.com/training/run-background-service/create-service` (visited on 2020-11-27).

[Goo20d] ——, “Exposure Key export file format and verification”, Google API for Exposure Notifications. (2020-12-07), [Online]. Available: `https://developers.google.com/android/exposure-notifications/exposure-key-file-format` (visited on 2021-04-21).

[Goo20e] ——, “Exposure Notification: Risks and Mitigations FAQ”, Google, 2020-10-02. [Online]. Available: `https://github.com/google/exposure-notifications-internals/blob/aaada6ce5cad0ea1493930591557f8053ef4f113/en-risks-and-mitigations-faq.md` (visited on 2021-09-30).

[Goo20f] ——, “Exposure Notifications BLE calibration calculation”, Google API for Exposure Notifications. (2020-12-14), [Online]. Available: `https://developers.google.com/android/exposure-notifications/ble-attenuation-computation` (visited on 2021-08-30).

[Goo21a] ——, “AdvertisingSetParameters”, Android Developers. (2021-02-24), [Online]. Available: `https://developer.android.com/reference/android/bluetooth/le/AdvertisingSetParameters` (visited on 2021-10-02).

[Goo21b] ——, “BluetoothAdapter.getBluetoothLeAdvertiser()”, Android Developers. (2021-02-24), [Online]. Available: `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getBluetoothLeAdvertiser()` (visited on 2021-06-03).

[Goo21c] ——, “Buttons: Floating action button”, Material Design. (2021), [Online]. Available: `https://material.io/components/buttons-floating-action-button` (visited on 2021-10-06).

[Goo21d] ——, “Cards”, Material Design. (2021), [Online]. Available: `https://material.io/components/cards` (visited on 2021-10-06).

[Goo21e] ——, “Exposure Notifications API”, Google API for Exposure Notifications. (2021-06-03), [Online]. Available: `https://developers.google.com/android/exposure-notifications/exposure-notifications-api` (visited on 2021-09-19).

[Goo21f] ——, *Google/gson*, Google, 2021-06-03. [Online]. Available: `https://github.com/google/gson` (visited on 2021-06-04).

[Goo21g] ——, “Introduction”, Material Design. (2021), [Online]. Available: `https://material.io/design/introduction` (visited on 2021-05-19).

[Goo21h] ——, “Language Guide | Protocol Buffers”, Google Developers. (2021-03-31), [Online]. Available: `https://developers.google.com/protocol-buffers/docs/overview` (visited on 2021-04-21).

[Goo21i] ——, *Material-components/material-components-android*, Material Components, 2021-05-19. [Online]. Available: `https://github.com/material-components/material-components-android` (visited on 2021-05-19).

[Goo21j] ——, “Room”, Android Developers. (2021-05-17), [Online]. Available: https://developer.android.com/jetpack/androidx/releases/room (visited on 2021-05-18).

[Gor20] E. Gordon. “COVID-19: Lessons from Singapore and how it handled SARS”, The World from PRX. (2020-03-02), [Online]. Available: https://www.pri.org/stories/2020-03-02/covid-19-lessons-singapore-and-how-it-handled-sars (visited on 2020-12-17).

[Gos05] J. Gossman. “Introduction to Model/View/ViewModel pattern for building WPF apps”, Tales from the Smart Client. (2005-08-10), [Online]. Available: https://docs.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps (visited on 2021-05-18).

[Gov20a] Government of Singapore. “Home”, BlueTrace. (2020), [Online]. Available: https://bluetrace.io (visited on 2020-12-18).

[Gov20b] ——, “Protecting more people with the TraceTogether Token”, TraceTogether. (2020-07-20), [Online]. Available: https://www.tracetogether.gov.sg/ (visited on 2020-12-04).

[Hip21] D. R. Hipp. “SQLite Home Page”. (2021-04-19), [Online]. Available: https://sqlite.org/index.html (visited on 2021-05-18).

[HL02] M. Howard and D. LeBlanc, *Writing Secure Code*, 2nd ed. Microsoft Press, 2002-12, ISBN: 978-0-7356-9146-9. [Online]. Available: https://learning.oreilly.com/library/view/writing-secure-code/0735617228/ch04.html (visited on 2021-09-28).

[Hoe20] J. Hoerdt. “New features: Corona-Warn-App Version 1.5 is now available for download”, Corona-Warn-App Open Source Project Blog. (2020-10-19), [Online]. Available: https://www.coronawarn.app/en/blog/2020-10-19-version-1-5/ (visited on 2021-03-07).

[Hoe21] ——, “Current facts and figures about the Corona-Warn-App”, Corona-Warn-App Open Source Project Blog. (2021-08-20), [Online]. Available: https://www.coronawarn.app/en/blog/2021-08-20-facts-and-figures/ (visited on 2021-08-22).

[Hol20] M. Holland. “Zero Rating: Mobilfunk-Provider berechnen keinen Traffic für Corona-Warn-App”, heise online. (2020-06-16), [Online]. Available: https://www.heise.de/news/Zero-Rating-Mobilfunk-Provider-berechnen-keinen-Traffic-fuer-Corona-Warn-App-4785202.html (visited on 2021-04-13).

[Hou21] R. Houben. “Key figures dashboard for the Corona-Warn-App”, Corona-Warn-App Open Source Project Blog. (2021-10-04), [Online]. Available: https://www.coronawarn.app/en/blog/2021-10-04-key-figures-dashboard/ (visited on 2021-10-05).

[hua20a] huaxia. “China introduces novel coronavirus close contact detection app”, Xinhua | English.news.cn. (2020-02-10), [Online]. Available: http://www.xinhuanet.com/english/2020-02/10/c_138770415.htm (visited on 2020-12-17).

[hua20b] ——, “Shanghai offers health QR codes in public transport”, Xinhua | English.news.cn. (2020-02-25), [Online]. Available: http://www.xinhuanet.com/english/2020-02/25/c_138817911.htm (visited on 2020-12-17).

[Hue20a] M. Huebler. “Available Diagnosis Keys will soon be linkable across 12 days · Issue #620 · corona-warn-app/cwa-server”, GitHub. (2020-06-24), [Online]. Available: `https://github.com/corona-warn-app/cwa-server/issues/620` (visited on 2020-12-03).

[Hue20b] ——, *Mh-/diagnosis-keys*, 2020-11-27. [Online]. Available: `https://github.com/mh-/diagnosis-keys` (visited on 2020-12-03).

[IVV21a] V. Iovino, S. Vaudenay, and M. Vuagnoux, “On the Effectiveness of Time Travel to Inject COVID-19 Alerts”, in *Topics in Cryptology – CT-RSA 2021*, K. G. Paterson, Ed., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 422–443, ISBN: 978-3-030-75539-3. DOI: `10.1007/978-3-030-75539-3_18`.

[IVV21b] ——, “On the effectiveness of time travel to inject COVID-19 alerts”, 2020/1393, 2021-01-27. [Online]. Available: `https://eprint.iacr.org/2020/1393`.

[JS 21] JS Foundation. “Appium: Mobile App Automation Made Awesome.” (2021), [Online]. Available: `https://appium.io/` (visited on 2021-07-04).

[KE10] H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF)”, Internet Engineering Task Force (IETF), 5869, 2010-05. [Online]. Available: `https://datatracker.ietf.org/doc/html/rfc5869` (visited on 2021-05-19).

[KNT20] N. Kobeissi, G. Nicolas, and M. Tiwari, “Verifpal: Cryptographic Protocol Analysis for the Real World”, in *Progress in Cryptology – INDOCRYPT 2020*, K. Bhargavan, E. Oswald, and M. Prabhakaran, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 151–202, ISBN: 978-3-030-65277-7. DOI: `10.1007/978-3-030-65277-7_8`.

[Kob21] N. Kobeissi, *Verifpal User Manual*, 2021-07-01.

[Kub21] M. Kubica, *Leonidas-from-XIV/node-xml2js*, 2021-07-03. [Online]. Available: `https://github.com/Leonidas-from-XIV/node-xml2js` (visited on 2021-07-04).

[LeB07] D. LeBlanc. “DREADful”, David LeBlanc’s Web Log. (2007-08-14), [Online]. Available: `https://docs.microsoft.com/en-us/archive/blogs/david_leblanc/dreadful` (visited on 2021-09-28).

[Leg20] Legion of the Bouncy Castle. “The Legion of the Bouncy Castle Java Cryptography APIs”. (2020-12-21), [Online]. Available: `https://bouncycastle.org/java.html` (visited on 2021-05-19).

[LF20] D. J. Leith and S. Farrell, “Measurement-based evaluation of Google/Apple Exposure Notification API for proximity detection in a light-rail tram”, *PLOS ONE*, vol. 15, no. 9, J. Soldani, Ed., e0239943, 2020-09-30, ISSN: 1932-6203. DOI: `10.1371/journal.pone.0239943`.

[Lin20] Linux Foundation. “Linux Foundation Public Health - Collaborating to battle COVID-19”, Linux Foundation Public Health. (2020), [Online]. Available: `https://www.lfph.io/` (visited on 2020-12-18).

[LLCR20] D. Lewis, S. Leibrand, B. Carr, and H. Rodda, *Co-Epi/website*, CoEpi, 2020-02-23. [Online]. Available: `https://github.com/Co-Epi/website/tree/aa2cd8f8b2ca03640bbd2c054783d6e6f3acd656` (visited on 2020-12-18).

[Mar20] S. Marsiske, *Full Disclosure: CVE-2020-24722: GAEN Protocol Metadata Deanonymization and Risk-score Inflation Issues*, E-mail, 2020-10-05. [Online]. Available: `https://seclists.org/fulldisclosure/2020/Oct/12` (visited on 2021-08-30).

[MHJ+08] J. Mossong *et al.*, “Social Contacts and Mixing Patterns Relevant to the Spread of Infectious Diseases”, *PLOS Medicine*, vol. 5, no. 3, e74, 2008-03-25, ISSN: 1549-1676. DOI: `10.1371/journal.pmed.0050074`.

[mic21] microG contributors, *Microg/GmsCore*, microG Project, 2021-01-20. [Online]. Available: `https://github.com/microg/GmsCore` (visited on 2021-01-20).

[MNS20] M. Miettinen, T. D. Nguyen, and A.-R. Sadeghi. “How the TraceCORONA App Works”, TraceCORONA. (2020), [Online]. Available: `https://tracecorona.net/how-tracecorona-works/` (visited on 2021-01-14).

[MSCB13] S. Meier, B. Schmidt, C. Cremers, and D. Basin, “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”, in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2013, pp. 696–701, ISBN: 978-3-642-39799-8. DOI: `10.1007/978-3-642-39799-8_48`.

[NAE+21] K. Nomoto, M. Akiyama, M. Eto, A. Inomata, and T. Mori, “Poster: Can the Exposure Notification Framework Expose Personal Information?”, presented at the Network and Distributed Systems Security (NDSS) 2021, Poster Session, San Diego, 2021-02-21/2021-02-25. [Online]. Available: `https://www.ndss-symposium.org/wp-content/uploads/NDSS2021posters_paper_18.pdf`.

[Neu20] L. Neumann. “10 requirements for the evaluation of "Contact Tracing" apps”. (2020-04-06), [Online]. Available: `https://www.ccc.de/en/updates/2020/contact-tracing-requirements` (visited on 2020-11-20).

[NGI21] NGINX. “Nginx”. (2021), [Online]. Available: `https://nginx.org/` (visited on 2021-07-04).

[NMS20] T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi, “Long Live Randomization: On Privacy-preserving Contact Tracing in Pandemic”, in *Proceedings of the 7th ACM Workshop on Moving Target Defense*, ser. MTD’20, New York, NY, USA: Association for Computing Machinery, 2020-11-09, pp. 1–9, ISBN: 978-1-4503-8085-0. DOI: `10.1145/3411496.3421229`.

[OBr11] T. O’Brien. “iPhone 4S claims title of first Bluetooth 4.0 smartphone, ready to stream data from your cat”, Engadget. (2011-10-12), [Online]. Available: `https://www.engadget.com/2011-10-12-iphone-4s-claims-title-of-first-bluetooth-4-0-smartphone-ready.html` (visited on 2020-11-20).

[Ope20] OpenTrace contributors. “OpenTrace”, GitHub. (2020), [Online]. Available: `https://github.com/opentrace-community` (visited on 2020-12-18).

[Ope21a] OpenJS Foundation. “Node.js”, Node.js. (2021), [Online]. Available: `https://nodejs.org/en/` (visited on 2021-07-04).

[Ope21b] ——, “WebdriverIO · Next-gen browser and mobile automation test framework for Node.js”. (2021), [Online]. Available: `https://webdriver.io/` (visited on 2021-07-04).

[ORJ20] P. H. O'Neill, T. Ryan-Mosley, and B. Johnson. "A flood of coronavirus apps are tracking us. Now it's time to keep track of them.", MIT Technology Review. (2020-05-07), [Online]. Available: `https://www.technologyreview.com/2020/05/07/1000961/launching-mittr-covid-tracing-tracker/` (visited on 2020-12-03).

[PF20] PRIVATICS team, Inria and Fraunhofer AISEC, "ROBERT: ROBust and privacy-presERving proximity Tracing", White Paper, 2020-06-02. [Online]. Available: `https://raw.githubusercontent.com/ROBERT-proximity-tracing/documents/master/ROBERT-specification-EN-v1_1.pdf` (visited on 2020-12-18).

[Pfi20] J. Pfister. "Corona Tracker Tracker". (2020), [Online]. Available: `https://ctt.pfstr.de/` (visited on 2020-12-03).

[Pie20] K. Pietrzak, "Delayed Authentication: Preventing Replay and Relay Attacks in Private Contact Tracing", 2020/418, 2020-04-20. [Online]. Available: `https://eprint.iacr.org/2020/418` (visited on 2021-03-12).

[Roo21] F. Roos. "Padding of diagnosis key upload messages (requestPadding) inconsistent/wrong · Issue #591 · corona-warn-app/cwa-documentation", GitHub. (2021-04-22), [Online]. Available: `https://github.com/corona-warn-app/cwa-documentation/issues/591` (visited on 2021-04-22).

[Rot20] J. Rothwell. "Americans' Social Contacts During the COVID-19 Pandemic", Gallup Blog. (2020-04-21), [Online]. Available: `https://news.gallup.com/opinion/gallup/308444/americans-social-contacts-during-covid-pandemic.aspx` (visited on 2021-08-22).

[SB18] S. Stewart and D. Burns, "WebDriver", W3C, W3C Recommendation, 2018-06-05. [Online]. Available: `https://www.w3.org/TR/2018/REC-webdriver1-20180605/` (visited on 2021-07-04).

[SBG+20] M. Schrems, M. Blocher, M. Garrido de Vega, "TBussmann", and other Wiki editors. "Projects using personal data to combat SARS-CoV-2", GDPRhub. (2020-06-25), [Online]. Available: `https://gdprhub.eu/index.php?title=Projects_using_personal_data_to_combat_SARS-CoV-2` (visited on 2020-12-03).

[SF20] C. Smith and R. Farragher. "Tracking coronavirus via bluetooth", Naked Scientists. (2020-03-03), [Online]. Available: `https://www.thenakedscientists.com/articles/interviews/tracking-coronavirus-bluetooth` (visited on 2020-12-17).

[Squ20] Square. "Retrofit", Square Open Source. (2020), [Online]. Available: `https://square.github.io/retrofit/` (visited on 2021-06-04).

[Sta19] States of Jersey. "Population estimates", Government Website. (2019), [Online]. Available: `http://www.gov.je:80/Government/JerseyInFigures/Population/Pages/Population.aspx` (visited on 2021-10-05).

[Sta21] StatCounter. "Mobile Operating System Market Share Worldwide", StatCounter Global Stats. (2021-10), [Online]. Available: `https://gs.statcounter.com/os-market-share/mobile/worldwide` (visited on 2021-10-06).

[Sti20] A. Stiefel. "Software Design Verification Server", GitHub. (2020-07-23), [Online]. Available: `https://github.com/corona-warn-app/cwa-verification-server` (visited on 2020-12-22).

[TCN20a] TCN contributors, *Covidwatchorg/covidwatch-ios-tcn*, Covid Watch, 2020-08-20. [Online]. Available: `https://github.com/covidwatchorg/covidwatch-ios-tcn` (visited on 2020-12-18).

[TCN20b] ——, *TCNCoalition/TCN*, TCN Coalition, 2020-12-11. [Online]. Available: `https://github.com/TCNCoalition/TCN` (visited on 2020-12-18).

[The21a] The PostgreSQL Global Development Group. "8.4. Binary Data Types", PostgreSQL Documentation. (2021-02-11), [Online]. Available: `https://www.postgresql.org/docs/13/datatype-binary.html` (visited on 2021-04-21).

[The21b] The Tcpdump Group. "TCPDUMP/LIBPCAP public repository". (2021), [Online]. Available: `https://www.tcpdump.org/` (visited on 2021-07-04).

[Thu13] Thucydides, *Thucydides: The War of the Peloponnesians and the Athenians*, J. Mynott, Ed., trans. by J. Mynott, ser. Cambridge Texts in the History of Political Thought. Cambridge: Cambridge University Press, 2013, ISBN: 978-0-521-84774-2. DOI: `10.1017/CBO9781139050371`.

[TPH+20a] C. Troncoso *et al.*, "Decentralized Privacy-Preserving Proximity Tracing (First Version)", White Paper, 2020-04-03. [Online]. Available: `https://github.com/DP-3T/documents` (visited on 2020-12-22).

[TPH+20b] ——, "Decentralized Privacy-Preserving Proximity Tracing (Latest Version)", White Paper, 2020-05-25. [Online]. Available: `https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf` (visited on 2020-12-22).

[Uni21] United Nations Department of Economic and Social Affairs, *World Population Prospects 2017 - Volume I: Comprehensive Tables.* United Nations, 2021-08-27, ISBN: 978-92-1-000101-4. DOI: `10.18356/9789210001014`.

[US 21] US Census Bureau. "Annual Estimates of the Resident Population for the United States, Regions, States, and the District of Columbia: April 1, 2010 to July 1, 2020", The United States Census Bureau. (2021-07-23), [Online]. Available: `https://www.census.gov/programs-surveys/popest/technical-documentation/research/evaluation-estimates/2020-evaluation-estimates/2010s-totals-national.html` (visited on 2021-10-05).

[Var20] Various Authors, *Joint Statement on Contact Tracing*, Letter, 2020-04-19. [Online]. Available: `https://drive.google.com/file/d/1OQg2dxPu-x-RZzETlpV3lFa259Nrpk1J/view` (visited on 2020-12-18).

[Vau20] S. Vaudenay, "Analysis of DP3T", 399, 2020. [Online]. Available: `https://eprint.iacr.org/2020/399` (visited on 2021-09-25).

[VV20] S. Vaudenay and M. Vuagnoux, Eds., *Analysis of SwissCovid.* 2020.

[Wik20] Wikipedia contributors, *COVID-19 apps*, in *Wikipedia*, 2020-12-03. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=COVID-19_apps&oldid=992019750` (visited on 2020-12-03).

[Wik21] ——, *Exposure Notification*, in *Wikipedia*, 2021-09-11. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Exposure_Notification&oldid=1000033849` (visited on 2021-01-19).

[Wir21] Wireshark Foundation. "Wireshark · Go Deep." (2021), [Online]. Available: `https://www.wireshark.org/` (visited on 2021-07-06).

[Wis21] J. Wise, *JoshuaWise/better-sqlite3*, 2021-07-04. [Online]. Available: `https://github.com/JoshuaWise/better-sqlite3` (visited on 2021-07-04).

[Wol20] S. Wolf. "How does the Corona-Warn-App identify an increased risk?", GitHub. (2020-12-06), [Online]. Available: `https://github.com/corona-warn-app/cwa-documentation/blob/master/cwa-risk-assessment.md` (visited on 2020-12-22).

[Wor21] World Health Organization. “WHO Coronavirus (COVID-19) Dashboard”. (2021), [Online]. Available: `https://covid19.who.int` (visited on 2021-08-22).

[Wu21] J. Wu, *Topjohnwu/Magisk*, 2021-07-04. [Online]. Available: `https://github.com/topjohnwu/Magisk` (visited on 2021-07-04).

[Yon09] E. Yoneki. “FluPhone Specification”. (2009), [Online]. Available: `https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/fluphone2/fluphone_spec.html` (visited on 2020-12-17).

[Yon11] ——, “FluPhone study: Virtual disease spread using haggle”, in *Proceedings of the 6th ACM workshop on Challenged networks - CHANTS ’11*, Las Vegas, Nevada, USA: ACM Press, 2011, p. 65, ISBN: 978-1-4503-0870-0. DOI: `10.1145/2030652.2030672`.

Appendix

A. Verifpal Models

Listing 1: tracecorona.vp

```
attacker[active]

// Beforehand: Server certificate signed by CA, guarded against attacks -> []

principal Server[
    knows private ServerPrivateKey
    ServerPublicKey = G^ServerPrivateKey
]

principal CA [
    knows private CAPrivateKey
    CAPublicKey = G^CAPrivateKey
]

CA -> UserI: [CAPublicKey]
CA -> UserJ: [CAPublicKey]
CA -> HealthAuthority: [CAPublicKey]
Server -> CA: [ServerPublicKey]

principal CA [
    ServerCert = SIGN(CAPrivateKey, ServerPublicKey)
]

CA -> Server: [ServerCert]

// Phones generate keys, rolling IDs

principal UserI[
    // literal post box and key of the user, for delivery of TANs
    // modelled using PKE
    knows private UserIPostboxKey
    UserIPostbox = G^UserIPostboxKey

    // protocol values
    generates UserIPrivateKey1
    UserIPublicKey1 = G^UserIPrivateKey1
    generates UserIRollingID1
]

principal UserJ[
    generates UserJPrivateKey1
    UserJPublicKey1 = G^UserJPrivateKey1
    generates UserJRollingID1
]

// Advertisement/Scanning: UserJ discovers UserI is nearby

UserI -> UserJ: UserIRollingID1

// UserJ begins App-to-App handshake using (EC)DH

UserJ -> UserI: UserJPublicKey1
UserI -> UserJ: UserIPublicKey1

principal UserI[
    UserIEncToken1 = UserJPublicKey1^UserIPrivateKey1
    UserIEncTokenHash1 = HASH(UserIEncToken1)
]

principal UserJ[
    UserJEncToken1 = UserIPublicKey1^UserJPrivateKey1
    UserJEncTokenHash1 = HASH(UserJEncToken1)
]

// UserI is tested positive, health authority distributes TAN secretly to user and server

UserI -> HealthAuthority: [UserIPostbox]

principal HealthAuthority[
    generates TAN
    // Note: PKE_ENC in this case is likely a letter or phone call
    // An attacker cannot intercept the letter, hence does not gain knowledge or is able to modify the TAN in this model
    TANForUser = PKE_ENC(UserIPostbox, TAN)
    // Server-side communication over TLS with DHE
    generates HAClientRandom
]

HealthAuthority -> UserI: [TANForUser]

principal UserI[
    // User unlocks postbox and opens letter with positive test result + TAN
    UserTAN = PKE_DEC(UserIPostboxKey, TANForUser)
]

// ClientHello
HealthAuthority -> Server: HAClientRandom

principal Server[
    generates HAServerRandom
    generates HADHServerSecret
    HADHServerParam = G^HADHServerSecret
]
```

```

// ServerHello
Server -> HealthAuthority: HAServerRandom, HADHServerParam, ServerPublicKey,
    ServerCert

principal HealthAuthority[
    knows public HAsalt, info, tanAD
    HAServerOK = SIGNVERIF(CAPublicKey, ServerPublicKey, ServerCert)?
    generates HADHClientSecret
    HADHClientParam = G^HADHClientSecret
    HADHClientPreMasterSecret = HADHServerParam^HADHClientSecret
    HADHClientTrafficKey = HKDF(HAsalt, HADHClientPreMasterSecret, info)
    TANForServer = AEAD_ENC(HADHClientTrafficKey, TAN, tanAD)
]

// ClientResponse
HealthAuthority -> Server: HADHClientParam, TANForServer

principal Server[
    knows public HAsalt, info, tanAD
    HADHServerPreMasterSecret = HADHClientParam^HADHServerSecret
    HADHServerTrafficKey = HKDF(HAsalt, HADHServerPreMasterSecret, info)
    ServerTAN = AEAD_DEC(HADHServerTrafficKey, TANForServer, tanAD)
]

// UserI establishes TLS, requests nonce

principal UserI[
    generates UIClientRandom
]

// ClientHello
UserI -> Server: UIClientRandom

principal Server[
    generates UIServerRandom
    generates UIDHServerSecret
    UIDHServerParam = G^UIDHServerSecret
]

// ServerHello
Server -> UserI: UIServerRandom, UIDHServerParam, ServerPublicKey, ServerCert

principal UserI[
    knows public UIsalt, info, tan2AD
    UIServerOK = SIGNVERIF(CAPublicKey, ServerPublicKey, ServerCert)?
    generates UIDHClientSecret
    UIDHClientParam = G^UIDHClientSecret
    UIDHClientPreMasterSecret = UIDHServerParam^UIDHClientSecret
    UIDHClientTrafficKey = HKDF(UIsalt, UIDHClientPreMasterSecret, info)
    UserTANForServer = AEAD_ENC(UIDHClientTrafficKey, UserTAN, tan2AD)
]

// ClientResponse
UserI -> Server: UIDHClientParam, UserTANForServer

principal Server[
    knows public UIsalt, tan2AD, nonceAD
    UIDHServerPreMasterSecret = UIDHClientParam^UIDHServerSecret
    UIDHServerTrafficKey = HKDF(UIsalt, UIDHServerPreMasterSecret, info)
    ServerUserTAN = AEAD_DEC(UIDHServerTrafficKey, UserTANForServer, tan2AD)
    TANMatches = ASSERT(ServerUserTAN, ServerTAN)?
    generates ServerNonce
    NonceForUser = AEAD_ENC(UIDHServerTrafficKey, ServerNonce, nonceAD)
]

Server -> UserI: NonceForUser

principal UserI[
    knows public nonceAD, uploadAD, keyAD
    UserNonce = AEAD_DEC(UIDHClientTrafficKey, NonceForUser, nonceAD)
    knows private State
    StateAndNonce = CONCAT(State, UserNonce)
    generates StateAndNonceKey
    StateAndNonceEncrypted = ENC(StateAndNonceKey, StateAndNonce)
    StateAndNonceKeyEncrypted = ENC(UserIEncToken1, StateAndNonceKey)
    EncTokenUploadMessage = CONCAT(UserIEncTokenHash1, StateAndNonceKeyEncrypted,
         StateAndNonceEncrypted)
    EncTokenUploadMessageForServer = AEAD_ENC(UIDHClientTrafficKey,
        EncTokenUploadMessage, uploadAD)
    StateAndNonceKeyForServer = AEAD_ENC(UIDHClientTrafficKey, StateAndNonceKey,
        keyAD)
]

UserI -> Server: EncTokenUploadMessageForServer, StateAndNonceKeyForServer

principal Server[
    // verify nonce in upload message
    ServerEncTokenUploadMessage = AEAD_DEC(UIDHServerTrafficKey,
        EncTokenUploadMessageForServer, uploadAD)
    ServerStateAndNonceKey = AEAD_DEC(UIDHServerTrafficKey,
        StateAndNonceKeyForServer, keyAD)
    ServerEncTokenHash1, ServerStateAndNonceKeyEncrypted,
        ServerStateAndNonceEncrypted = SPLIT(ServerEncTokenUploadMessage)
    ServerStateAndNonce = DEC(ServerStateAndNonceKey,
        ServerStateAndNonceEncrypted)
    ServerState, ServerNonceFromUser = SPLIT(ServerStateAndNonce)
    ServerNonceMatches = ASSERT(ServerNonce, ServerNonceFromUser)?
]

// Download process: UserJ downloads list of encounter tokens

principal UserJ[
    generates UJClientRandom
]

// ClientHello
UserJ -> Server: UJClientRandom

principal Server[
    generates UJServerRandom
    generates UJDHServerSecret
    UJDHServerParam = G^UJDHServerSecret
]

// ServerHello
Server -> UserJ: UJServerRandom, UJDHServerParam, ServerPublicKey, ServerCert

```

```
principal UserJ[
    knows public UJsalt, info
    UJServerOK = SIGNVERIF(CAPublicKey, ServerPublicKey, ServerCert)?
    generates UJDHClientSecret
    UJDHClientParam = G^UJDHClientSecret
    UJDHClientPreMasterSecret = UJDHServerParam^UJDHClientSecret
    UJDHClientTrafficKey = HKDF(UJsalt, UJDHClientPreMasterSecret, info)
]

// ClientResponse
UserJ -> Server: UJDHClientParam

principal Server[
    knows public UJsalt, downloadAD
    UJDHServerPreMasterSecret = UJDHClientParam^UJDHServerSecret
    UJDHServerTrafficKey = HKDF(UJsalt, UJDHServerPreMasterSecret, info)
    EncTokenDownloadMessageForUserJ = AEAD_ENC(UJDHServerTrafficKey,
        ServerEncTokenUploadMessage, downloadAD)
]

Server -> UserJ: EncTokenDownloadMessageForUserJ

principal UserJ[
    knows public downloadAD
    EncTokenDownloadMessage = AEAD_DEC(UJDHClientTrafficKey,
        EncTokenDownloadMessageForUserJ, downloadAD)
    UserJEncTokenHash1Download, UserJStateAndNonceKeyEncrypted,
        UserJStateAndNonceEncrypted = SPLIT(EncTokenDownloadMessage)
    UserJEncTokenHashMatches = ASSERT(UserJEncTokenHash1Download,
        UserJEncTokenHash1)?
    UserJStateAndNonceKey = DEC(UserJStateAndNonceKeyEncrypted, UserJEncToken1)
    UserJStateAndNonce = DEC(UserJStateAndNonceEncrypted, UserJStateAndNonceKey)
]

queries[
    // 1: Encounter token is never released publicly (false)
    confidentiality? UserJEncToken1

    // 2: Encounter token established between users I and J is the same on both
        sides (true)
    equivalence? UserIEncToken1, UserJEncToken1

    // The following query did not finish in 1 week of runtime

    // 3: Attacker cannot upload encounter tokens or modify them (should be true)
    authentication? UserI -> Server: EncTokenUploadMessageForServer
]
```

Listing 2: tracecorona-3people-simple.vp

```
// Simple model:
// - Instead of approximating TLS by DH and signature verification, use pre-
    shared keys
// - Leave out the health authority
// - Leave out rolling IDs for now

attacker[active]

// Beforehand: Server and client pre-share keys (simulating TLS)
```

```

principal Server[
    knows private ServerUserIKey
    knows private ServerUserJKey
    knows private ServerUserKKey
]

// Phones generate keys, rolling IDs

principal UserI[
    knows private ServerUserIKey
    generates UserIPrivateKey1
    UserIPublicKey1 = G^UserIPrivateKey1
]

principal UserJ[
    knows private ServerUserJKey
    generates UserJPrivateKey1
    UserJPublicKey1 = G^UserJPrivateKey1
]

principal UserK[
    knows private ServerUserKKey
    generates UserKPrivateKey1
    UserKPublicKey1 = G^UserKPrivateKey1
    leaks UserKPrivateKey1
]

// UserJ begins App-to-App handshake with UserI using (EC)DH, this cannot be
    influenced

UserJ -> UserI: [UserJPublicKey1]
UserI -> UserJ: [UserIPublicKey1]

principal UserI[
    UserIEncToken1 = UserJPublicKey1^UserIPrivateKey1
    UserIEncTokenHash1 = HASH(UserIEncToken1)
]

principal UserJ[
    UserJEncToken1 = UserIPublicKey1^UserJPrivateKey1
    UserJEncTokenHash1 = HASH(UserJEncToken1)
]

// UserK (malicious) begins App-to-App handshake with UserK using (EC)DH
// UserJ private key is the same (same 30 minute time window)

phase[1]

UserK -> UserJ: UserKPublicKey1
UserJ -> UserK: UserJPublicKey1

principal UserJ[
    UserJEncToken2 = UserKPublicKey1^UserJPrivateKey1
    UserJEncTokenHash2 = HASH(UserJEncToken2)
]

principal UserK[
    UserKEncToken2 = UserJPublicKey1^UserKPrivateKey1
    UserKEncTokenHash2 = HASH(UserKEncToken2)
]

// UserI requests nonce
principal UserI [
    knows public tan2AD
    knows private UserITAN
    UserTANForServer = AEAD_ENC(ServerUserIKey, UserITAN, tan2AD)
]

// ClientResponse
UserI -> Server: UserTANForServer

principal Server[
    knows public nonceAD, tan2AD
    knows private UserITAN
    ServerUserTAN = AEAD_DEC(ServerUserIKey, UserTANForServer, tan2AD)?
    TANMatches = ASSERT(ServerUserTAN, UserITAN)?
    generates ServerNonce
    NonceForUser = AEAD_ENC(ServerUserIKey, ServerNonce, nonceAD)
]

Server -> UserI: NonceForUser

principal UserI[
    knows public nonceAD, uploadAD, keyAD
    UserNonce = AEAD_DEC(ServerUserIKey, NonceForUser, nonceAD)?
    knows private State
    StateAndNonce = CONCAT(State, UserNonce)
    generates StateAndNonceKey
    StateAndNonceEncrypted = ENC(StateAndNonceKey, StateAndNonce)
    StateAndNonceKeyEncrypted = ENC(UserIEncToken1, StateAndNonceKey)
    EncTokenUploadMessage = CONCAT(UserIEncTokenHash1, StateAndNonceKeyEncrypted,
         StateAndNonceEncrypted)
    EncTokenUploadMessageForServer = AEAD_ENC(ServerUserIKey,
        EncTokenUploadMessage, uploadAD)
    StateAndNonceKeyForServer = AEAD_ENC(ServerUserIKey, StateAndNonceKey, keyAD)
]

UserI -> Server: EncTokenUploadMessageForServer, StateAndNonceKeyForServer

principal Server[
    // verify nonce in upload message
    ServerEncTokenUploadMessage = AEAD_DEC(ServerUserIKey,
        EncTokenUploadMessageForServer, uploadAD)?
    ServerStateAndNonceKey = AEAD_DEC(ServerUserIKey, StateAndNonceKeyForServer,
        keyAD)?
    ServerEncTokenHash1, ServerStateAndNonceKeyEncrypted,
        ServerStateAndNonceEncrypted = SPLIT(ServerEncTokenUploadMessage)
    ServerStateAndNonce = DEC(ServerStateAndNonceKey,
        ServerStateAndNonceEncrypted)
    ServerState, ServerNonceFromUser = SPLIT(ServerStateAndNonce)
    ServerNonceMatches = ASSERT(ServerNonce, ServerNonceFromUser)?
]

// Download process: UserJ downloads list of encounter tokens

principal Server[
    knows public downloadAD
    EncTokenDownloadMessageForUserJ = AEAD_ENC(ServerUserJKey,
        ServerEncTokenUploadMessage, downloadAD)
]

Server -> UserJ: EncTokenDownloadMessageForUserJ

principal UserJ[
    knows public downloadAD
    UserJEncTokenDownloadMessage = AEAD_DEC(ServerUserJKey,
        EncTokenDownloadMessageForUserJ, downloadAD)?
    UserJEncTokenHash1Download, UserJStateAndNonceKeyEncrypted,
        UserJStateAndNonceEncrypted = SPLIT(UserJEncTokenDownloadMessage)
    UserJEncTokenHashMatches = ASSERT(UserJEncTokenHash1Download,
        UserJEncTokenHash1)?
    UserJStateAndNonceKey = DEC(UserJStateAndNonceKeyEncrypted, UserJEncToken1)
    UserJStateAndNonce = DEC(UserJStateAndNonceEncrypted, UserJStateAndNonceKey)
]

// Download process: UserK downloads list of encounter tokens

principal Server[
    knows public download2AD
    EncTokenDownloadMessageForUserK = AEAD_ENC(ServerUserKKey,
        ServerEncTokenUploadMessage, download2AD)
]

Server -> UserK: EncTokenDownloadMessageForUserK

principal UserK[
    knows public download2AD
    UserKEncTokenDownloadMessage = AEAD_DEC(ServerUserKKey,
        EncTokenDownloadMessageForUserK, download2AD)?
    UserKEncTokenHash1Download, UserKStateAndNonceKeyEncrypted,
        UserKStateAndNonceEncrypted = SPLIT(UserKEncTokenDownloadMessage)
    // No tokens match, so UserK cannot proceed
]

queries[
    // 1a: Encounter token is never released publicly (true)
    confidentiality? UserJEncToken1

    // 1b: Encounter token established between users I and J is the same on both
        sides (true)
    equivalence? UserIEncToken1, UserJEncToken1

    // 2: Private keys of users I and J are not leaked (true)
    confidentiality? UserIPrivateKey1
    confidentiality? UserJPrivateKey1

    // 3: Encounter token established between users J and K is the same on both
        sides (false)
    equivalence? UserJEncToken2, UserKEncToken2

    // 4: Attacker cannot upload encounter tokens or modify them (false)
    authentication? UserI -> Server: EncTokenUploadMessageForServer

    // 5: Attacker cannot decrypt state, only UserJ can (true)
    confidentiality? State
]
```

Listing 3: coronawarn.vp

```
// Simple model:
// - Instead of approximating TLS by DH and signature verification, use pre-
    shared keys
// - Leave out the health authority
// - Leave out rolling IDs for now

attacker[active]

// Beforehand: Server and client pre-share keys (simulating TLS)

principal UserI[
    generates UIDHClientSecret
    UIDHClientParam = G^UIDHClientSecret
]
UserI -> Server: [UIDHClientParam]

principal UserJ[
    generates UJDHClientSecret
    UJDHClientParam = G^UJDHClientSecret
]
UserJ -> Server: [UJDHClientParam]

principal UserK[
    generates UKDHClientSecret
    UKDHClientParam = G^UKDHClientSecret
]
UserK -> Server: [UKDHClientParam]

principal Server[
    generates UIDHServerSecret
    UIDHServerParam = G^UIDHServerSecret
    ServerUserIKey = UIDHClientParam^UIDHServerSecret
    generates UJDHServerSecret
    UJDHServerParam = G^UJDHServerSecret
    ServerUserJKey = UJDHClientParam^UJDHServerSecret
    generates UKDHServerSecret
    UKDHServerParam = G^UKDHServerSecret
    ServerUserKKey = UKDHClientParam^UKDHServerSecret
]
Server -> UserI: [UIDHServerParam]
Server -> UserJ: [UJDHServerParam]
Server -> UserK: [UKDHServerParam]

principal UserI[
    UserIServerKey = UIDHServerParam^UIDHClientSecret
]

principal UserJ[
    UserJServerKey = UJDHServerParam^UJDHClientSecret
]

principal UserK[
    UserKServerKey = UKDHServerParam^UKDHClientSecret
    leaks UserKServerKey
]


```

```
// Phones generate TEKs, RPIs

principal UserI[
    knows public ENRPIKString, ENAEMKString, PaddedData1, PaddedData2
    knows private UserIMetadata11, UserIMetadata12
    generates UserITEK1
    UserIRPIK1 = HKDF(nil, UserITEK1, ENRPIKString)
    UserIRPI11 = ENC(UserIRPIK1, PaddedData1)
    UserIAEMK1 = HKDF(nil, UserITEK1, ENAEMKString)
    UserIAEMK11 = CONCAT(UserIAEMK1, UserIRPI11) // extra step to simulate AES-
        CTR with RPI as IV
    UserIAEM11 = ENC(UserIAEMK11, UserIMetadata11)
    UserIBroadcast11 = CONCAT(UserIRPI11, UserIAEM11)
    UserIRPI12 = ENC(UserIRPIK1, PaddedData2)
    UserIAEMK12 = CONCAT(UserIAEMK1, UserIRPI12) // extra step to simulate AES-
        CTR with RPI as IV
    UserIAEM12 = ENC(UserIAEMK12, UserIMetadata12)
    UserIBroadcast12 = CONCAT(UserIRPI12, UserIAEM12)
]

principal UserJ[
    knows public ENRPIKString, ENAEMKString, PaddedData1
]

principal UserK[
    knows public ENRPIKString, ENAEMKString, PaddedData1
]

// UserJ and UserI exchange their RPIs, without MITM

UserI -> UserJ: [UserIBroadcast11]

principal UserJ [
    UserJUserIRPI11, UserJUserIAEM11 = SPLIT(UserIBroadcast11)
]

// UserK (malicious) and UserI exchange RPIs
// UserI has same TEK, but different RPI (same day, different time window)
UserI -> UserK: UserIBroadcast12

principal UserK[
    UserKUserIRPI12, UserKUserIAEM12 = SPLIT(UserIBroadcast12)
]

// UserI requests registration token
principal UserI[
    knows public tan2AD
    knows private UserITAN
    UserTANForServer = AEAD_ENC(UserIServerKey, UserITAN, tan2AD)
]

// ClientResponse
UserI -> Server: UserTANForServer

principal Server[
    knows public tan2AD, regTokenAD
    knows private UserITAN
    ServerUserTAN = AEAD_DEC(ServerUserIKey, UserTANForServer, tan2AD)?
    TANMatches = ASSERT(ServerUserTAN, UserITAN)?
    generates ServerRegToken
    RegTokenForUser = AEAD_ENC(ServerUserIKey, ServerRegToken, regTokenAD)
]

Server -> UserI: RegTokenForUser

principal UserI[
    knows public regToken2AD, regTokenAD
    UserRegToken = AEAD_DEC(UserIServerKey, RegTokenForUser, regTokenAD)?
    RegTokenForServer = AEAD_ENC(UserIServerKey, UserRegToken, regToken2AD)
]

UserI -> Server: RegTokenForServer

principal Server[
    knows public regToken2AD, testResultAD, positiveTestResult
    ServerUserRegToken = AEAD_DEC(ServerUserIKey, RegTokenForServer, regToken2AD)
        ?
    RegTokenMatches = ASSERT(ServerUserRegToken, ServerRegToken)?
    TestResultForUser = AEAD_ENC(ServerUserIKey, positiveTestResult, testResultAD
        )
]

Server -> UserI: TestResultForUser

principal UserI[
    knows public testResultAD, positiveTestResult, uploadAD
    UserTestResult = AEAD_DEC(UserIServerKey, TestResultForUser, testResultAD)?
    IsTestPositive = ASSERT(positiveTestResult, UserTestResult)?
    UploadMessage = CONCAT(UserRegToken, UserITEK1)
    UploadMessageForServer = AEAD_ENC(UserIServerKey, UploadMessage, uploadAD)
]

UserI -> Server: UploadMessageForServer

principal Server[
    knows public uploadAD
    // verify regToken in upload message
    ServerUploadMessage = AEAD_DEC(ServerUserIKey, UploadMessageForServer,
        uploadAD)?
    ServerUserRegToken2, ServerUserITEK1 = SPLIT(ServerUploadMessage)
    RegTokenMatches2 = ASSERT(ServerRegToken, ServerUserRegToken2)?
]

// Download process: UserJ downloads list of encounter tokens

principal Server[
    knows public downloadAD
    DownloadMessageForUserJ = AEAD_ENC(ServerUserJKey, ServerUserITEK1,
        downloadAD)
]

Server -> UserJ: DownloadMessageForUserJ

principal UserJ[
    knows public downloadAD
    UserJDownloadMessage = AEAD_DEC(UserJServerKey, DownloadMessageForUserJ,
        downloadAD)?
    UserJUserIRPIK1 = HKDF(nil, UserJDownloadMessage, ENRPIKString)
    UserJUserIRPI11Downloaded = ENC(UserJUserIRPIK1, PaddedData1)
    UserJKeyMatches = ASSERT(UserJUserIRPI11, UserJUserIRPI11Downloaded)?
    UserJUserIAEMK1 = HKDF(nil, UserJDownloadMessage, ENAEMKString)
    UserJUserIAEMK11 = CONCAT(UserJUserIAEMK1, UserJUserIRPI11) // extra step to
        simulate AES-CTR with RPI as IV
    UserJUserIMetadata = DEC(UserJUserIAEMK11, UserJUserIAEM11)
]

// Download process: UserK downloads list of encounter tokens

principal Server[
    knows public downloadAD
    DownloadMessageForUserK = AEAD_ENC(ServerUserKKey, ServerUserITEK1,
        downloadAD)
]

Server -> UserK: DownloadMessageForUserK

principal UserK[
    knows public downloadAD
    UserKDownloadMessage = AEAD_DEC(UserKServerKey, DownloadMessageForUserK,
        downloadAD)?
    leaks UserKDownloadMessage
    UserKUserIRPIK1 = HKDF(nil, UserKDownloadMessage, ENRPIKString)
    UserKUserIRPI12Downloaded = ENC(UserKUserIRPIK1, PaddedData2)
    UserKKeyMatches = ASSERT(UserKUserIRPI12, UserKUserIRPI12Downloaded)?
    UserKUserIAEMK1 = HKDF(nil, UserKDownloadMessage, ENAEMKString)
    UserKUserIAEMK12 = CONCAT(UserKUserIAEMK1, UserKUserIRPI12) // extra step to
        simulate AES-CTR with RPI as IV
    UserKUserIMetadata = DEC(UserKUserIAEMK12, UserKUserIAEM12)
]

queries[
    // 1: Attacker cannot decrypt metadata, only UserJ can (false)
    confidentiality? UserIMetadata12
    confidentiality? UserIMetadata11

    // 2a: Encounter metadata established between users J and K is the same on
        both sides (false)
    equivalence? UserIMetadata12, UserKUserIMetadata

    // 2b: Encounter metadata established between users I and J is the same on
        both sides (should be true, but is false)
    equivalence? UserIMetadata11, UserJUserIMetadata

    // 3: Attacker cannot upload diagnosis keys or modify them (should be true,
        but is false)
    authentication? UserI -> Server: UploadMessageForServer

    // The following queries did not finish in >1 month of runtime

    // 4a: Freshness of generated keys (should be true)
    freshness? UIDHClientSecret
    freshness? UIDHServerSecret
    freshness? UJDHClientSecret
    freshness? UJDHServerSecret
    freshness? UKDHClientSecret
    freshness? UKDHServerSecret
    freshness? UserIServerKey
    freshness? UserJServerKey
    freshness? UserKServerKey

    // 4b: If one knows one RPI, they cannot infer that others come from the same
         person (should be false)
    unlinkability? UserIRPI11, UserIRPI12
]
```

B. Full TraceCORONA UML package diagram

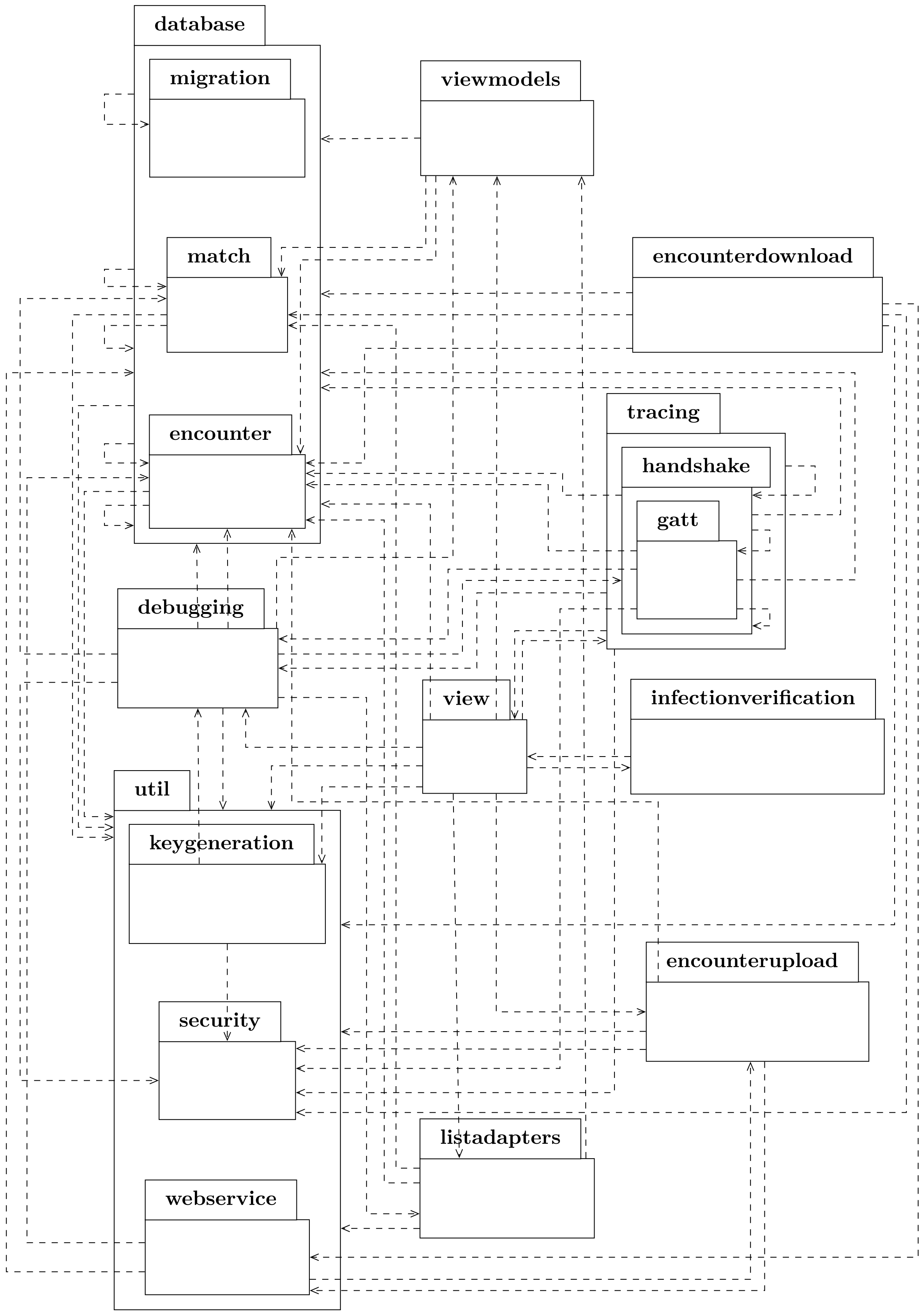


Figure B.1.: Full TraceCORONA package diagram, from a slightly older version than described in the thesis