ConTra Corona: Contact Tracing against the Coronavirus by Bridging the Centralized–Decentralized Divide for Stronger Privacy

Abstract: Contact tracing is one of the most important interventions to mitigate the spread of COVID-19/SARS-CoV-2. Smartphone-facilitated *digital contact tracing* may help to increase tracing capabilities as well as extend the coverage to those contacts one does not know in person. The emerging consensus is that a decentralized approach with local Bluetooth Low Energy (BLE) communication to detect contagion-relevant proximity, together with cryptographic protections, is necessary to guarantee the privacy of the users of such a system.

However, current decentralized protocols, including DP3T [31] and the protocol by Canetti, Trachtenberg, and Varia [8], do not sufficiently protect infected users from having their status revealed to their contacts, which may raise fear of stigmatization.

By taking a dual approach, we propose a new and practical solution with stronger privacy guarantees even against active adversaries. In particular, we solve the aforementioned problem with additional pseudorandom warning identities that are associated to the broadcasted public identity. This association is only known to a non-colluding dedicated server, which does not learn to whom the public identity belongs. Then, only these anonymous warning identities are published.

Moreover, our solution allows warned contacts to prove that they have been in contact with infected users, an important feature in times of restricted testing capacities. Among other additional security measures, we detail how the use of secret sharing can prevent the unnecessary and potentially panic-inducing warning of contacts that have only been around the infected person for a very brief time period.

Finally, to show our construction secure and privacy-preserving in a strong formal sense, we propose a modeling of the digital contact tracing functionality and the associated security and privacy guarantees in terms of an ideal functionality, and present an according simulator for our protocol.

Keywords: Digital Contact Tracing, Privacy, SARS-CoV-2, COVID-19, Active Security, Anonymity, Security Modeling, Ideal Functionality

DOI Editor to enter DOI

Received ..; revised ..; accepted ...

1 Introduction

One of the most important interventions to contain the SARS-CoV-2 pandemic is – besides the reduction of face-to-face encounters in general – the consequent isolation of infected persons, as well those who have been in close contact with them ("contacts") to break the chain of infections. However, tracing contacts manually (by interviews with infected persons) is not feasible when the number of infections is too high. Hence, more scalable and automated solutions are needed to safely relax restrictions of personal freedom imposed by a strict lockdown, without the risk of returning to a phase of exponential spread of infections. In contrast, *digital contact tracing* using off-the-shelf smartphones has been proposed as an alternative (or an additional measure) that is more scalable, does not depend on infected persons' ability to recall their location history during the days before the interview, and can even track contacts between strangers.

In many digital contact tracing protocols, e.g. [2, 10, 26, 8, 29, 31, 22, 6, 11, 4, 3], users' devices perform automatic proximity detection via short-distance wireless communication mechanisms, such as Bluetooth Low Energy (BLE), and jointly perform an ongoing cryptographic protocol which enables users to check whether they have been colocated with contagious users. However, naïve designs for digital contact tracing may pose a significant risk to users' privacy, as they process (and may expose) confidential information about users' location history, meeting history, and health condition.

This has sparked a considerable research effort for designing protocols for privacy-preserving contact tracing, most of which revolve around the following

idea: User's devices continuously broadcast ephemeral and short-lived pseudonyms[1] and record pseudonyms broadcast by close-by users. When a user is diagnosed with COVID-19, she submits either all the pseudonyms her device used while she was contagious or all the pseudonyms her device has recorded (during the same period) to a server. Users' devices either are actively notified by the server, or they regularly query the server for pseudonyms uploaded by infected users.

Some of the most prominent designs following this idea are the centralized PEPP-PT proposals ROBERT [24] and NTK [23], as well as the more decentralized DP3T [31] approach, at which also the Apple/Google-API proposal [3] aims at. While the centralized approaches of PEPP-PT do not guarantee users' privacy against the central server infrastructure [13, 14], the DP3T approach [31], as well as the similar protocol by Canetti, Trachtenberg, and Varia [8], expose the ephemeral pseudonyms of every infected user, which enables her contacts to learn about whether she is infected. The interested reader is referred to [18] for a fine grained comparison.

We argue that both, *protection against a centralized actor*, as well as *protection of infected users from being stigmatized for their status*, is of utmost importance for any real-world solution. By specifying a protocol that achieves both of these goals and detailing the corresponding (modular) design choices, we aim to contribute to the ongoing discussion on privacy-preserving digital contact tracing.

1.1 Contribution

We propose a new and improved protocol for privacy-preserving contact tracing, which enjoys the following main properties:

- Our protocol does not allow anything non-trivial[2] to be learned on who is infected, even towards contacts that are warned by their app. This is done by splitting the broadcasted identifiers into two unlinkable pseudorandom identifiers, where one is used for broadcasting, and the other for publication by the server, in case the broadcasted identifier is uploaded by an infected individual.
  Additionally, we discuss approaches to preventing Sybil attacks (where an attacker creates multiple accounts to observe which of them is warned), based on the literature on the topic. Such attacks were deemed to be inherent by [12, IR 1: Identify infected individuals]. We believe, however, that Sybil attacks can be effectively mitigated by existing solutions.
- Our protocol makes use of a strict server separation concept to mitigate the threat to users' privacy posed by data collection on centralized servers. In our protocol, the medical professional uploads signed and encrypted public identifiers (without learning them) to a dedicated "matching" server, which does a lookup of the respective registered secret identity, which is an encryption of an (also pseudorandom) "warning identity". These can then be decrypted by a dedicated warning server that publishes them after de-duplication. This separation does not lead to a significant increase of computation on the side of the smartphone.
  Note that the specialized servers may be distributed amongst well-known independent institutions.Thus, in order to compromise the server architecture to the full extent, multiple institutions would have to cooperate maliciously.
- Our protocol allows warned users to securely prove the fact that they have been warned, e.g., towards medical professionals that administer tests for COVID-19. Without this feature, anybody who is curious about their status but not at risk could get tested, e.g., by showing a screenshot of a warning from someone else's smartphone – which would be unacceptable in times of restricted testing resources.
- As far as possible, our protocol prevents active attackers from injecting false positive warnings and from suppressing warnings supposed to be generated by the protocol (*false negatives*).
- As of yet, no formal modeling of digital contact tracing in terms of an ideal functionality has been achieved. We alleviate this fact by proposing (to the best of our knowledge) the first ideal functionality for digital contact tracing, which makes the leakage to an attacker explicit. With this, it becomes clear, e.g., what knowledge about the social graph can be observed by an attacker of the scheme – thus enabling a proper debate about the appropriateness of BLE-based digital contact tracing in general.

Moreover, we identify the timing of Bluetooth beacons as a side-channel that can be exploited to link distinct public identifiers, and propose a concrete solution for

1 In order to save energy and maximize devices' battery life, these pseudonyms should be as short as possible (e.g. 128 bits).
2 Except for leakage, such as when a warned person has only been in contact with one other person.

adding timing jitter that is compatible with an update of identifiers.

1.2 Related Work

First note that there are far too many approaches for contact tracing to list them here fully. An overview of the related work and the several offered apps has been compiled in a collaborative effort and published at [27]. See also [28] for a discussion of some recent proposals of digital contact tracing. We focus here on the cryptographic discussion that has been ongoing on preprint servers, such as arXiv and the IACR eprint archive, and the git repositories of the most prominently discussed proposals of PEPP-PT [22] and DP3T [31], and focus on those that are most similar to our work.

First, let us note that early proposals, such as from the just-mentioned centralized PEPP-PT [22] project required that "No geolocation, no personal information or other data are logged that would allow the identification of the user. This anonymous proximity history cannot be viewed by anyone, not even the user of phone A." [22, Sect. 2]. However, the emerging consensus is that any app for the purpose of contact tracing should be transparent and open source, which makes it possible that a curious user can run a slightly modified application without any restriction on what to log. Hence, we believe that already everything that a user can observe during the protocol should not leak personal information about other users. This will exclude all more intransparent approaches from further consideration.

Canetti et al. [8] mention an extension of their protocol using private set intersection protocols in order to protect the health status of infected individuals. However, it is unclear how feasible such a solution is with regard to the computational load incurred on both, the smartphone and the server, cf. [15, P3].DP3T DP3T DP3T Whereas [31] accepts this issue as inherent by [12, IR 1: Identify infected individuals] and therefore does not further address possible countermeasures. Our protocol tackles this problem at its root and is augmented to obtain additional security and privacy guarantees, such as preventing the contacts of an infected individual to learn this status (assuming the proposed anti-Sybil protections are effective).

Chan et al. [10, Sect. 4.1] include a short discussion of protocols that upload observed identifiers in case of infections (as in our case), and propose a certain form of rerandomization of identifiers, albeit completely at the side of the smartphone. Hence, this approach puts a regular heavy computation cost on the user's device, and is likely not practical, as it has to query all possible identity rerandomizations from the server.

Bell et al. [4] propose two solutions for digital contact tracing, with the first of them also making use of a splitting of the role of the health care provider, and separate (non-colluding) government run servers.

There are several approaches that use public keys as identities, e.g. [11]. However, the maximum message length of BLE broadcasts does not permit sending entire public keys of asymmetric encryption schemes, cf. [15].

Besides BLE-based approaches, there are also proposals that use GPS traces of infected individuals to discover COVID-19 hot spots as well as colocation (albeit with a lower resolution), such as [5, 17]. However, there is a consensus that GPS-based approaches do not offer a a sufficient spatial resolution to estimate the distance between two participants with sufficient precision.

[19], and [9] (also a hybrid approach), broadcasts public key and computes Diffie-Hellman shared secret upon receiving a broadcast. See also [33] for a general discussion on hybrid approaches.

1.3 Outline

We define our informal security model for BLE-based contact tracing in Section 2, the formal version is given in Appendix A. We specify a first basic protocol that illustrates our main idea in Section 3. For this protocol, Section 4 proposes a number of useful extensions in a modular way, which are applied to obtain our overall protocol presented in Section 5. An informal security and privacy analysis of the protocol follows in Section 6. We conclude in Section 7.

2 Security Model

This section presents our (informal) security model. A formal, simulation-based definition of security for privacy-preserving contact tracing protocols will be given in Appendix A.

Our main goals are *privacy* (i.e. limiting disclosure of information about participating individuals) and *security* (i.e. limiting malicious users' abilities to produce false positives and false negatives). For privacy, we consider the following types of private information:

– where users have been at which point in time (i.e. their location history),
– whom they have met (and when and where),
– whether a user has been infected with SARS-CoV-2,
– whether a user has received a warning because she was colocated with an infected user.

We refer the interested reader to [20] for an extensive systematization of different privacy desiderata.

Participants of our protocol assume one or more of the following roles:

Users, who have installed the contact tracing application on their mobile phone in order to receive a warning if they have been in close proximity to one or more infected persons, and who want to warn other users in case they are infected themselves.

Medical Professionals, who administer tests for the SARS-CoV-2 virus and medical treatment as appropriate.

(Centralized) Servers, operated by health authorities or other responsible organizations.

We assume all communication between the servers uses secure (i.e. confidential and authenticated) channels. We assume the attacker cannot permanently prevent communication between other parties.

We assume centralized servers and medical professionals to be trusted with respect to security (but only partially trusted regarding privacy), i.e. we assume they will not engage in actions undermining the availability or correct functioning of the protocol. This entails they will not deviate from the prescribed protocol in order to cause fake warnings or suppress due ones. Furthermore, we trust medical professionals to not disclose data regarding the users who are under their care, as is their duty under standard medical confidentiality. The centralized servers are assumed not to cooperate in breaking users' privacy, cf. Section 6.1.2. Non-medical users are not trusted to adhere to the protocol.

When considering the distance of corrupted users to another user A, we use a slightly relaxed notion of "proximity": We consider the attacker to be close to A iff she is able to communicate with A's mobile phone via the BLE protocol (potentially using dedicated equipment such as high-gain antennas). This includes situations where the attacker only communicates with A's device via relays that are in "proximity" to A, as considered by [25].

Given this model, we strive to attain the following important security and privacy goals. (See below for a justification of the limitations in our goals.)

1. A coalition of malicious/corrupted users must not learn private information about uncorrupted users, except for information that can be observed via other means (e.g. by one malicious user being in close proximity to the victim).
2. The same holds for medical professionals, except that medical professionals may be aware of the health conditions of users under their care.
3. Even if all of the central servers are compromised and colluding with malicious users, the privacy impact for the users must be as low as possible.
4. Users should be able to prove they have been in proximity with an infected individual and received a warning by the application.
5. A coalition of malicious users (not including medical professionals) must not be able produce a *false negative* result, i.e. they must not be able to cause a situation where an uncorrupted user who has been in colocation with an uncorrupted infected user (close enough, for long enough) does not receive a warning (or cannot prove possession of a warning), unless a corrupted user has been suppressing communication between the victim and the infected person during the colocation.
6. (Coalitions of) malicious users (not including medical professionals) must not be able to cause a warning to be delivered to an uncorrupted user, unless the victim has indeed been colocated with an infected user or the attacker has been colocated both with an infected person and with the victim (*false positive* regarding an uncorrupted user).
7. (Coalitions of) malicious users (again, not including medical professionals) must not be able to prove possession of a warning, unless one of the malicious users has in fact been in colocation with an infected user (*false positive* regarding a corrupted user).

We do not consider adaptive corruptions, i.e. users are either honest or corrupted, but this does not change during the protocol execution. Dealing with adaptive corruptions is out-of-scope for this paper. We do not distinguish between "the attacker" and corrupted, malicious, or compromised parties.

We believe the limitations mentioned in the above goals are somewhat fundamental for an app that is based on tracking colocation:

– Regarding goals 1 and 2, observe that corrupted users can always learn information about other users by other means. For example, a corrupted medical professional who is administering tests or treatment to a potentially infected user will obviously be aware of the user's medical situation, and hence know whether the user is infected. We consider such information leakage inevitable.
– Medical professionals may always be able to cause suppression of a due warning or delivery of an undue warning by tampering with the test procedure. Again, this seems inevitable, so we only consider corrupted users not being medical professionals for goals 5 to 7.
– If an infected user is colocated with a corrupted user, the corrupted user can always simply choose to ignore a warning delivered to her (and/or uninstall the application and delete all associated data). Thus, it is unnecessary to provide guarantees of delivery of warnings to corrupted users in goal 5.
– If an infected, corrupted user wants to suppress warnings being delivered to colocated uncorrupted users, she can simply uninstall or deactivate the application. This limitation is reflected in goal 5.
– We do not fully address the issue of replay/relay attacks as discussed by Vaudenay [32] and [25]. In such an attack, a corrupted user records broadcast messages sent at one time and place and replays them at another time and/or place. This may lead contact tracing applications to register an encounter between users A, B who have not actually been in contact with one another, and hence lead to a false positive warning if A is infected. Thus, we only aim to provide security against false positives when the attacker is not close to both A and B (see goal 6).
We believe that any contact tracing protocol which is strictly based on non-interactive sending and receiving of local broadcasts without considering the actual time and location will be vulnerable to such an attack. As proposed by Vaudenay [32], executing an interactive protocol between two users having an encounter may provide a way to achieve better security regarding these attacks.
– If a user (uncorrupted or corrupted) has been colocated with an infected person, it is legitimate for the user to receive a warning, and to be able to prove "possession" of this warning. Goal 7 allows corrupted users to "exchange"/"trade" a received warning among them. Even if there were some cryptographic mechanism binding a warning to a device, corrupted users could still simply exchange their devices in order to "trade" a warning.

3 Basic (Preliminary) Protocol

As a starting point for the remainder of this work we propose the following protocol. H is a hash function, where $\mathsf{H}(k\|x)$ is assumed to be a pseudorandom function (PRF) with key $k \in \{0,1\}^n$ evaluated on input x.

Generation of "Random" Identities. For every time period t, the device generates a random key $k_t \leftarrow_{\$} \{0,1\}^n$, and computes $\mathsf{sid}_t := \mathsf{H}(k_t\|0)$ and $\mathsf{pid}_t := \mathsf{H}(k_t\|1)$, stores them, and (anonymously) uploads k_t to the central server, who recomputes $\mathsf{sid}_t, \mathsf{pid}_t$ in the same way. Both parties store $(\mathsf{sid}_t, \mathsf{pid}_t)$.

Broadcasting. During each time period t, the device repeatedly broadcasts pid_t. When it receives a broadcast value pid' from someone else, it stores $(\mathsf{date}, \mathsf{pid}')$, where date is the current date. Every day, the device deletes all tuples $(\mathsf{date}, \mathsf{pid}')$ where date is three weeks ago or older.

Warning co-located users. When a user is positively tested for SARS-CoV-2, (or the medical personnel believes the user to be most likely infected), the medical personnel estimates the first day sdate during which the user probably was infective, taking into consideration the incubation period of SARS-CoV-2 and the time it takes for an infected person to become contagious her-/himself.
Afterwards, one extracts a list of all recorded pid' from the infected user's device, where the associated date is no earlier than sdate. The user hands this list to the medical personnel, who forward the data to the health authorities, who finally upload this list to the server. (This chain of course needs to be authenticated.)
The central server infrastructure maintains a list of all pid' reported as having had contact with an infected person. Furthermore, the server has a list of $(\mathsf{sid}, \mathsf{pid})$ tuples uploaded by all users.
For each pid' reported as potentially infected, the server looks up the respective sid in his database of $(\mathsf{sid}, \mathsf{pid})$ tuples and marks the respective sid as potentially infected.
Either the server publishes a list of all potentially infected sids in regular intervals (signed by the server/the health authorities), or the server allows

users to query for a given sids, answering whether the sid has been marked as potentially infected.

This protocol illustrates our idea of separating the broadcast public identities pid from the secret identities sid which are published as warnings to their owners. However, this protocol still falls short of the privacy and security goals (see Section 2) that we are trying to achieve, since a user can link the sid being published as a warning to the time and location the corresponding pid was broadcast. Thus, the user knows when and where she met the infected person and might be able to deduce the infected user's identity.

We discuss the shortcomings and propose various extensions solving these issues in Section 4. We present our full "combined protocol", which encompasses some of these extensions, in Section 5.

4 Core Security Mechanisms

The simple protocol described above does not meet our requirements regarding security and privacy. This section introduces several improvements to the basic protocol.

4.1 Reusing the Secret Identifiers

In the basic protocol described above, users receiving a warning can immediately observe which of their secret identities sid was published. By correlating this information with the knowledge on when they used which public identity pid, they can learn at which time they have met an infected person, which poses a threat to the infected person's privacy. Note that the DP3T protocol [31] and the scheme by Canetti, Trachtenberg, and Varia [8] succumb to analogous problems.

To mitigate this risk, we propose to associate the same secret identity sid with many public identities pid. Care must be taken to make sure dishonest users must follow the protocol, we modify the process of deriving sid and pid values. Instead of choosing sid and pid as in the basic protocol, the user generates a single random key, now called *warning identifier*, once for a longer interval, e.g. one day. We propose the following concrete solution: A user generates a random warning identifier $\mathsf{wid} \leftarrow\!\!\$\ \{0,1\}^n$ per day, and encrypts it with the server's public key pk to obtain $\mathsf{sid} := \mathsf{Enc}_{\mathsf{pk}}(\mathsf{wid})$. For each shorter time period i, the user generates a rerandomization sid'_i of sid, where the randomness is derived from a PRG,and computes $\mathsf{pid}_i := \mathsf{H}(\mathsf{sid}_i)$. The user uploads sid and the PRG seed to the server, who performs the same rerandomization, obtaining the same sid'_i and pid_i values. The user broadcasts the pid_i in random order during the day.

Note that there is a fundamental trade-off to be made here: On the one hand, users should be roughly aware of when they have been in contact with an infected person, so that they can self-quarantine for an appropriate period. Moreover, if they start to show symptoms of COVID-19 in the following days, knowing the time of infection can help medical personnel to estimate when they have been contagious more precisely, and thus enable them to give more precise warnings to other users. Additionally, switching sid values with a high frequency restricts the number of pid values that can be linked in case of a server compromise. On the other hand, if users can determine the time of contact with an infected person exactly, they may be able to deduce the identity of the infected user. In our full protocol (see Section 5), we compromise by informing users about the *day* they have been in contact with an infected user.

4.2 Splitting-Up the Server into a Pipeline

The change introduced in Section 4.1 allows to split the process of warning co-located users into three tasks for three non-colluding servers, the *submission server*, the *matching server*, and the *warning server*. This eliminates the single point of failure a single server would constitute. See Section 6.1.2 for a privacy analysis regarding compromised servers.

- The *submission server* collects the uploaded secret and public identifiers from different users (more precisely, it receives sid and the seed for the PRG) and then computes the $(\mathsf{sid}'_i, \mathsf{pid}_i)$ pairs using the PRG with the given seed. It rerandomizes the sid'_i values another time with fresh, non-reproducible randomness (obtaining sid''_i), and stores $(\mathsf{sid}''_i, \mathsf{pid})$ for a short period of time. When the submission server has accumulated submissions by a sufficient number of clients, it shuffles them and then sends them to the matching server.
- The *matching server* collects the $(\mathsf{sid}''_i, \mathsf{pid}_i)$ pairs and stores them. Upon receiving the pids recorded by the devices of infected users, the matching server looks up the respective sid''_is of all potentially infected users and sends them to the *warning server*.

– The *warning server* decrypts sid_i'' to recover $\mathsf{wid} := \mathsf{Dec}_{\mathsf{sk}}(\mathsf{sid}_i'')$ for all potentially infected users and publishes a deduplicated list of warning ids.

4.3 Hiding Information from Medical Personnel

In the basic protocol from Section 3, the user unveils all public identities of every meeting recorded by the application to her medical personnel, who forwards it to the matching server. This puts the user's privacy at an unnecessary risk, since the medical personnel does not need learn about the user's meetings. To mitigate this issue, the application can simply encrypt the list of public identities with a public key of the matching server.[3]

4.4 Anonymous Communication Channels

When a user uploads her $(\mathsf{sid}, \mathsf{pid})$ pairs to the centralized servers, the servers can easily link these pairs with communication metadata (such as the user's IP address), which might be used to ultimately link these pairs to a specific individual. We therefore propose to use an anonymous communication channel for the submission of the $(\mathsf{sid}, \mathsf{pid})$ pairs. In practice, one might employ the TOR onion routing network [30] or send the submission via other publicly available proxies.

4.5 Using Secret Sharing to Enforce a Lower Bound on Contact Time

The DP3T document [31] proposes splitting the broadcasted identifiers with a secret sharing scheme to ensure that malicious users cannot record identifiers that they observe for less than a specified period of time (e.g. 15 minutes). However, it does not specify how one would rotate such shared identifiers if one wishes to switch to the next public identifier. Just stopping with one set of shares and starting the next set of shares (of a different public identifier) would prevent recording of contact if the contact happens during such an identity rotation.

To solve this issue, we propose to broadcast multiple public identities in parallel with overlapping intervals. As an example we could use a 15-out-of-30 secret sharing scheme and always broadcast two identities, in such a way that the new identity starts to be broadcast when the last identity has already had 15 shares broadcast. That way every contiguous interval of 15 minutes contains enough shares of one identity to be able to reconstruct the identity.

Additionally, care has to be taken that an observer needs to know which beacons belong to the same shared identifier, in order to choose the right shares to combine.

A per-identity marker can be incorporated into the payload. It needs to be long enough to make local collisions unlikely. In this situation the Bluetooth hardware address should be rotated once per beacon to not provide any unnecessary linkability between multiple identities.

4.6 Broadcast Timing Side Channel

If the application sends broadcasts in strict, exact intervals, an attacker might be able to link the two public identities by observing the offset of the broadcast times to her own clock. For example, if an application sends a broadcast in exact intervals of one minute and the attacker can observe that one device is continuously broadcasting whenever the attacker's clock is 10 seconds into the current minute, the attacker may be able to link several broadcasts to the same device even if the public identities being broadcast have changed in between. This may be used to link public identities both if they are used in direct succession, and if the attacker did not observe any broadcasts for a longer period of time.

To mitigate this attack, we propose to add random jitter to the starting point for broadcasting identities. When applying jitter, care has to be taken to add a few more shares to each identity to still ensure that the identity can be reconstructed from any 15 minute interval. When broadcasting the identity as a single beacon the jitter adds uncertainty to the observed exposure times. The two variants, namely absolute or relative jitter, both have their advantages and disadvantages. For ease of exposition, we describe the relative jitter in the following.

When applying relative jitter, one can think of the jitter as a random pause time between broadcasting identities. Using the notation from Variant 1, the user would start to send identity pid_i at $i \cdot \delta + \sum_{j=0}^{i} \Delta_j$. This way the jitter accumulates over time, and after a long enough period without observation the starting point for broadcasting identities will appear to be random.

3 This still surrenders the approximate length of the list to the medical personnel. Future work might consider further improvements in order to mitigate this remaining leakage.

As an example, consider 15-out-of-45 secret sharing, with every share being broadcast in 1-minute intervals. When a broadcast is started a random time between 15 and 30 minutes is chosen uniformly at random and after this delay the next ID-broadcast is started. Note that with this change two or three identities are being used simultaneously at every point in time. This ensures that in any 15 minute interval there is at least one public identifier broadcast completely covering the interval. Additionally this jitter accumulates very quickly to destroy the linkability of different broadcasted IDs.

4.7 Proving a Warning

In order to enable a user to prove to a third party a warning has been issued for her, her warning identity wid could be chosen as a Pedersen commitment to a representation rid of her real identity (e.g. her name), i.e. $\mathsf{rid} := g^u \cdot h^{\mathsf{rid}}$, where g, h are distinct generators of a group $\mathbb{G}$ (such that the discrete logarithm between g and h is not known) and u is a random number between zero (including) and the group order (excluding).

When a user wants to prove she has received a warning (e.g. to a medical professional), she discloses the respective warning identity wid she has received a warning for and her real identity rid, and issues a zero-knowledge proof-of-knowledge showing she could open the commitment to rid. The receiver verifies the proof, verifies the user's real identity and queries the warning server to check that wid is at-risk.

4.8 Protecting from Encounter-wise Warning Identities and Sybil Attacks

If the extension from Section 4.2 is applied, one might additionally want to prevent users from switching their warning identity wid too quickly, because of the following attack:

An attacker being able to upload an unlimited number of sid values to the submission server may be able to perform an attack similar to the *Paparazzi Attack* described by [32], as follows: After each encounter with another participant, the adversary records the time of the encounter, whom she has met, and which pids she sent during that time period. Then, the attacker switches to a pid value representing another warning identity. This way, when one of her warning identities wid is published by the warning server, the attacker can link wid to the encounter, and thus possibly the identity of the infected person.

Preventing this type of attack requires limiting uploads of sids to one identity per app instance per day. However, an attacker might try to bypass this restriction by running a *Sybil attack*, i.e. creating multiple (seemingly) independent app instances.

A defense strategy is to force the adversary to invest additional resources for spawning Sybil instances. We bind each app instance to a phone number during a registration process using an SMS challenge. (Note that this approach does not prevent an attacker from performing a Sybil attack on lower scale, as the attacker might own multiple phone numbers.)

4.8.1 Anonymity while Binding to Phone Numbers

Binding an app to an identifiable resource (such as a valid phone number) endangers the user's anonymity, since the server might store the data linking resource to the app instance. In order to achieve rate limiting without disclosing the link between uploaded sids and an identifiable resource, we propose using the periodic n-times anonymous authentication scheme from [7] or related schemes offering this functionality. In our setting, we choose $n = 1$ and a time period of one day, i.e. the user can obtain one "e-token" per day to upload a new sid (and PRG seed) to the submission server. The submission server validates the "e-tokens" and only accepts submissions with valid tokens while checking for double-spending. The token dispenser is then issued to the user during a registration process, which uses, e.g., remotely verifiable electronic ID cards or phone numbers that are verified via SMS challenges.

5 Our Contact-Tracing Protocol

We now describe the protocol that results the inclusion of our previously developed core security mechanism.

Let n denote the security parameter, $\mathbb{G}$ be a group of prime order q such that the decisional Diffie-Hellman problem in $\mathbb{G}$ is intractable, and let g, h be generators of $\mathbb{G}$. We assume two secure public key encryption schemes $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$: One of them having message space $\mathcal{M} = \mathbb{G}$ and rerandomizable ciphertexts, and one of them having message space $\mathcal{M} = \{0,1\}^*$. (We propose standard ElGamal and a standard hybrid encryption scheme for instantiation, respectively.) Let PRG be

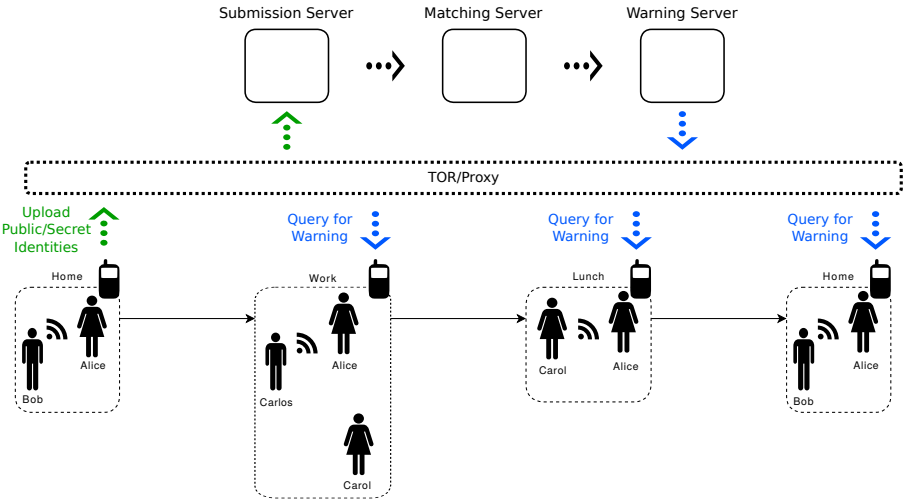

Fig. 1. Overview of the application's infrastructure. The figure depicts different possible scenarios during the day: In the morning, Alice uploads her daily public/secret identities to the submission server, and periodically queries the warning server for warnings. Throughout the day, while she is in proximity to Bob, Carlos and Carol, the application automatically exchanges public identifiers with their phones. The figure layout is inspired by [16].

a secure pseudorandom generator, and H be a collision-resistant hash function.

Each device continuously and randomly partitions time into overlapping intervals. Whenever one interval begins (say, at time t_0), the application chooses a random time difference Δ (between 15 and 30 minutes, with sub-second precision) and the next interval will begin at $t_0 + \Delta$. Each interval has a duration of 45 minutes. Thus, each point in time belongs to either two or three intervals, two successive intervals overlap by at least 15 minutes, and there are at most $24 \times \frac{60}{15} = 96$ beginnings of an interval in each day. We note that devices sample this partitioning independently of each other.

Server Setup. The matching server and the warning server each generate a key-pair for a public-key encryption scheme: The warning server for the rerandomizable scheme, the matching server for the other one. The public keys are published in a way users can retrieve them in an authenticated fashion.

App Setup. When the proximity tracing software is first installed on a user's device, the user enters her real identity rid, such as her name. (This information will only be shared with medical professionals treating her.) To avoid fears, the application should present an understandable explanation of why this is necessary (cf. Section 4.7). Additionally, for anti-Sybil measures as described in Section 4.8, the application proves possession of a phone number (e.g. via an SMS challenge) and obtains a e-token dispenser.

Creating Secret Warning Identifiers. For each day, the application generates a *warning identifier* wid as a Pedersen commitment [21] to the user's real identity. (That is, wid is computed as $\mathsf{wid} := g^u h^{\mathsf{rid}}$, where $u \leftarrow_\$ \mathbb{Z}_q$. We assume rid is implicitly mapped to $\mathbb{Z}_q$ in a collision resistant manner.) The application stores the unveiling information u for later use, deleting it after four weeks.[4]

Deriving Public Identities. For each warning identifier wid, the application computes $\mathsf{sid} := \mathsf{Enc}(\mathsf{pk}, \mathsf{wid})$, where Enc is the encryption algorithm of a rerandomizable, IND-CPA-secure public-key encryption scheme, and pk is the warning server's public key. Additionally, the application chooses a random $\mathsf{seed} \leftarrow_\$ \{0,1\}^n$ (*rerandomization seed*) per warning identity.

The application (interactively) presents an e-token τ to the submission server via an anonymous channel (e.g. via the TOR network), and uploads $(\mathsf{sid}, \mathsf{seed})$ to the submission server via the same channel. If the e-token is invalid (or the server detects double-spending of this e-token), the server refuses to accept $(\mathsf{sid}, \mathsf{seed})$. Both the submission server and the application compute $24 \times 4 = 96$ rerandomization

4 If some user A has been in contact with an infected user B during the day of validity of the respective warning identity, and even if B takes three weeks to show symptoms and have a positive test result, then A will be able to prove "ownership" of the respective warning for another week, which is sufficient time for her to get tested herself.

values $r_1, \ldots, r_{96} = \mathsf{PRG}(\mathsf{seed})$, and rerandomize sid using these values, obtaining $\mathsf{sid}'_i := \mathsf{ReRand}(\mathsf{sid}; r_i)$ for $i \in \{1, \ldots, 96\}$. The ephemeral public identities of the user are defined as $\mathsf{pid}_i := \mathsf{H}(\mathsf{sid}'_i)$ for all i.

The application saves the public identities for broadcasting during the day of validity of wid.

The submission server rerandomizes all the sid'_i values another time (using non-reproducable randomness), obtaining $\mathsf{sid}''_i := \mathsf{ReRand}(\mathsf{sid}'_i)$, and saves a list of the $(\mathsf{sid}''_i, \mathsf{pid}_i)$ tuples. When the submission server has accumulated a sufficiently large list of such tuples, originating from sufficiently many submissions, it shuffles the list and forwards all tuples to the matching server and clears the list afterwards.

The matching server maintains a list of all tuples it has received from the submission server, deleting each tuple after three weeks.[5]

Broadcasting Public Identities. For each time interval i, the application randomly chooses one of the public identities pid precomputed for the current day (but not used so far), computes a 15-out-of-45 secret sharing $s_1, \ldots, s_{45} = \mathsf{Share}(\mathsf{pid}_i)$, and selects a random identifier m. (m may be used as the Bluetooth MAC address if possible.)

During the respective interval, the application broadcasts the shares s_j (one at a time, with one minute between the broadcasts) together with the random identifier m.[6]

Receiving Broadcasts by other Users. The application continuously listens for broadcast messages by other users and maintains a database of these. When it receives a new broadcast (m', s'), the application checks if the database contains another broadcast value with the same random identifier m'. If it does, and the previous broadcast is less than (approximately) 60 seconds ago, the newly received message is discarded. Otherwise, the application temporarily saves the received broadcast tuple in its database. All database entries in this database are deleted after at most 45 minutes.

When the application has accumulated 15 broadcasts (m', s'_j) with the same random identifier m', it recombines the shares s'_j to recover the public identity pid' that was shared by the remote application, and records the occurrence of a meeting of its user with the user having the public identifier pid' at the current time. The information about this meeting is stored for the retention period, i.e. 21 days, and deleted afterwards.

Sending a Warning. When a user is tested positively for SARS-CoV-2 by medical personnel or the medical personnel considers an infection sufficiently likely, the medical personnel estimates the first day sdate during which the user was probably contagious.

The user instructs the application to collect a list of all meetings she has had from sdate until the present, and the respective list of public identities pid'. She encrypts the list of public identities using the public key of the matching server to obtain a ciphertext c.

The user sends c to the medical personnel (via an authenticated channel), who (directly or indirectly) forwards it to the matching server (again, via an authenticated channel, such that the matching server can verify c was sent from some authorized medical professional).

Centralized Processing of Warnings. When medical personnel submits a ciphertext c, the matching server decrypts the ciphertext to recover the list of public identities the application has recorded a meeting with.

The server looks up the corresponding secret identifiers sid in its database and sends the secret identifiers to the warning server.

The warning server decrypts the secret identifiers to recover the warning identifier wid contained in them, and regularly publishes a deduplicated list of all warning identifiers it has received during the last two weeks.

Retrieving Warnings. The application regularly fetches the list of published warning identifiers from the warning server (via an authenticated channel) and compares it with the list of warning identifiers it has used during the last 28 days itself.

If there is an overlap, it informs the user she has been in contact with an infected person on the day the warning identifier was used.

Proving Possession of a Warning. If the user reports to a hospital, asking to be tested for SARS-CoV-2, she surrenders her real identity rid and her warning identity wid to the medical personnel. Using a zero-knowledge proof (e.g., using the Fiat-Shamir heuristic), she shows her wid is a valid Pederson commitment to rid.

5 If some user A has been in contact with an infected user B who observes the respective pid, and even if B takes up to three weeks to show symptoms and have a positive test result, the data retention on the matching server is sufficient to deliver a warning to A.

6 Since intervals are overlapping such that any point in time belongs to two or three intervals, the user will be sending a broadcast every 20 to 30 seconds on average.

The medical personnel verifies the proof and verifies that wid has indeed been published by the warning server. (The medical personnel uses an authenticated channel for retrieving the list of warning identities from the warning server.)

This concludes the description of our protocol.

6 Security and Privacy Analysis

We now present a preliminary, informal analysis of the protocol described above regarding its security and privacy. As mentioned in Section 4.8 we leave the specific choice of countermeasures against Sybil attacks for future work. We discussed naïve but efficient solutions that thwart such attacks up to a certain degree, and suggested to use the anonymous e-token system by [7] for a sophisticated but more resource-consuming solution to this problem. With these solutions an attacker can only create a limited amount of Sibyls. From a security point of view, this situation is not different from the case where the attacker is a small group of malicious users. Therefore, we exclude excessive Sybil attacks from the security analysis.

6.1 Privacy

Note that, due to the security of the secret sharing scheme, observing a public identity requires being in proximity to the user's device for approximately 15 minutes. This restrains the attacker's ability to observe public identities at specific times and places in the first place.

For our privacy analysis, we assume corrupted users can link some public identities they directly observe to the real identities of the corresponding user, i.e. by accidentally meeting someone they know. This pessimistic approach yields a worst-case analysis regarding the information available to corrupted users.

A corrupted user may be able to modify the application (or use a completely different one) in order to collect additional data not typically collected. Corrupted users might also deviate from the protocol in order to obtain additional information.

6.1.1 Privacy in Case of Corrupted Participants

Multiple corrupted users may work together, combining their information. Thus, we end up with an adversary in possession of a large amount of recorded broadcasts with meta data on time and location. We conservatively assume an adversary can receive BLE traffic within the entire area where a targeted person may have moved and can link some of the broadcasts to the targeted person, by e.g., meeting her in person or utilizing video surveillance.

6.1.1.1 Hiding Multiple Exposures

The frequency of exposures, i.e., warnings, may allow the warned participant to re-identify the positively tested participant, who has uploaded the received public identities pids. In Section 4.2 we describe our approach of deduplication of warning wids before publishing. The deduplication prevents a user from learning how many of her public identities have been recorded by infected users, thus hiding the frequency of exposures from the warned participant.

6.1.1.2 Privacy of Positively Tested Participants

To capture the privacy risk of positively tested participants, note that the only difference in their protocol behavior is that they hand over an encrypted list of recorded pids of their contacts to the medical professional for an upload to the matching server. We assume that the hand over is done via a confidential channel and that the uploading happens without any reference to the users identity. Assuming the servers are uncorrupted, the only change in the attacker's view is the additional warning identities published on the warning server. Hence, this section only concerned about the privacy risk incurred by this extra information. However, the following discussion will argue that these warning identities are not linkable to any public identities broadcast by uncorrupted users (except under trivial conditions, e.g. that the user was in contact with only one other user during a wid's one-day lifetime), and hence do not pose an additional privacy risk. (In particular, this ensures confidentiality w.r.t. the user's infection status.)

As discussed above, we can assume the adversary is at most a small group of n colluding users (without additional Sybils). The one-day lifetime of these corrupted users' warning identities guarantees that no single user can (on their own) distinguish which of her encounters during the day caused her to receive a warning. (Note,

by working together and making use of group testing mechanisms, the malicious users' might be able to single out the infected person from a larger group of 2^n users. This attack is inherent in the desired functionality.) Thus, our protocol is only susceptible to *inherent* attacks.

6.1.1.3 Privacy of Warned Participants

Our protocol naturally protects the privacy of warned participants and their social graph as the published warning identity is computationally unlinkable to any information that can be recorded locally, thus it is only of use to the warned application. Specifically, each honestly generated warning identifier wid is a Pedersen commitment to the user's real identity. Since Pedersen commitments are perfectly hiding, the attacker cannot infer the user's real identity rid from wid, and also deciding whether some identities belong to the same user, is impossible. Thus, the warning identity wid does not help the attacker in breaking the users' privacy.

6.1.2 Privacy in the Case of Compromised Servers

This section presents a preliminary analysis of the privacy guarantees offered by our protocol if servers are compromised.

6.1.2.1 Linking Public Identities from the Same Day

If the submission server is compromised, the attacker will be able to link different public identities pid to the same secret sid, and hence can link the public identities the user is using on the same day. This situation only poses a privacy threat, if the attacker additionally has observed some of the targeted public identities pid, which requires colluding users. See Section 4.1 for a detailed discussion about this trade-off.

Similarly, if both the matching server and the warning server are corrupted, the attacker can decrypt the sid values stored by the matching server to recover the wid value, and hence again link public identities to the secret identities sid and the respective warning identity wid. We analyze how the additional capability of linking public identities capability may help an attacker in breaking user's privacy.

An attacker (including users cooperating to break others' privacy) may be able to link public identities to times and places where these identities have been broadcast. An attacker additionally having compromised the servers may therefore be able to re-identify a user at different times and places during the same day. Thus, an attacker may be able to observe parts of the user's location history and track a user for up to one day.

We stress that even if all servers are compromised, an attacker will not be able to link public identities used on different days (assuming the use of anonymous communication and no other leakage).

6.1.2.2 Contact Information of Infected Users

Information about encounters between users is stored strictly on the user's devices. Only the meeting history (more precisely: the list of encountered public identities, without times and places of meetings) of infected users is transmitted to the central servers.

If the attacker has compromised the matching server *and* is able to link public identities used on the same day (as in the previous scenario), the attacker might be able to infer repeated meetings of the infected user, i.e. she can learn how many encounters with the same persons the infected user's device has registered within each day. If the attacker has additionally observed some of the warned public identities at specific times and places, the attacker will also learn where and when (approximately) the encounter took place, and hence learn parts of the location history of the infected user as well as the warned users.

6.1.2.3 Warnings Issued

If the attacker has compromised the matching server, she can immediately observe the public identities of all users who have been colocated with infected users. If the attacker can additionally link a public identity to a specific individual, the attacker can conclude this person has received a warning. (Note that a similar attack is possible in the DP3T protocol [31], but even without compromising a server.) Linking a public identity to a specific individual will require learning the public identity in the first place, which (again) requires staying in proximity to the user for approximately 15 minutes.

6.2 Security

We now analyze an attacker's ability to cause false negatives or false positives. As stated above, we consider a coalition of malicious users, who may be deviating from the prescribed protocol. However, we assume medical

personnel as well as the central servers do not participate in such attacks, i.e. they follow the protocol.

6.2.1 Creating False Negatives

A false negative occurs when an uncorrupted user A has been in colocation with an uncorrupted infected user B and no corrupted user was present during the colocation, but either A does not receive a warning, or she cannot prove the possession of the warning to the medical personnel. A warning is issued by the warning server publishing the respective warning identity, which is a Pedersen commitment to A's real identity.

Observe that once B's device has recorded an encounter with A's public identity pid, corrupted users can no longer interfere with the delivery of the warning.

Thus, an attacker wanting to produce a false negative must prevent B's device from registering the encounter with A's public identity pid. This, however, requires the attacker being able to interfere with the local BLE communication between them, and thus to be in proximity to them while the encounter is taking place.

This shows our protocol achieves the required security guarantees regarding false negatives.

6.2.2 False Positives Regarding Honest Users

An honest user A is subject of a false positive if she has not been colocated with an infected user, but she nonetheless receives a warning. Our security goal is to prevent false positives, unless the attacker has been in proximity to both an infected user and A.

In order to cause a warning for A, an attacker must have the warning server publish one of her warning identifiers, i.e. one of A's public identifiers pid must be uploaded to the matching server by a medical professional, and hence an infected user B must present a list including A's pid.[7]

If B is uncorrupted, the attacker must trick B's device into registering an encounter with A's pid. B's device will register the encounter when having received sufficiently many shares of A's pid. Since the application discards shares with the same random identifier m if they are sent too quickly, the attacker needs to be in proximity with B for approximately 15 minutes. (If B is corrupted, this part of the attack can be skipped.)

In any case, the attacker needs to learn one of A's public identities. As argued in Section 6.1, this requires the attacker to be close to A for approximately 15 minutes, due to the secret sharing scheme employed.

This concludes our argument that producing a false positive for an honest user requires proximity both to the honest user and to an infected user.

6.2.3 False Positives Regarding Corrupted Users

We now analyze corrupted users' ability to prove possession of warnings. Since the medical personnel retrieves the list of warning identities from the warning server via an authenticated channel, the attacker can only prove possession of warnings regarding warning identities published by the warning server. Let wid be the warning identity the attacker wants to prove ownership of.

If wid was generated by an honest user, it is a Pedersen commitment to the real identity rid of the honest user created with a decommitment value u. Since honest users never share the unveiling information (not even with medical personnel), we consider it unlikely an attacker learns the value u.

Thus, proving the ownership of wid would require the attacker to present her real identity rid' and a zero-knowledge proof showing she knows a corresponding unveiling information u'. However, since the Pedersen commitment scheme is computationally binding (under the discrete logarithm assumption in $\mathbb{G}$), and if the zero-knowledge proof system is sound, the attacker will not be able to forge a proof.

If wid was generated by a corrupted user, the attacker may be able to prove ownership of the respective warning identity wid. In this case, however, the attacker will have to make sure one of the public identities pid derived from wid is reported to the matching server by medical personnel, and hence an infected user must hand out a list containing pid to the medical personnel.

If the infected user is corrupted herself, this is trivial. if the infected user is honest, causing her to output pid requires her device registering an encounter. Since the application is rate-limiting the reception of shares of public identities, this requires the attacker to stay in proximity with the infected user for approximately 15 minutes.

This concludes our discussion of the protocol's security properties. A more formal security discussion, including a simulator is given in Appendix A.

7 Since the public identities stored on the matching server are created as hash values of (rerandomizations of) A's sid, collisions between differing public identities are very unlikely.

7 Summary

We presented ConTra Corona, a new approach to achieve contact tracing in a privacy preserving manner. Leakage of private information that was previously thought inevitable turned out unnecessary after all. We protect it by decoupling the identities used for warning at-risk users from the information that is broadcast locally and can be observed by other users. However, our improvements introduce new challenges (e.g. our improvements reinforce the need for protection against Sybil attacks). To address these challenges, our approach requires some additional protective measures, and we highlighted how these can be implemented using existing techniques.

Moreover, we introduce protection against a side-channel assisted linking of different broadcasts by randomizing the starting points of broadcast blocks of secret shares. In order to reduce the required trust into the central server components, we described how the server's functions may be separated by distributing core functions to different organizations, which removes the single point of failure regarding privacy of a purely centralized contact tracing application. We argued that, even if all servers are compromised and colluding with malicious participants, our protocol still achieves almost the same privacy guarantees as previous works, such as [31]. Thus, in conclusion we argue that our protocol represents an overall improvement regarding security and privacy, while still being relatively practical.

References

[1] D. Achenbach et al. "Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework". In: *FC*. 2019.

[2] T. Altuwaiyan, M. Hadian, and X. Liang. "EPIC: Efficient Privacy-Preserving Contact Tracing for Infection Detection". In: *IEEE ICC*. 2018.

[3] Apple and Google. *Privacy-Preserving Contact Tracing*. 2020. URL: http://www.apple.com/covid19/contacttracing.

[4] J. Bell et al. *TraceSecure: Towards Privacy Preserving Contact Tracing*. Tech. rep. 2004.04059. ArXiv, 2020.

[5] A. Berke et al. *Assessing Disease Exposure Risk with Location Data: A Proposal for Cryptographic Preservation of Privacy*. Tech. rep. 2003.14412. ArXiv, 2020.

[6] S. Brack, L. Reichert, and B. Scheuermann. *Decentralized Contact Tracing Using a DHT and Blind Signatures*. 2020.

[7] J. Camenisch et al. "How to win the clonewars". In: *ACM CCS*. 2006.

[8] R. Canetti, A. Trachtenberg, and M. Varia. *Anonymous Collocation Discovery: Harnessing Privacy to Tame the Coronavirus*. Tech. rep. 2003.13670. ArXiv, 2020.

[9] C. Castelluccia et al. *DESIRE: A Third Way for a European Exposure Notification System*. 2020. URL: https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE.

[10] J. Chan et al. *PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing*. Tech. rep. 2004.03544. ArXiv, 2020.

[11] H. Cho, D. Ippolito, and Y. W. Yu. *Contact Tracing Mobile Apps for COVID-19: Privacy Considerations and Related Trade-offs*. Tech. rep. 2003.11511. ArXiv, 2020.

[12] DP-3T Project. *Privacy and Security Risk Evaluation of Digital Proximity Tracing Systems*. 2020. URL: https://github.com/DP-3T/documents/blob/master/Securityanalysis/Privacy%20and%20Security%20Attacks%20on%20Digital%20Proximity%20Tracing%20Systems.pdf.

[13] DP-3T Project. *Security and privacy analysis of the document 'PEPP-PT: Data Protection and Information Security Architecture'*. 2020. URL: https://github.com/DP-3T/documents/blob/master/Securityanalysis/PEPP-PT_%20Data%20Protection%20Architecture%20-%20Security%20and%20privacy%20analysis.pdf.

[14] DP-3T Project. *Security and privacy analysis of the document 'ROBERT: ROBust and privacy-presERving proximity Tracing'*. 2020. URL: https://github.com/DP-3T/documents/blob/master/Securityanalysis/ROBERT%20-%20Security%20and%20privacy%20analysis.pdf.

[15] DP3T Project. *FAQ: Decentralized Proximity Tracing*. 2020. URL: https://github.com/DP-3T/documents/blob/master/FAQ.md.

[16] L. Ferretti et al. "Quantifying SARS-CoV-2 transmission suggests epidemic control with digital contact tracing". In: *Science* (2020).

[17] J. K. Fitzsimons et al. *A note on blind contact tracing at scale with applications to the COVID-19 pandemic*. Tech. rep. 2004.05116. ArXiv, 2020.

[18] Fraunhofer AISEC. *Pandemic Contact Tracing Apps: DP-3T, PEPP-PT NTK, and ROBERT from a Privacy Perspective*. Tech. rep. 489. IACR, 2020.

[19] G. Garofalo et al. *Striking the Balance: Effective yet Privacy Friendly Contact Tracing*. Tech. rep. 559. IACR, 2020.

[20] C. Kuhn, M. Beck, and T. Strufe. *Covid Notions: Towards Formal Definitions – and Documented Understanding – of Privacy Goals and Claimed Protection in Proximity-Tracing Services*. Tech. rep. 2004.07723. ArXiv, 2020.

[21] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO*. 1991.

[22] PePP-PT e.V. *Pan-European Privacy-Preserving Proximity Tracing*. 2020. URL: https://www.pepp-pt.org/content.

[23] PePP-PT e.V. *PEPP-PT NTK High-Level Overview*. 2020. URL: https://github.com/pepp-pt/pepp-pt-documentation/blob/master/PEPP-PT-high-level-overview.pdf.

[24] PePP-PT e.V. *ROBust and privacy-presERving proximity Tracing protocol*. 2020. URL: https://github.com/ROBERT-proximity-tracing/documents.

[25] K. Pietrzak. *Delayed Authentication: Replay and Relay Attacks on DP-3T*. Tech. rep. 418. iacr, 2020.

[26] R. L. Rivest et al. *The PACT protocol specification*. 2020. URL: https://pact.mit.edu/wp-content/uploads/2020/04/The-PACT-protocol-specification-ver-0.1.pdf.

[27] stop-covid.tech, covid-watch.org, M. Ardron, et al. *Unified research on privacy-preserving contact tracing and exposure notification for COVID-19*. 2020. URL: https://bit.ly/virustrackertracker.

[28] Q. Tang. *Privacy-Preserving Contact Tracing: current solutions and open questions*. Tech. rep. 426. iacr, 2020.

[29] TCN Coalition. *A Global Coalition for Privacy-First Digital Contact Tracing Protocols to Fight COVID-19*. URL: https://tcn-coalition.org/.

[30] The Tor Project, Inc. *TOR Project*. URL: https://www.torproject.org/.

[31] C. Troncoso et al. *Decentralized Privacy-Preserving Proximity Tracing*. Tech. rep. 2005.12273. ArXiv, 2020.

[32] S. Vaudenay. *Analysis of DP3T*. Tech. rep. 399. IACR, 2020.

[33] S. Vaudenay. *Centralized or Decentralized? The Contact Tracing Dilemma*. Tech. rep. 531. IACR, 2020.

A Formal Security Notion

For a formal security notion we consider a scenario in which a Distinguisher $\mathcal{D}$ has to find out, if an experiment is conducted in a "real" world or an "ideal" world. In the "real" world, the protocol is executed and an attacker interferes with it. In the "ideal" world, the attacker is replaced by a Simulator $\mathcal{S}$ and all honest parties calculate their result via an ideal functionality $\mathcal{F}_{\mathrm{CT}}$.

In the "real" world, the protocol uses the assumed functionality $\mathcal{F}_{\mathrm{mat}}$ to model the interaction between participants. This functionality is used for local broadcast and to decide which participant receives a particular pid broadcast. The Distinguisher $\mathcal{D}$ uses $\mathcal{F}_{\mathrm{mat}}$ to set the physical scenario, determining which participants meet and which participants turn out to be infected.

Channels between the parties and functionalities or the servers are assumed to be confidential and authentic (in the fitting direction). To keep the complexity manageable, several assumptions have been made:

- The per-day uploads are synchronous. We assume that before any pid is broadcast, all parties have made their per-day upload. All parties, even corrupted ones only make one broadcast per epoch. (The distinguisher can emulate a single corrupted party making multiple broadcasts by using additional corrupted parties with similar/equal sets of recipients.)
- The Distinguisher $\mathcal{D}$ calls the participants one-by-one to perform their broadcast.
- Contacts happening on the day an infected person is uploading their list do not incur warnings.
- In order to model jamming attacks the distinguisher can emulate a suppression of broadcasts by leaving out edges within the neighborhood graph.

A.1 Ideal Protocol

We model the security in the ideal setting via an ideal contact-tracing functionality $\mathcal{F}_{\mathrm{CT}}$, which interacts with a simulator $\mathcal{S}$. For now, we assume that the servers are uncorrupted, and leave the formal modeling of server corruption as future work. However, note our more informal reasoning on security against server corruptions in Section 6.1.2.

It is important that $\mathcal{F}_{\mathrm{CT}}$ also explicitly models the leakage of a contact-tracing protocol fulfilling our security guarantees. For this note that the only information given to the simulator is in the part "Set Neighborhood/Infected", in step 6, namely a pseudonymized version of the neighborhood graph, and the subset of corrupted parties that are infected. In this pseudonomyzed graph, uncorrupted receivers are split into individual nodes (before renaming), as two corrupted senders cannot detect if they are broadcasting to the same participant. Uncorrupted senders are kept, as two corrupted receivers can detect that they receive the same pid and therefore know that they were in contact with the same party (in the same short-term epoch). Edges between uncorrupted parties are removed entirely. Additionally, all renaming/pseudonymizations are independent for each epoch. This means that an attacker can not re-identify uncorrupted users (or split-up individual nodes) across short-term epochs.

$\mathcal{F}_{\mathrm{CT}}$

State:

- Short term epoch $e_{st} \in \mathbb{Z}_{96}$ and long term epoch $e_{lt} \in \mathbb{N}$.
- Set of corrupted parties $\mathcal{P}_{corrupted}$.
- Set of infected parties $\mathcal{P}_{infected}$
- A sequence of all neighbourhood graphs so-far $(G_i = (\hat{\mathcal{P}}_i, E_i))_i$, i.e. the global meeting history. Let $I = \mathbb{N} \times \mathbb{Z}_{96}$ be the index set of the sequence.
- The current neighborhood graph $G = (\hat{\mathcal{P}}, E) = G_{(e_{lt}, e_{st})}$ and its pseudonymyzed version $G' = (\hat{\mathcal{P}}, E')$

– Parties at risk $\mathcal{WP} \subseteq \hat{\mathcal{P}} \times \mathbb{N}$, that assigns for each $P \in \mathcal{WP}$ a set of epochs during which the party met with an infected party (hence evoking the warning).

Set Neighborhood/Infected:

1. Receive directed neighborhood graph $G = (\hat{\mathcal{P}}, E)$ and a set of infected parties $\mathcal{P}_{infected}$ from a party P_{mat}.
2. Add G to the global meeting history.
3. Set $E' = \{(P_0, P_1) \in E \mid P_0 \in \mathcal{P}_{corrupted} \vee P_1 \in \mathcal{P}_{corrupted}\}$.
4. For all $\alpha = (P_0, P_1) \in E'$ with $P_0 \in \mathcal{P}_{corrupted}$, $P_1 \in \hat{\mathcal{P}} \setminus \mathcal{P}_{corrupted}$, replace α with $\alpha' = (P_0, \alpha)$.
5. Select a random, injective mapping $\text{pseudonymize}_i : (\hat{\mathcal{P}} \setminus \mathcal{P}_{corrupted}) \cup (\hat{\mathcal{P}} \times \hat{\mathcal{P}}) \mapsto \{0,1\}^{2n}$. Extend pseudonymize_i by $\text{pseudonymize}_i(P) = P$ for all $P \in \mathcal{P}_{corrupted}$. Set $E' := \{(\text{pseudonymize}_i(x), \text{pseudonymize}_i(y)) \mid (x, y) \in E'\}$, i.e. rename all nodes in E'. Let Q be the set of nodes associated with the set of edges E'.
6. Leak $(\mathcal{Q}, E'), \mathcal{P}_{infected} \cap \mathcal{P}_{corrupted}$ to the Simulator $\mathcal{S}$.
7. Increment e_{st} (in $\mathbb{Z}_{96}$).
8. If e_{st} then increment e_{lt} and delete all (P, t) pairs from $\mathcal{WP}$ where $0 \leq t \leq e_{lt} - 14$.

Send Broadcast:

1. Receive and ignore (*sendBroadcast*) from a participant P.

Replay/Relay:

1. Receive $(\text{relay}, t, P_1, P_2, P_3, P_4)$ from the simulator.
2. Let $P'_j := \text{pseudonymize}_i^{-1}(P_j)$ for $j \in \{1, 2, 3, 4\}$.
3. If $(P'_1, P'_2) \in E_t$, $(P'_3, P'_4) \in E$, $P'_2 \in \mathcal{P}_{corrupted}$, and $P'_3 \in \mathcal{P}_{corrupted}$, add the new edge (P'_1, P'_4) to E_t.

Release a Warning:

1. Receive (*positive*) from party P.
2. Let $R := \mathbb{N} \cap [e_{lt} - 14, e_{lt})$.
3. If P is corrupted, send (*forceWarning*) to the simulator, asking for subsets S_i of (the pseudonyms of) uncorrupted parties which have been in proximity to a corrupted party during the epochs in R, i.e. $S_i \subseteq \{q \in \text{pseudonymize}_i(\hat{\mathcal{P}} \setminus \mathcal{P}_{corrupted}) \mid \exists q' \in \mathcal{P}_{corrupted} \text{ where } (\text{pseudonymize}_i^{-1}(q), q') \in E\} \cup \mathcal{P}_{corrupted}$. After the simulator responds, set $\Delta\mathcal{WP}_i = \text{pseudonymize}_i^{-1}(S_i)$ as the set of parties that will receive a warning in the current epoch.
4. If P is uncorrupted, lookup in the meeting history G_i, per epoch $i \in R \times \mathbb{Z}_{96}$, for edges with P as receiver and determine the according set of senders $\Delta\mathcal{WP}_i$.
5. For each $i = (i_{lt}, i_{st}) \in R \times \mathbb{Z}_{96}$, add $\{(P', i_{lt}) \mid P' \in \Delta\mathcal{WP}_i\}$ to the list of active warnings $\mathcal{WP}$.

Handle Warning Query:

1. Receive $(query, t)$ from party P
2. Return 1 if $(P, t) \in \mathcal{WP}$, otherwise return 0.

A.2 Real Protocol

In this section, we give the pseudocode of the of the real protocol for all involved parties. To avoid duplication and for ease of comprehension, we leave out most of the concrete technical formulas for computing things, and refer the reader to our protocol description in Section 5.

We have make use of three (hybrid) ideal functionalities: i) $\mathcal{F}_{\text{med}}$, which represent the medical professional/health authority which gives out a TAN to upload data to parties which are deemed infected, ii) $\mathcal{F}_{\text{mat}}$, which represents the "world state", including a representation of who met whom (controlable by the environment), and a synchronized "epoch-wise" clock, and iii) $\mathcal{F}_{\text{reg}}$, which represents the party towards which parties run the registration protocol, and which keeps a list of registered parties.

$\mathcal{F}_{\text{med}}$

State:

– Set of infected parties $\mathcal{P}_{infected}$.

Set Infected:

1. Receive and store set of infected parties $\mathcal{P}_{infected}$ from a party P_{mat}.

Handling Warning Request:

1. Upon (*warningRequest*) from party $P \in \mathcal{P}_{infected}$.
2. Generate a tan $\leftarrow_{\$} \{0,1\}^m$.
3. Send (tan) to the *Matching Server*.
4. Send (tan) to participant P.

The following ideal functionality $\mathcal{F}_{\text{mat}}$ is a representation of the "material world"[8], i.e. it encodes/represents the succession of time, as well as who meets whom (then materialized as broadcasts), and who tests positive. A special incorruptible party P_{mat} will receive "orders to this world interface" from the environment, and forwards it to $\mathcal{F}_{\text{mat}}$ in the real world, see below. (In the ideal world, these orders are forwarded to $\mathcal{F}_{\text{CT}}$.)

$\mathcal{F}_{\text{mat}}$

State:

- Neighborhood graph $G = (\hat{\mathcal{P}}, E)$
- Current time $e = (e_{lt}, e_{st})$.

Set Neighborhood:

1. Receive and store directed neighborhood graph $G = (\hat{\mathcal{P}}, E)$ from a party P_{mat}.
2. increment e_{st} (in $\mathbb{Z}_{96}$). If $e_{st} = 0$, increment e_{lt} and send (*newLongTermEpoch*) to the submission server, the matching server, the warning server and all parties except P_{mat} (after the messages to the servers).

Receiving Broadcasts:

1. Receive (pid) from a participant P.
2. Send (pid) to participant $P_i \in \{P_j \in \hat{\mathcal{P}} \mid (P, P_j) \in E\}$.

As mentioned above, the special incorruptible party P_{mat} just forwards the relevant information (who met whom, and who is infected in this period) to the relevant functionalities.

Protocol of P_{mat}

Update Neighborhood and Infections:

- Receive a directed neighborhood graph $G = (\hat{\mathcal{P}}, E)$ and a set of infected parties $\mathcal{P}_{infected}$ from the environment.
- Send $G = (\hat{\mathcal{P}}, E)$ to $\mathcal{F}_{\text{mat}}$.
- Send $\mathcal{P}_{infected}$ to $\mathcal{F}_{\text{med}}$.

The following (hybrid) functionality $\mathcal{F}_{\text{reg}}$ represents the registration interface for parties that participate in the registration process. This is e.g. for obtaining a token dispenser to perform the regular uploads.

In the real protocol, this is usually augmented by an SMS challenge, to avoid an adversary to register a large amount of Sybils. To keep the model simple, we do not incorporate this into $\mathcal{F}_{\text{reg}}$. (Note that an SMS challenge, as well as the upload TAN, might be modeled via an authenticated channel from the party, for which an adversary can break authentication by guessing the short. See [1] for a formal hybrid functionality of this channel).

$\mathcal{F}_{\text{reg}}$

State:

- Registered parties $\mathcal{RP}$
- Issuer secret and public key for e-token dispensers $(\mathsf{sk}_{\mathcal{I}}, \mathsf{pk}_{\mathcal{I}})$

Registering a Party:

1. Upon $(register, \mathsf{pk}_{\mathcal{U}})$ from party P over an authenticated channel: if $P \notin \mathcal{RP}$ store P in $\mathcal{RP}$, else abort.
2. Issue a new token dispenser for P acting as $\mathcal{U}$ by participating as $\mathcal{I}$ in $\mathbf{obtain}(\mathcal{U}(\mathsf{pk}_{\mathcal{I}}, \mathsf{sk}_{\mathcal{U}}, 1), \mathcal{I}(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{I}}, 1))$.

Now, we are ready to give the protocol of the application, as run by the honest parties:

Protocol of the app

8 Internally, the author(s) humorously prefer to read the name of $\mathcal{F}_{\text{mat}}$ as "the matrix".

State:
- Current epoch $e = (e_{lt}, e_{st}) \in \mathbb{N} \times \mathbb{Z}_{96}$
- $(\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$
- Set of earlier tuples $(\mathsf{wid} := g^u h^{\mathsf{rid}}, u, k)$, where k is the according epoch.
- Set of recorded broadcasts of pids.
- $(\mathsf{rid}, \mathsf{wid} := g^u h^{\mathsf{rid}}, u)$
- $\mathsf{sid} := \mathsf{Enc}(\mathsf{pk}_{\mathsf{wid}}, \mathsf{wid})$
- seed
- $\{r_j\}_{j\in[1,\cdots,96]}$ with $r_j := \mathsf{PRG}(\mathsf{seed}, j)$
- Token dispenser D.
- $\{(\mathsf{sid}'_j, \mathsf{pid}_j)\}_{j\in[1,\cdots,96]}$ with $\mathsf{sid}'_j := \mathsf{ReRand}(\mathsf{sid}; r_j)$ and $\mathsf{pid}_j := \mathsf{H}(\mathsf{sid}'_j)$

Register:
1. When a new party with real identity rid is created by the environment, it first generates a secret/public key pair $(\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$ and then sends $(\mathit{register}, \mathsf{pk}_{\mathcal{U}})$ to $\mathcal{F}_{\mathrm{reg}}$.
2. Obtain a token dispenser D by participating as $\mathcal{U}$ in $\mathbf{obtain}(\mathcal{U}(\mathsf{pk}_{\mathcal{I}}, \mathsf{sk}_{\mathcal{U}}, 1), \mathcal{I}(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{I}}, 1))$ with $\mathcal{F}_{\mathrm{reg}}$ acting as $\mathcal{I}$.
3. Initialize the state and call **Upload Submission**.

Upload Submission:
1. Generate fresh $(\mathsf{wid}, \mathsf{seed}, \mathsf{sid})$ and the according list of $\{(\mathsf{sid}'_j, \mathsf{pid}_j)\}_{j\in[1,\cdots,96]}$.
2. Submit the e-token by participating as $\mathcal{U}$ in $\mathbf{show}(\mathcal{U}(D, \mathsf{pk}_{\mathcal{I}}, e_{lt}, 1), \mathcal{V}(\mathsf{pk}_{\mathcal{I}}, e_{lt}, 1))$ to the *Submission Server*, which acts as $\mathcal{V}$.
3. Send $(\mathsf{seed}, \mathsf{sid})$ over the same channel to the *Submission Server*.

Scheduled Upload:
1. Upon $(\mathit{newLongTermEpoch})$ from $\mathcal{F}_{\mathrm{mat}}$.
2. Increment e_{lt}.
3. Dequeue outdated stored wids and recorded pids.
4. Enqueue the current $(\mathsf{wid} := g^u h^{\mathsf{rid}}, u, e_{lt})$ and delete the remaining local ephemeral storage.
5. Continue as in "Upload Submission".

Sending Broadcasts:
- Upon $(\mathit{sendBroadcast})$ as input from the environment.
- Send $(\mathsf{pid}_{e_{st}})$ to $\mathcal{F}_{\mathrm{mat}}$ and increment e_{st}.

Recording Broadcasts:
- Upon (pid) from $\mathcal{F}_{\mathrm{mat}}$.
- Enqueue (pid, e_{lt}).

Match Request:
- Upon (*positive*) as input from the environment.
- Send $(\mathit{warningRequest})$ to $\mathcal{F}_{\mathrm{med}}$.
- Receive (tan) from $\mathcal{F}_{\mathrm{med}}$.
- Extract the list L of all recorded/received public identities from the queue.
- Send (L, tan) to the *Matching Server*.

Querying a Warning:
1. Upon (query, t) from the environment.
2. Find the corresponding wid for t and send (wid) to the *Warning Server*.
3. Receive bit b.
4. Output b to the environment.

Last but not least, we specify the protocol of the relevant servers, namely i) the submission server, ii) the matching server, and iii) the warning server.

Protocol of the Submission Server

State:
- Current epoch e_{lt}.
- The current epoch's batch of $\{(\mathsf{sid}'^k_j, \mathsf{pid}^k_j)\}_{j\in[1,\cdots,96]}$.

Handling Submissions:
1. Verify the token by participating as $\mathcal{V}$ in $\mathbf{show}(\mathcal{U}(D, \mathsf{pk}_{\mathcal{I}}, e_{lt}, 1), \mathcal{V}(\mathsf{pk}_{\mathcal{I}}, e_{lt}, 1))$.
2. Detect possible Double-Spending.
3. Receive $(\mathsf{seed}, \mathsf{sid})$ from $\mathcal{U}$.
4. Generate $\{(\mathsf{sid}'_j, \mathsf{pid}_j)\}_{j\in[1,\cdots,96]}$ with the help of the seed.
5. Add the generated tuples to the batch of e_{lt}.

Forwarding Submissions:
1. Upon $(\mathit{newLongTermEpoch})$ from $\mathcal{F}_{\mathrm{mat}}$.
2. Shuffle the last batch and send the complete batch to the *Matching Server* together with e_{lt}.
3. Increment e_{lt}.
4. Create a new empty batch for the new epoch.

Protocol of the Matching Server

State:
- The current epoch e_{lt}.
- Per epoch e_{lt} a set $\mathcal{B} := \{(\mathsf{sid}'^n_j, \mathsf{pid}^n_j)\}_{j\in[1,\cdots,96]}$, where n is the number of overall stored batches.
- Set of pending matching requests L.

Removing Outdated Information:
1. Upon (*newLongTermEpoch*) from $\mathcal{F}_{\text{mat}}$.
2. Increment e_{lt} and delete all $(\mathsf{sid}', \mathsf{pid})$ pairs stored in the internal state which were stored in epoch $0 \leq e'_{lt} \leq e_{lt} - 14$.

Handling Submissions:
- Receive $\{(\mathsf{sid}'^k_j, \mathsf{pid}^k_j)\}_{j\in[1,\cdots,96]}$, *epoch* from the *Submission Server* and store it in $\mathcal{B}$.

Preparing Match Request:
- Receive (tan) from $\mathcal{F}_{\text{med}}$ and insert it into L.

Handling Match Request:
1. Receive a (S, tan) from a participant P, where S is a set of pids.
2. If $\text{tan} \in L$, remove tan from L, otherwise abort.
3. For each $\mathsf{pid} \in S$, look up the corresponding sid' in the internal state, i.e., the sid' for which $(\mathsf{sid}', \mathsf{pid})$ is contained in the internal state.
4. Rerandomize and forward the set of matched encrypted warning identifiers $\{\mathsf{sid}''_l := \mathsf{ReRand}(\mathsf{sid}'_l)\}$ per *epoch* to the *Warning Server*.

Protocol of the Warning Server

State:
- The current epoch e_{lt}.
- $(\mathsf{sk}_{\mathsf{wid}}, \mathsf{pk}_{\mathsf{wid}})$.
- Set $\mathcal{W}$ of released wids and their validity epoch t.

Removing Outdated Information:
1. Upon (*newLongTermEpoch*) from $\mathcal{F}_{\text{mat}}$.
2. Increment e_{lt} and delete all $(\mathsf{wid}, t) \in \mathcal{W}$, with $0 \leq t \leq e_{lt} - 14$.

Issuing Warnings:
- Receive a list $\{(\mathsf{sid}''_l, t_l)\}$ from the *Matching Server*.
- Decrypt, deduplicate and add the received warning identifiers $\{(\mathsf{wid}_l = \mathsf{Dec}(\mathsf{sk}_{\mathsf{wid}}, \mathsf{sid}''), t_l)\}$ to $\mathcal{W}$.

Warning Query:
- Receive warning identifier (wid).
- Search all finished *epoch* for wid and return 1 if a match is found, 0 otherwise.

A.3 Simulator

We now give the pseudocode of the simulator. A proof showing the distinguisher $\mathcal{D}$ cannot distinguish the "real world" experiment (where a real adversary tries to attack the real protocol) and the "ideal world" (where the simulator $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\text{CT}}$) will appear in the full version.

Simulator $\mathcal{S}$

State:
- Short term epoch $e_{st} \in \mathbb{Z}_{96}$ and long term epoch $e_{lt} \in \mathbb{N}$. We write e for (e_{lt}, e_{st}).
- Set of corrupted parties $\mathcal{P}_{corrupted}$.
- Set of infected corrupted parties $(\mathcal{P}_{infected})_i$.
- A sequence of all pseudonymized neighborhood graphs so-far $(G'_i = (\mathcal{Q}_i, E'_i))_i$, i.e. a pseudonymized meeting history from the point of view of the adversary.
- The current pseudonymized neighborhood graph $G' = (\mathcal{Q}, E') = G'_{(e_{lt}, e_{st})}$.
- List $\mathcal{W}$ of active of warnings (wid, t) from corrupted parties to corrupted parties.
- Issuer secret and public key for e-token dispensers $(\mathsf{sk}_{\mathcal{I}}, \mathsf{pk}_{\mathcal{I}})$.
- The key pair $(\mathsf{sk}_{\mathsf{wid}}, \mathsf{pk}_{\mathsf{wid}})$ of the simulated warning server.
- Set $\mathcal{C}$, which records the information exchanged between corrupted parties and the simulated parties by storing tuples $(\mathsf{wid}, e_{lt}, \mathsf{pid}_1, \ldots \mathsf{pid}_{96})$
- Set L which is the current pending upload request.

Register a Party:

1. Upon receiving $(\mathit{register}, \mathsf{pk}_{\mathcal{U}})$ from $P \in \mathcal{P}_{\mathit{corrupted}}$ for $\mathcal{F}_{\mathrm{reg}}$.
2. Issue a new token dispenser for P like $\mathcal{F}_{\mathrm{reg}}$ does.

Next Epoch:

1. Upon receiving $((\mathcal{Q}, E'), \mathcal{P}'_{\mathit{infected}})$ from $\mathcal{F}_{\mathrm{CT}}$ replace the current neighborhood graph and add the most recent neighborhood graph to the history of pseudonymized neighborhood graphs, and replace it with $(\mathcal{Q}, E')$.
2. Assign random pids to honest nodes.
3. For each edge $(P_1, P_2) \in E'$, where $P_1 \in \hat{\mathcal{P}} \setminus \mathcal{P}_{\mathit{corrupted}}$ and $P_2 \in \mathcal{P}_{\mathit{corrupted}}$, send the pid assigned to P_1 to P_2 on behalf of $\mathcal{F}_{\mathrm{mat}}$.
4. Increment e_{st} (in $\mathbb{Z}_{96}$).
5. If $e_{st} = 0$ then increment e_{lt} and remove all entries (wid, t) from $\mathcal{W}$ where $t \leq e_{lt} - 14$.

Upload from Corrupted User:

1. When $P \in \mathcal{P}_{\mathit{corrupted}}$ wants to show a token, act as $\mathcal{V}$ and verify the token.
2. If the token is correct (and not reused) receive $(\mathsf{seed}, \mathsf{sid})$ from $P \in \mathcal{P}_{\mathit{corrupted}}$ for the *Submission Server*.
3. Expand the input in the same way as the *Submission Server*. Store the expanded set for this epoch e.
4. Compute $\mathsf{wid} \coloneqq \mathsf{Dec}(\mathsf{sk}_{\mathsf{wid}}, \mathsf{sid})$.
5. Store $(\mathsf{wid}, e_{lt}, \mathsf{pid}_1, \ldots \mathsf{pid}_{96})$ in $\mathcal{C}$.

Broadcast from Corrupted User:

1. Upon receiving (pid) from a participant $P \in \mathcal{P}_{\mathit{corrupted}}$ for $\mathcal{F}_{\mathrm{mat}}$.
2. Look up for all targets of this broadcast in G'.
3. Annotate all according edges with pid.
4. Send (pid) to all corrupted targets from $\mathcal{F}_{\mathrm{mat}}$.
5. Look up if one of the nodes in one of the G'_i (or G') was assigned pid. If so, let P_1 be the node, P_2 be an arbitrary corrupted receiver of the broadcast from P_1, $P_3 = P$, t be the corresponding epoch of G'_i (or G'). For each target P_4, send $(\mathit{relay}, t, P_1, P_2, P_3, P_4)$ to $\mathcal{F}_{\mathrm{CT}}$.

Warning Request from Corrupted User:

1. Upon receiving $(\mathit{warningRequest}, P)$ from $P \in (\mathcal{P}_{\mathit{infected}})_{(e_{lt}, e_{st})}$ for $\mathcal{F}_{\mathrm{med}}$.
2. Generate a Tan tan.
3. Add tan to the list of valid tans.
4. Send tan to P from $\mathcal{F}_{\mathrm{med}}$.

Matching Request:

1. Upon receiving (L, tan) from a participant $P \in \mathcal{P}_{\mathit{corrupted}}$ for the *Matching Server*, where L is a finite set of public identities.
2. If the tan is invalid, ignore the request.
3. For each $\mathsf{pid} \in L$ try looking up pid in $\mathcal{C}$. If there is a match, let wid, t be the corresponding warning identity and epoch. If $e_{lt} - 14 \leq t < e_{lt}$, mark wid as at-risk by inserting (wid, t) into $\mathcal{W}$.
4. Store L in the state.
5. Send $(\mathit{positive})$ to $\mathcal{F}_{\mathrm{CT}}$ from P.

Force a Warning:

1. Upon receiving $(\mathit{forceWarning})$ from $\mathcal{F}_{\mathrm{CT}}$.
2. Find the corresponding set L the corrupted party was trying to send to the matching server.
3. Let $R \coloneqq \mathbb{N} \cap [e_{lt} - 14, e_{lt})$.
4. Let $S_i \coloneqq \emptyset$ for $i \in R \times \mathbb{Z}_{96}$.
5. For each $\mathsf{pid} \in L$: Search the pseudonymized graphs G'_i (for $i \in R \times \mathbb{Z}_{96}$) for a node q labeled pid. If a node is found in graph G'_i, insert q into S_i.
6. Send the sets S_i back to $\mathcal{F}_{\mathrm{CT}}$.

Warning Query from Corrupted User:

1. Upon receiving (wid) from a participant $P \in \mathcal{P}_{\mathit{corrupted}}$ for the *Warning Server*.
2. If there is a t such that (wid, t) is found in $\mathcal{W}$, return 1.
3. If wid is found in $\mathcal{C}$, but not in $\mathcal{W}$, let $\mathsf{pid}_1, \ldots \mathsf{pid}_{96}$ be the public identities for wid stored in $\mathcal{C}$. Determine all broadcasts sent by corrupted parties where one of the pid_i was sent. For each of these broadcasts, determine the sending corrupted party P and the (long-term) epoch t' of the broadcast. Send (query, t') from party P to $\mathcal{F}_{\mathrm{CT}}$. If $\mathcal{F}_{\mathrm{CT}}$ responds with 1, return 1. Otherwise continue with the remaining parties and epochs.
4. If wid is not found in $\mathcal{C}$, return 0.