Chapter 3
Decentralized Contact Tracing Protocols

Abstract  Digital Contact Tracing (DCT) protocols and systems usually rely upon close-range communication between handheld devices (like smartphones or tablets), primarily using the Bluetooth Low Energy (BLE) interface and a client-server mode of communication between the apps installed on the devices and the backend server(s). Depending upon the role played by the backend server(s), DCT protocols, and systems are usually classified under centralized or decentralized categories. In this chapter, we focus on decentralized DCT protocols, namely, Apple-Google Exposure Notification Framework (ENF), DP3T, East Coast PACT, West Coast PACT, and TCN. Towards the end, we describe Epione, which though categorized under decentralized systems according to the classification provided in Chap. 2, contains many features akin to that of centralized systems.

3.1 Introduction

The proximity graph of a contact tracing system is one in which users are considered as vertices and any two vertices are connected by an edge if the apps of corresponding users detect the presence of each other’s device in their vicinity. This graph could be directed or undirected according to whether the detection occurs in pairs of users interacting with each other (by exchanging proximity identifiers) or independently (by a broadcast of identifiers, which may be detected by other users even when the broadcasting app may not detect the receiving users) respectively. Centralized systems for contact tracing allow the central server to determine the neighborhoods of infected users; it may not always be possible or feasible to trust it with such information. This may be due to various reasons, some of which include security concerns like direct attacks on the server for access to or manipulation of bulk information from malicious entities, hardware and software capacity concerns like the possibility of overload on the server and related issues like generation of errors in the system,
social and legal matters such as privacy laws in a country or lack of trust of users towards the authority that manages the server, etc. Most of these reasons, especially the last one, may discourage a large proportion of the population from participating in the app-based contact tracing process, which may adversely affect the effectiveness of such measures in controlling the spread of the disease. Thus, in addition to the problem of contact tracing with security against non-authoritative adversaries, these concerns give rise to the problem of carrying out these tasks in a manner that is also secure from entities of the central administration like the backend server and the health authorities.

The dependence on the trustworthiness of the health authority cannot be completely eliminated as the system depends on it for diagnosis and medical advice; the security measures that can be taken from a cryptographic standpoint are quite simply the implementation of precautions such as encryption of user data, authentication, and verification of data through suitable algorithms, use of blockchains, hash chains, etc., (some of these approaches also provide added layers of security against attacks from users and other entities). However, the system must undergo drastic changes to ensure security against the central server. This fundamentally requires a definition of the amount and nature of data to which the server has access, in terms of its capability to compute additional information about users (e.g., infection or diagnosis status, neighborhoods, primary or secondary contacts, location, time markers, etc.) from this data.

### 3.1.1 Characteristics of Decentralized Systems

The common characteristics of almost all decentralized systems are as follows.

- Any centralized agency like the Administrative Authority or the Epidemiological Authority have limited information about the users (that can be used to personally identify the individual), contact events, or identifiers being broadcast/received between the apps or devices. However, the Health Authority would have access to infected users’ identities in the majority of the protocols.
- Most of the backend servers (e.g., the server that stores the list of proximity identifiers of infected users or the seeds from which these identifiers can be derived) are considered to be semi-trusted entities. The Health Authority server is considered to be a trusted one.
- If a user tests positive, then he/she would be expected to upload the proximity identifiers (that the user-app shared with the neighboring devices and/or received from such devices over the last 14 days or so) to the backend/bulletin-board server(s).
- The risk computation is mostly done at the user-app end.
- In case the server data is compromised, the extent of damage would be limited as no personal or personally identifiable user data are usually stored at the server end. In a distributed bulletin-board implementation of the server with sophisticated cryptographic algorithms, the list of random seeds/identifiers corresponding to the
infected users or the contact events can remain safe even if the data falls in the hands of malicious entities.

- Epidemiological requirements are not addressed in most of the decentralized protocols. In fact, DP3T specification explicitly puts this under out-of-scope.
- Intuitively it appears that the interoperability among the decentralized systems may be easy to achieve. However, none of the specifications elaborates a possible mechanism.
- Since most of the critical operations including evaluation of the at-risk or not at-risk status of users is carried out at device end, it may become complex to roll-out system-level changes, bug-fixes, and enhancements (e.g., modification of the risk-scoring algorithm) in decentralized systems as compared to centralized systems.

The common cryptographic primitives of such systems would be similar to those of centralized systems.

### 3.2 A General Framework of Decentralized Protocol

In the previous chapter, we have described centralized contact tracing protocols. Now we describe how a decentralized contact tracing protocol works. If A and B interact through a decentralized app, their time-of-contact data get stored locally in their apps through a direct broadcast, without the involvement of the server. For this reason, we call those protocols decentralized. If one of them (say A) has tested positive for SARS-CoV-2, a part of all interaction of A with other users would be reported to the server through a proper authentication channel. Any user, say B, can verify contact with A by matching the reported interaction with his own stored interaction. Thus, the role of the central server in any decentralized protocol is to facilitate through its database/bulletin board, the communication between an infected user with all other users.

In Chap. 1, we have described different phases of a DCTS. Here, we focus on the features that are specific to decentralized systems.

1. **Registration and Initialization Phase:** At the time of registration and initialization, the user-app would generate the first set of seeds/keys from which the proximity identifiers (as well as the subsequent daily or hourly seeds) would be formed. There would be no involvement of the backend server(s) in this activity.
2. **Contact-Broadcast Phase:** There is no difference in this phase between the centralized and decentralized systems/protocols.
3. **Reporting Phase:** If at any point of time, A tests positive, A’s app sends some information to the central authority. This authority uses a server to broadcast this information, which may be encrypted by a public key or maintained in a blockchain for added security. Alternatively, symmetric key generation and encryption using such keys can also be implemented.
4. **Risk-level (or Risk-score) Computation Phase:** Different apps that receive this information (automatically at regular intervals, or manually through user-generated requests) use a verification algorithm to check whether they had indeed come into contact with A using the broadcast information obtained from the server and the key(s) shared directly with A.

Below we describe the possible attack scenarios that could be mounted by various types of malicious users (or user groups), which every decentralized DCTS must be prepared to handle in a graceful manner.

### 3.2.1 Attack Scenarios

In the case of decentralized digital contact tracing systems, servers and communication channels can be considered to be less vulnerable as compared to the centralized systems, due to minimal roles played by the non-user entities. Hence, such systems can be envisaged to have stronger cryptographic safeguards. However, errors in communication, heavy traffic, overloading, etc. (because of genuine reasons or malicious adversaries), cannot be completely ruled out. Thus, false negatives, linkage attacks and DoS attacks may still occur as a consequence of these factors.

The designated health agencies who have the testing authority are also expected to guide patients through diagnosis, evaluation of exposure risks, etc. The possible misuse of these functions by a corrupt health authority, even though may happen in reality, cannot be cryptographically controlled. Hence, for the purpose of security analysis, we would consider the Health Authority as a trusted entity. In the following subsections, we would consider the attack scenarios mainly from the context of DP3T, Apple-Google Framework, East Coast PACT, West Coast PACT, and TCN.

#### 3.2.1.1 Integrity Attacks Through False Positives

Adversaries may execute replay attacks by either broadcasting identifiers of infected individuals as soon as they are uploaded to the server or by creating a pool of (false) identifiers and broadcasting them to unknowing users. The first attack can be averted by simply securing the communication channels to the server. The second attack is only successful if the pool of false identifiers contains some matches with the identifiers uploaded by infected users; it can be easily prevented by timestamping all identifiers. However, care must be taken to keep the timestamps as coarse as possible as the inclusion of exact times could lead to loss of privacy. Another way to avoid replay attacks is to replace the broadcasting of identifiers by an interactive protocol between two users coming into contact with each other for authentication.

Relay attacks are stronger than replay attacks as they can circumvent timestamping and other measures that could deal with replay attacks. Implementation of distance-bounding protocols, inclusion of a coarse location-stamp through GPS, etc., can be
3.2 A General Framework of Decentralized Protocol

used to mitigate relay attacks. Distance-bounding is more difficult to implement and could lower the privacy of users. Implementing an interactive protocol between two users along with a binding commitment scheme (as proposed for DP3T in [38]) is another solution, albeit costly.

An inverse-Sybil attack can be executed on such protocols, possibly on a large scale, by collecting the identifiers broadcasted by infected users (before or after uploading to the server), and further broadcasting them to other users. A malicious entity could pay users for such information if the identifiers provided by them are indeed uploaded to the server. Use of timestamping and distance-bounding protocols together could be a means to evade the inverse-Sybil attack, but this is certainly costly. Encrypting the data on the server could inconvenience an adversary wanting to check the uploads on the server, but would not prevent this attack. Involvement of the health authority for authentication and upload of identifiers of positively tested users could reduce the extensiveness of this attack.

False reports of infection could be sent to the server by malicious users, but this can be easily mitigated in various ways such as running the user-app on a Trusted Platform Module (TPM), authenticating the identifiers (or keys or seeds) uploaded by the users through the health authority or strictly restricting the uploads to the health authority (on consent from the user) and accepting identifier sets only in acceptable sizes according to the length of the infection period.

3.2.1.2 Integrity Attacks Through False Negatives

As users have the choice of not uploading their keys even on positive testing, some malicious users could accomplish a basic false negative attack by refusing to upload (some, all or correct values of) their identifiers to the server even on infection. An honest infected user may be forced to behave like this under coercion. Other reasons for false negatives could be fluctuations in Bluetooth signal strengths, incorrect evaluation, DoS attack, etc.

3.2.1.3 Privacy Attacks

Privacy of users may be compromised if the decentralized systems are subjected to a linkage attack, deanonymization attack, single-entry attack, address carryover attack, shared carryover attack, etc. Sending alerts to exposed users after an acceptable amount of time delay could be a possible way to work around some of these attacks. Other methods such as randomly permuting the identifiers received could also be employed. As all protocols discussed here involve some kind of proximity identifiers (pseudonymous and pseudorandom numbers), an important point of consideration is the frequency at which these identifiers must be changed. Tracking of BLE MAC addresses could lead to deanonymization or linkage attacks if performed on a system with a lower frequency of change.
3.3 DP-PPT/DP3T

Decentralized Privacy-Preserving Proximity Tracing (DP3T/DP-3T) is a protocol and framework proposed by a group of researchers, technologists, engineers, epidemiologists, and legal experts from a number of institutions and organizations. The list of affiliations of the team and the entire set of documents are included in the Github repository [14]. The team has also released the reference implementations of the app and the server code in the open-source domain [15, 16]. The “Who we are” page [14] clarifies that DP3T is a “free-standing effort.” Although some of its members are part of the umbrella organization called Pan-European Privacy-Preserving Proximity Testing (PEPP-PT) project, it would be incorrect, as per the DP3T team, to use the term PEPP-PT for any particular protocol (e.g., DP3T), since there is a drastic difference in the approach of DP3T with some of the other centralized protocols endorsed by PEPP-PT.

For the purpose of this section, we primarily refer to the most recent white paper [11] published by the group, dated 25 May 2020.

3.3.1 Framework and Design Principles

The DP3T repository includes a set of documents that describe protocol specifications, recommendations, analyses, and comparative study (with respect to other protocol specifications like BlueTrace, PEPP-PT NTK, ROBERT, DESIRE, etc.). The current version of the white paper [11] describes three variations of a decentralized protocol, called—(1) the low-cost design, (2) the unlinkable design, and (3) the hybrid design. Apart from this main document, the repository contains an Interoperability Specification [17], an analysis of Privacy and Security Attacks [18], the Upload Authorization Analysis and Guidelines [19], the Exposure Score Calculation [20], etc.

In scope and out-of-scope areas: The specification [11] highlights that the purpose of the protocol and the framework is to complement (and not replace) the manual contact tracing process through a smartphone-based digital proximity tracing system by providing “a mechanism that alerts users who have been in close physical proximity to a confirmed COVID-19 positive case for a prolonged duration that they may have been exposed to the virus.” The aims of the DP3T project are documented in [12].

Interestingly, it also points out the out-of-scope areas (called non-goals), like (a) tracking of users who have been diagnosed positive (b) identify the hotspot areas or the trajectories of infected cases, and (c) sharing any data for epidemiological research purposes. The last point (about not sharing any data for research) might have some disagreement with certain statements in the Simplified Overview docu-
ment [13], like “epidemiologists obtain an anonymized proximity graph with minimal information.”

The system requirements of DP3T framework have been grouped under the following categories.

- **Proximity tracing requirements**: (a) Completeness (implying that all close proximity events are captured), (b) Precision (which means reported events correspond to the close proximity events), (c) Authenticity (e.g., users cannot fake exposure events), (d) Confidentiality (e.g., contact history would not get compromised or leaked), and (e) Notification (i.e., individuals can be informed about exposure risk through the system).

- **Digital privacy rights and data protection requirements**: For example, adherence to European General Data Protection Regulations [31], etc.

- **Scalability and interoperability requirements**: This is to ensure that the system can be used for the pandemic for a long period of time (if needed) at a global scale.

- **Technical feasibility requirements**: The system should not rely on new technological breakthroughs and should be deployable over a wide variety of hardware types (as mobile devices).

In spite of declaring these system requirements in an explicit fashion, the document does not validate, analyze, or study these properties on the proposed protocol for any of the three variations of the design. Surprisingly, there is no mention of the words like “Completeness” or “Precision” in the entire white paper apart from the “System requirements” section.

### 3.3.2 Protocol Details

We now describe the protocol details for the three design variations as proposed in the current version [11] of the specification. In all the designs, the proximity identifier is referred to as the ephemeral ID (EphID).

**Low-cost design:** A random 32-byte seed \( SK_t \) is generated on the first day \( t \) as per UTC convention of days and on every day thereafter a new secret seed is generated as \( SK_{t+1} = H(SK_t) \), where \( H \) is a cryptographic hash function. A day is divided in \( L \)-minute-long fixed durations called the epochs. For the entire day, a list of \( n \) EphIDs are generated, where \( n \) equals \( \frac{24 \times 60}{L} \) in the following manner:

\[
\text{EphID}_1 \| \ldots \| \text{EphID}_n = \text{PRG(PRF}(SK_t, \text{“broadcast key”}))
\]

where PRF is a pseudorandom function like HMAC-SHA256 and PRG is a pseudorandom generator (e.g., AES in counter mode). The user-app randomly picks one of the \( n \) EphIDs for each epoch and broadcasts the same in that duration. When an app receives one such EphID inside a broadcast BLE packet, it stores the same in local app-storage with the signal attenuation indicator and the day (like “July 10”).
This results in a storage requirement of typically 6.1 MB, for a period of 14 days. In addition, the app also stores the $SK_t$ values for a period of 14 days (which is a configurable parameter that is set as per recommendations from the epidemiologists). When a user is diagnosed positive, he may choose to upload (after receiving relevant authorization/validation from the Health Authority), the $(SK_t, t)$ pair corresponding to the starting day from when he could have been contagious. After uploading is done, the infected user’s app chooses a new random key as the starting secret key to generate the EphIDs from that day onwards. Each user-app on a regular interval downloads the list of such pairs corresponding to the infected users from the backend and derives the subsequent EphIDs to check for any match in the local database and determine the extent of exposure by referring to the associated metadata like signal attenuation and date.

**Unlinkable design:** In this approach, the smartphone draws a random 32-byte seed $(seed_i)$ for the epoch $i$ and sets $EphID_i = \text{LEFTMOST128}(H(seed_i))$, where LEFTMOST128 takes the leftmost 128 bits from the output of the cryptographic hash function $H$. The user-app stores all the generated seeds in the past 14 days or so. When an app receives a broadcast BLE beacon, it stores a hashed version of the received EphID in the form of $H(EphID || i)$, where $H$ is the cryptographic hash function and $i$ is the epoch number in which the beacon was received. Like the low-cost design, the app also stores the signal attenuation and the day (like “July 10”). This results in a storage requirement of typically 6.9 MB, for a period of 14 days. In this design, the app allows an infected user to redact some of the entries to be uploaded to the backend server by choosing a subset $(I)$ of epochs from the entire period for which he could be contagious. The list of $(seed_i, i)$ pairs for all $i$-values that belong to the set $I$ are then uploaded to the server which periodically (every 2 h) runs through such sets and creates a new ‘Cuckoo’ filter $(F)$ that includes $H(\text{LEFTMOST128}(H(seed_i)) || i)$ entry for every $(seed_i, i)$ pair. Every user-app downloads the list of new Cuckoo filters and checks if there is any match against the received EphID entries stored in its local database. This ensures that the actual EphIDs of the infected users do not get revealed to any other user in the system. The parameters of the Cuckoo filter is designed in such a way that there is a very low probability of False Positives (like one in a million users who could be using the system for over 5 years). The exposure detection algorithm in this design is identical to that of the low-cost design.

**Hybrid design:** As the name suggests, this is a hybrid of the low-cost and the unlinkable designs. In this design, a new secret seed is generated afresh in a time-window $(w)$ which could typically be 2–4 h long and may range from 10 min to 1 day. The relationship of the epoch duration $(L)$ with $w$ and $n$ (the number of EphIDs in a time window $w$) is given by $n = \frac{w}{L}$. The 16-byte long EphIDs are generated as follows:

$$\text{EphID}_{w,1} || \ldots || \text{EphID}_{w,n} = \text{PRG}(\text{PRF}(seed_w, \text{“DP3T − HYBRID”}),$$
where PRF is a pseudorandom function like HMAC-SHA256 and PRG is a pseudorandom generator like AES in counter mode. When an app receives one such EphID inside a broadcast BLE packet, it stores the same in local app-storage with the signal attenuation indicator and the day (like “July 10”—this is similar to the approach used in the low-cost design. This results in a storage requirement of typically 4.8 MB, for a period of 14 days. When a user is diagnosed positive, he would have the option to redact certain entries before uploading the seeds to the server after necessary authorization from the Health Authority. The exposure detection step by matching the downloaded seeds from the server and the exposure detection algorithm are identical to that of the low-cost design. The specification has mentioned that if the value of $w$ is equal to 1 day (in minutes), the design would appear similar to that of the Apple-Google Exposure Notification Framework.

### 3.3.3 Highlights and Characteristics

Some of the highlights and characteristics of the DP3T framework are as follows.

- It has assimilated a diverse set of elements and parameters in the three design variations. Hence, the framework may work out very differently for the three approaches when it comes to implementation trade-offs or user experience.
- The framework is focused on keeping the footprints of data stores (at device and server end) and device to server communications small so that the bandwidth usage and latency can be minimized. It has also published details of estimated data sizes and the requirements of daily data download size for each of its three design variations in multiple European countries. To the best of our knowledge, no other protocol or framework has gone to this extent of analysis in terms of bandwidth usage and data requirements. The specification has analyzed the security and privacy considerations for the three design variations under different threat models in an extensive fashion. It has also published separate documents that study the security and privacy for decentralized systems in a generic way as well as presented specific security and privacy analyses for some of the other protocols, including PEPP-PT, ROBERT, DESIRE, etc. The document repository also includes a note that addresses the possible alternate ways and mechanisms on how the Health Authority may validate the infected user’s status.
- The specification also touches upon the multi-country or multi-state interoperability considerations for apps based on the DP3T protocol and the overview of a mechanism for an integrated exposure computation based on per-day exposure score.
- The “Readme” file [14] refers to the Apple-Google Exposure Notification Framework and has mentioned that “DP-3T appreciates the endorsement of these two companies for our solution and has been working with both of them to implement our app on their platforms”—it would be worthwhile to watch this space to see further developments that may come out through such a collaboration.
3.3.4 System Analysis

Since the basic system analysis has already been discussed in the previous subsections, let us go directly to the security and architecture analyses of the protocol.

3.3.4.1 Security Analysis

The following discussion is primarily inspired by the extensive analysis of DP3T done by Vaudenay in his well-referred paper [38]. However, the DP3T specification has also evolved over time and its proposed design approaches have been refined based on many of the comments and suggestions posted in the Github forum. The current version of the white paper also highlights most of these attack scenarios.

- **Backend impersonation**: The possibility of backend impersonation may not be ruled out and it would depend on the level of security and certification followed for the server that is expected to publish the seeds or Cuckoo filters corresponding to the infected users. Irrespective of which of the three designs are followed in the implementation, a malicious backend server can cause harm by sharing incorrect seeds/filters with the apps, which would generate a large-scale False Negatives.

- **Server data breach**: The backend server does not store any user-specific personal or personally identifiable information. Hence, even if there is a data breach at the server end, there would be no serious risk of losing sensitive data. However, there could be a considerable risk of deanonymization of infected users who have uploaded their seeds to the server.

- **Replay attack**: In the low-cost design, there is a high chance of replay attack within a duration of a day. In the other two designs, this could be feasible within the duration of an epoch (L minutes).

- **Relay attack**: DP3T is not resistant to relay attacks (although engineering such an attack is always cumbersome for an adversary) for all the three design variations.

- **Deanonymization of users**: Since the seeds/filters corresponding to the infected users are directly downloaded by every user-app, deanonymization can be implemented by malicious users or user groups in all the three design variations. Such users may arrange to passively listen to EphIDs of users (along with additional information like captured images of people who would have come near the beacon capturing devices), especially in places where the density of infected users is expected to be high (like in hospitals) and matching those against the EphIDs generated from the seeds (or running those through the Cuckoo filters). Single-entry attacks can also lead to deanonymization.

- **Inverse-Sybil attack**: DP3T is not resistant to such attacks as there is no permanent identifier of the user that is attached to a particular device identifier (SIM number or IMEI number) for any of the three design variations.

- **Coercion threats**: DP3T is also susceptible to such attacks irrespective of the design variation used. An infected person’s device can be misused by a malicious third party (before he/she uploads the proximity data to the server) by deliberately
3.3 DP-PPT/DP3T

• Bringing that device in proximity to a large number of people, which would trigger many false alarms in the ecosystem.

• **Miscellaneous:** DP3T is susceptible to Denial of Service (DoS) attack and many other generic mobile device’s OS or API layer attacks. In the white paper, it is mentioned as a footnote that MAC address change is assumed to be synchronized with the EphID change in BLE messages. If that does not happen properly, the protocol can be susceptible to address carryover attacks. There is no mention of message authentication at the BLE protocol layer. Hence, it is not clear how DP3T may resist message tampering attempts at the lowermost level.

3.3.4.2 Architecture Analysis

We now describe some of the possible open areas in the DP3T architecture.

• **Fault tolerance:** The specification has not talked about the login-id/user-id generation process. If a user’s app crashes and the user registers afresh to the system after installing the app again, would the previous user-id be linkable to the new user-id? If not, would the previous history of exposure be lost forever?

• **Background/foreground mode:** It is not specified if the app needs to run in foreground or background mode in different platforms so that it may function effectively.

• **Interoperability:** The interoperability specification talks about multi-country implementation of systems/frameworks based on the DP3T protocol; however, there is no mention of how multiple decentralized protocols or multiple centralized as well as decentralized systems may work together.

3.4 Apple-Google Exposure Notification Framework

Google and Apple have jointly developed a digital contact tracing protocol and a system of APIs, together called the Exposure Notification Framework (ENF) [22] that can support both Android and iOS devices. It is a decentralized protocol and has a lot of similarities with the DP3T [11] and TCN [10] specifications. The most significant difference of this framework with any other system is a tight integration of the APIs at the OS level that not only makes the APIs efficient but also allows any app that uses these APIs function effectively in the background mode, even when the device is locked. No other automated contact tracing protocol or system currently has this capability.

3.4.1 Framework

In case of earlier releases, users required an app to access the Exposure Notification APIs. These apps need to be approved by Google and Apple and the approval is given
only to the apps that belong to public health authorities. In the current release of ENF, Google, and Apple have developed Exposure Notification Express that integrates the core functionality in the form of a customizable app (for Android) or at the OS layer (for iOS), so that users may be able to benefit from the features even without installing any contact tracing app.

The complete list of criteria [33] that an app must meet to qualify for Google and Apple’s approval is as follows.

- The app must belong to a Government Health Agency that acts as the Health Authority in a given jurisdiction and must be deemed to be part of the authority’s response to the COVID-19 situation
- The app must explicitly seek user’s consent for accessing the APIs and for running in the background
- It must require user’s explicit consent before sharing any app-data or device-data with the backend server(s) when a user is diagnosed as positive (called the “affected user” in ENF)
- The app should gather minimal personal or personally identifiable data and any data collected should only be used for COVID-19 purpose and not for any other purpose (like an advertisement)
- Such an app must not access any geo-location information and should not seek consent from the user to have location access. In case, any existing DCT app has access to the geo-location data, it would not be able to utilize Exposure Notification APIs.
- One country can have only one such officially approved app unless that country has a federated state-based structure where each state is entitled to follow a separate approach (like the states of the USA) in which case the approval would be separately given for each state’s designated app.

ENF uses BLE in the broadcast (non-connectable) topology mode. The MAC addresses stored in the BLE broadcast packets are changed synchronously with the changing proximity identifiers.

### 3.4.2 Design Principle

Some of the design principles that have been highlighted by Google and Apple in their documentations [2], are as follows.

- A core design principle is to maintain the user’s security and privacy. This is achieved at multiple layers. In the API layers, this is ensured through the cryptographic primitives and algorithms. The decentralized architecture takes care of the fact that the backend server has a limited knowledge about user status and contact events. The computation of exposure risk happens entirely at the device end. The qualifying criteria for the apps (as described in the previous subsection) provides
users a better control over the security and privacy of their own data. A user’s choice plays a key role in how the app functions in his/her device.

- The specification is not dependent on any personal or personally identifiable data to be entered by the user (like phone number)
- Location metadata is not allowed to be tracked or stored in any form
- Google and Apple may also disable the service in any country or region once it is no longer required
- Both the companies are committed to not monetize the data exchanged as part of this framework
- Each public health authority may customize the parameters to define the way the app (based on ENF) may be used in that jurisdiction. This includes the way the app would determine the exposure risk (e.g., the minimum duration for which two devices must be near to each other to be recognized as an exposure event can vary between 5 and 30 min), the number of days beyond which the device data would automatically get deleted, the content of the notification message when an individual is identified by the app as a potentially exposed user, the look and feel of the app, etc.

### 3.4.3 Protocol Details

We describe the protocol in three parts as per the documented specifications—(1) the Bluetooth Specification, (2) the Cryptography Specification, and (3) the Framework API.

#### 3.4.3.1 Bluetooth Specification

The proximity identifiers are referred to as Rolling Proximity Identifiers (RPI) in this protocol. These identifiers are derived every day from a Temporary Exposure Key (TEK) and RPIs are changed every 15 min. TEKs are changed once every 24 h. The version of the protocol and the device’s transmit power are stored as the Associated Encrypted Metadata (AEM). AEM is also changed every 15 min along with the RPI. The exposure notification service is registered with the Bluetooth Special Interest Group (Bluetooth SIG) as a service 0xFD6F (Service UUID). The advertising payload contains 22-byte Service Data that includes 2 bytes of Service UUID, 16 bytes of RPI, and 4 bytes of AEM. The transmitted power level is encoded as part of the AEM and has a length of 1 byte.

We have already mentioned earlier that the BLE needs to operate in Broadcast Topology mode in ENF. The recommended broadcasting interval is 200–270 milliseconds, however, it can be changed. When a device acts as the observer, it receives the RPIs from nearby devices and stores the same along with the time-stamps and signal strength values (RSSI values). The recommended scanning
strategy is expected to be opportunistic where it should leverage the existing wake-up and scan intervals.

Whenever the app hands over the downloaded set of diagnostic keys (corresponding to the users who have been tested positive) to the Bluetooth layer, it picks up each of the stored RPIs and passes on that to the crypto layer along with each individual diagnosis key to determine if there has been any match or not. In the process of interaction to determine the exposure risk, the Bluetooth protocol layer acts as the intermediary between the app and the crypto API-layer.

3.4.3.2 Cryptologic Specifications

Time is discretized in 10-min intervals starting from the Unix Epoch Time and then every time interval is converted to a number \(i\) representing the count of intervals from that starting time. The TEK is generated as \(\text{TEK}_i = \text{CRNG}(16)\), where \(\text{CRNG}\) stands for a cryptographic random number generator and 16 stands for the number of bytes in the desired random number. TEK is regenerated after 144 time intervals (i.e., 24 h).

The Rolling Proximity Identifier Key (RPIK) is derived from TEK using the computation \(\text{RPIK}_i = \text{HKDF}(\text{TEK}_i, \text{NULL}, \text{UTF8}(\text{EN-RPIK})), 16\), where \(\text{HKDF}\) is a SHA-256 hash function as per RFC 5869. The Rolling Proximity Identifier (RPI) at a Unix Epoch Time \(j\) is derived from RPIK as per \(\text{RPI}_{i,j} = \text{AES128}(\text{RPIK}_i, \text{PaddedData}_j)\), where \(\text{PaddedData}_j\) is dependent upon the time interval value corresponding to \(j\).

Similarly, Associated Encrypted Metadata Key (AEMK) is derived from TEK using the computation \(\text{AEMK}_i = \text{HKDF}(\text{TEK}_i, \text{NULL}, \text{UTF8}(\text{EN-AEMK})), 16\). The Associated Encrypted Metadata (AEM) at a Unix Epoch Time \(j\) is derived from AEMK as per \(\text{AEM}_{i,j} = \text{AES-CTR128}(\text{AEMK}_i, \text{RPI}_{i,j}, \text{Metadata})\), where \(\text{AES-CTR128}\) is AES-128 block cipher in counter mode. The metadata can be decrypted by the receiving app at a later point of time for additional verification only after a match occurs between a proximity identifier value computed from the diagnosis key and the corresponding received RPI value.

3.4.3.3 Framework API

The framework API contains the following broad set of primitives:

- **ENStatus**: This signifies the overall status of the Exposure Notification System (the allowed values are “active,” “bluetoothOff,” “disabled,” “restricted,” and “unknown”)
- **ENAuthorizationStatus**: The authorization status can have values of “authorized,” “notAuthorized,” “restricted,” and “unknown”
- **ENRiskScore**: The risk score value (unsigned 1-byte integer)
- **ENRiskLevel**: It’s the estimated risk of user’s exposure (unsigned 1-byte integer)
• **ENManager**: Exposure Notification manager class including its API methods
• Some more classes and their associated APIs that help in getting the summary list of exposures, setting the configuration parameters related to exposure detection service, retrieving additional details of exposure events, etc.

Our idea of describing the above primitives is not to present an exhaustive listing of the APIs but instead to build a high-level understanding of how the Apple-Google Exposure Notification protocol is structured into its various components and layers. This understanding would help us while doing system analysis.

### 3.4.4 Highlights and Characteristics

We now look at the highlights and characteristics of this specification.

**It's the user who is in charge**: One of the most prominent characteristics of the protocol and the framework is that it has weaved in user’s consent at different stages of its operation. For the app to run properly, the user has to explicitly provide consent to enable the Exposure Notification Service in his/her device. The exposure risk level (or risk-score) is computed at the device end and if the user is diagnosed positive, it is only after he/she agrees the diagnosis keys from his/her device would get uploaded to the diagnosis server at the backend. The protocol does not require any personal or personally identifiable information of the user to run. Moreover, it can’t be utilized by any app that requires access to geo-location of the user. Users may decide to stop using this functionality at any point of time by turning off the Exposure Notification Service or by uninstalling the contact tracing app (at which point all local data also gets deleted). In fact, this functionality of the protocol of not sharing exposure event details with the backend server pushed quite a few countries (like the United Kingdom) to move away from the Apple-Google framework in favour of creating home-grown centralized solutions.

**Background mode of operation**: This is undoubtedly the strongest feature of this protocol. While using ENF, the DCT app can transparently run in the background (both in Android and iOS platforms) and thereby consuming very less power and creating a hassle free user experience. There is no other protocol or system at this point that can boast of this functionality. Perhaps that is the main reason why this framework is getting adopted at a fast pace by a large number of countries and states [5] including, Canada (COVID Shield), Denmark (smittestop), Germany (Corona Warn), Italy (Immuni), Japan (COCOA), Poland (ProteGO Safe), Saudi Arabia (Tabaud), Spain (Radar COVID), Switzerland (SwissCovid), etc. as well as many states in the US.

**Further integration at the OS level**: Google and Apple have already integrated the service at the OS level in such a way that one may get the benefit of the functionality
even in absence of a specific DCT app. It has also paved the way for multiple DCT apps that are based on this API framework to inter-operate.

**Modular architecture:** The entire framework has been developed in three layers—Bluetooth layer, Cryptography layer, and Framework API layer. This modularization not only helps the system to be used by multiple apps as per country-specific implementation, but any of the three layers can be easily customized and/or replaced if there is any such need. For example, if the device manufacturers come up with a physical communication protocol that is better than BLE, ENF may simply replace the Bluetooth layer of implementation with the new physical/link layer protocol. Similarly, if different cryptographic primitives are to be used for better security or faster computation, etc., the Cryptographic layer can be replaced with a new implementation. The apps would continue to function even if there happens to be any such change in the underlying layers.

### 3.4.5 System Analysis

We first present the security analysis and then concentrate upon the architecture analysis of the systems. We point out the possible open areas within each of these parts.

#### 3.4.5.1 Security Analysis

The specifications corresponding to each layer (BLE, Cryptographic, and API) have already been outlined in the Protocol Details subsection. Hence, we directly get into the discussion of how the framework may respond to the different attack scenarios.

- **Backend impersonation:** Interestingly, the current protocol specification is entirely centered around the user-app. The backend implementations are left to be defined by the individual administrative authorities of the countries or states that decide to build their systems using the Exposure Notification Framework. Hence, the possibility of backend impersonation may not be ruled out and it would depend on the level of security and certification followed for the server that acts as the diagnosis key repository. A malicious backend server can cause harm in various ways (e.g., by sharing bogus diagnosis keys with the apps and thereby generating large scale False Negatives). To mitigate this attack the approval process of the app (from the designated app stores) must include checks and balances on the server certification, domain name registration, security protocol used in the communication channel between app and backend server, etc.

- **Server data breach:** Unless the country or state-specific implementation of the framework expects the user to share any personal or personally identifiable information at the time of registration, the backend server(s) would not have any such
sensitive data elements. Hence, in the event of data breach at the server end, there would be no serious risk of losing sensitive data. However, there exists a considerable risk of deanonymization of infected users who have uploaded their diagnosis keys to the server.

- **Replay attack:** This could be feasible within the 10-min time-interval of an epoch since the RPI value remains constant in such an interval. To bring further granularity of time interval beyond which Replay Attack can be neutralized, Associated Encrypted Metadata (AEM) field could be used to accommodate some kind of time-stamp as well.

- **Relay attack:** This attack is feasible.

- **Deanonymization of users:** Since the diagnosis keys of the infected users are directly downloaded by every user-app, deanonymization can be masterminded by malicious users or user groups. Such users may arrange to passively listen to RPIs of users (along with additional information like captured images of people who would have come near the RPI capturing devices), especially in places where the density of infected users is expected to be high (like in hospitals) and matching those against the RPIs generated from the diagnosis keys that are downloaded from the backend server. One-entry attack can also lead to deanonymization.

- **Inverse-Sybil attack:** Since the system does not verify the actual device or SIM card number to authenticate the messages between the app and the server, the inverse-Sybil attack is feasible.

- **Coercion threats:** An infected person’s device can be misused by a malicious third party (before he/she uploads the proximity data to the server) by deliberately bringing that device in proximity to a large number of people and thereby triggering many false alarms in the ecosystem.

- **Miscellaneous:** The framework can be susceptible to Denial of Service (DoS) attack and many other generic mobile device’s OS or API layer attacks.

### 3.4.5.2 Possible Open Areas

In the previous subsection, we have talked about the possible attacks to which the Exposure Notification Framework appears to be susceptible as per current specifications. In addition, there are certain areas of the protocol and the APIs that may require further investigation. These are as follows:

- There is no mention of message authentication at the BLE protocol layer. Hence, it is not clear how the framework recommends the handling of message tampering.

- The server to server communication across countries or states may need to be handled in an ad-hoc fashion. At this point, there is no uniform clear specification on how the backend servers may communicate with each other in a federated system where multiple states of a country (e.g., the USA) may go for different implementations while following the same framework as in such a case there would be a need for people to move across state borders without losing the benefit of the system.
In Sect. 3.4.4, we have covered architectural elements. We now describe some of the possible open areas.

- There are insufficient details on how ENF expects the user registration process to happen. If a user’s app crashes and the user registers afresh to the system after installing the app again, would the previous user-id be linkable to the new user-id? If not, would the previous history of exposure be lost forever?
- The calibration of signal strengths for different manufacturers’ devices in the broadcast topology can be a critical success factor for the system to be effective—the calibration process (or any reference signal strength data) has not been shared along with the specification.
- The specifications talk about metadata like signal strength or duration being used in the risk-scoring algorithm, but it has not elaborated on how that would be captured.

### 3.5 East Coast PACT

East Coast PACT (Private Automated Contact Tracing) protocol has been jointly developed by Massachusetts Institute of Technology’s (MIT’s) Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts General Hospital Center for Global Health, MIT Internet Policy Research Initiative, MIT Lincoln Laboratory with additional collaborators including MIT Media Lab, Boston University, Brown University, Carnegie Mellon University, the Weizmann Institute, and other R&D centers. We refer to Version 0.1 of the protocol specification [25] that got released on 8 April 2020 and the Mission and Approach document [27] for our discussion. As per the PACT team, their protocol is largely similar to the Apple-Google Exposure Notification Framework, DP3T, and TCN proposals. It is licensed under the Creative Commons Attribution 4.0 License.

#### 3.5.1 Framework and Protocol Details

In this framework, there are primarily two protocol layers—(1) the chirping layer and (2) the tracing layer. We now describe the protocols for both the layers.

**Chirping Layer:** The proximity identifiers (referred as “chirps”) are 28 bytes long and these are generated from certain “seed values.” The seed values (which are random 32-byte values) are generated every hour and stored in the device along with the current time. If \( r_t \) denotes the chirp value at time \( t \) (measured to a precision of 1 min), and \( s \) is the seed value in that hour, then \( r_t = \text{PRF}(s, t) \), where \( \text{PRF} \) is a pseudorandom function. The chirps are sent every few seconds over the BLE interface to nearby devices in broadcast topology mode.

When a device receives a chirp from another device, it stores the same in the “contact log” along with additional information of signal strength and location-
specific information (optionally). The time is measured in this protocol with a 1-min precision and therefore if multiple chirps are received from a device within a minute, only one such chirp and the maximum signal strength (as a proxy for the distance between the devices) would be stored. The protocol also expects every implemented system to store the seed values corresponding to the chirp values sent by the device along with the time when it was sent. The specification recommends keeping the logs in the device for 3 months. It must be noted that when chirps are sent, no metadata is attached along with that value. As per the current specification, there is no provision to store additional information—like whether the other person is wearing a mask or any protective gear, the type of location (e.g., open space or closed space), etc., in the contact log.

**Tracing Layer:** This layer helps a user-app to determine the risk-score of getting infected. Whenever a user is diagnosed positive by a diagnostic center, a valid permission number (that may be used only once) is given by the test center to the user which would allow him/her to upload the seeds to a backend server (& database) corresponding to the chirps that his/her app have been sending over a period of time for which the user could have been contagious. The uploading of seeds instead of chirp values helps in optimizing the size of the transmitted data to the backend server. The upload format follows a tuple structure like \((s, t_1, t_2)\), where \(t_1\) and \(t_2\) signify the start and end time of the hour for which the seed was valid. The choice of upload lies with the user and even during the uploading process, the user may decide not to upload the seed values of certain hours due to privacy reasons in which case the app would allow the user to redact certain entries by generating fresh random numbers for those durations.

To check if the user is at risk of being infected, the user-app can download the exposure database (in case the database is geographically divided, only a part of the database may need to be downloaded), generate the chirp values corresponding to the seed values, cross-check the chirp values with its contact log to see if there is any match and if the time-window is approximately the same. The number of matches with chirp values generated from a particular seed value would reveal the duration of the event and the metadata of signal strength may reveal the average distance between the two devices. These data points are important to detect possible contact events where the user could have been exposed to other users who have been diagnosed positive later. The number of such contact events together can be used to derive a risk-score and, at this stage, there is a possibility of also exposing the granular details about the contact events to the user. The specification advises that users eventually need to consult medical professionals who can decide based on the risk-score and other data points (like user’s health condition, symptoms, location, etc.) and recommend the appropriate next steps to the user.

The specification has also considered the possibility of implementing chirp repeaters with a delay to handle the surface transmissions. For different reasons (e.g., for spotting the concentration zones and the nature of spread of the disease) if the location needs to be stored as metadata in the chirps, the protocol suggests a mechanism through which it may remain encrypted in the local device storage with
keys derived from the seed values of the contacts and therefore the deciphering would be feasible only for the matched entries and for no other cases.

### 3.5.2 Highlights

#### 3.5.2.1 The PACT Initiative

The PACT team has embarked on four major lines of effort:

- **Proximity detection efficacy**: Experimenting and objectively evaluating the BLE performance to determine the criteria for two devices to be “Too Close For Too Long (TC4TL)” as per epidemiological and medical guidelines and sharing the gathered results and insights with Public Health Authorities (PHA), with Apple-Google team and other teams for better decision making. The PACT team also plans to start investigating other signaling technologies (e.g., ultrasound, UWB, etc.)

- **Privacy**: Assess privacy impact of the integration of digital contact tracing systems with public health systems, study the developments of Apple-Google framework and gather insights with respect to the actual experience of users vs privacy-preserving goals of these systems.

- **Integration**: System architecture study of integrating multiple systems across geographical regions and advising Public Health Authorities based on the findings.

- **Public health efficacy**: To study how these systems can bolster the efforts of manual contact tracing and show measurable progress of controlling the pandemic over time.

The PACT stack is divided into three parts and it has focused on addressing challenges related to each of the parts separately.

- **Layer 1**: Proximity measurement
- **Layer 2**: Private cryptographic protocol
- **Layer 3**: (a) Public health interface and (b) Individual interface

The PACT team is focused on cross-layer activities that include

- Prototype building
- System analysis
- Data collection and experimentation
- Develop and roll out pilots for select organizations

#### 3.5.3 Design Principles

- Designed with simplicity of understanding and implementation in mind
- Compatibility with other decentralized protocols have also been a key factor of the specification
3.5 East Coast PACT

- It has been designed with the principles of autonomy, openness, and transparency (including the measures of graceful “sunset” at the end of the pandemic) and to uphold the values of user’s privacy and civil liberties without compromising the social health issues
- The size and rate of chirps have been designed in such a way that a wide range of device/OS combinations can be handled
- It can be easily integrated with Apple Find My protocol.

3.5.4 System Analysis

The system analysis of this protocol has a considerable overlap with the system analysis of the Apple-Google framework—hence we refer to the same in most of the places.

3.5.4.1 Security Analysis

We directly get into the discussion of how the framework may respond to the different attack scenarios.

- **Backend impersonation:** This would be identical to the analysis presented in case of Apple-Google framework.
- **Server data breach:** This would be similar to the analysis presented in case of Apple-Google framework, except here the seed values allow generation of all the chirps that have been emitted in an hour and hence the possibility of deanonymization of some of the infected users or creation of contact graphs corresponding to such users would be even more.
- **Replay attack:** This could be feasible within the 1 min time-interval within which the chirp value remains constant.
- **Relay attack:** This attack is feasible.
- **Deanonymization of users:** Since the seed values of the infected users are directly downloaded by every user-app, deanonymization can be masterminded by malicious users or user groups in a way similar to the attack described in case of Apple-Google framework. One-entry attack can also lead to deanonymization.
- **Inverse-Sybil attack:** Possible. This would be identical to the analysis presented in case of Apple-Google framework.
- **Coercion threats:** Possible. This would be identical to the analysis presented in case of Apple-Google framework.
- **Address carryover attacks:** This is possible as nothing has been mentioned of how to prevent the same in BLE layer.
- **Miscellaneous:** Possible. This would be identical to the analysis presented in case of Apple-Google framework.
3.5.4.2 Possible Open Areas

Apart from the vulnerabilities to the attacks mentioned in the previous subsection, the additional open areas of the protocol are as follows.

- The distinction between the responsibilities of different layers (API layer, cryptographic layer, and BLE layer) has not been mentioned.
- The details about the actual cryptographic/coding primitives and algorithms used have not been included.
- There is no mention of message authentication at the BLE protocol layer. Hence, it is not clear how the framework recommends the handling of message tampering.
- There is no clear specification on how the backends may communicate with each other in a federated system of servers.

The possible open areas in architectural description need to be enumerated too.

- The user registration process has not been described.
- The fault tolerance aspect of the framework at device end or server end has not been outlined.
- The calibration of signal strengths for different manufacturers’ devices in the broadcast topology can be a critical success factor for the system to be effective—the calibration process (or any reference signal strength data) has not been shared along with the specification.
- The specifications talk about metadata like signal strength or duration being used in the risk-scoring algorithm, but it has not elaborated on how that would be captured.

Next, we concentrate on West Coast PACT.

3.6 West Coast PACT

Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing (PACT) is a protocol that has been proposed by a team of technologists and researchers associated with the Paul G. Allen School of Computer Science and Engineering, University of Washington Medicine and Microsoft, through a white paper [28] published on 7 April 2020. The corresponding Github repository [8] contains a reference implementation of the same. As noted in the East Coast PACT specification document [25] that there are a lot of similarities in the approach and structure between the two proposals.

3.6.1 Framework and Protocol Details

One important element, that is different in the framework as compared to other decentralized systems, is that it allows the seeds of the infected users to be either
published by the hospitals post their validation and confirmation or alternatively self-reported by the users who are diagnosed as positive. We now describe the protocol design in brief.

There are two basic parameters—a number $\Delta$ and a unit of time $dt$ such that $\Delta \cdot dt$ (product) equals the infection window as guided by the epidemiologists (which could be typically 2 weeks). The pseudorandom IDs that are emitted by a device are 128-bit long. The $i$th ID ($ID_i$) is broadcast during the $i$th time-window where the first time-window is assumed to have started from the time the user-app got initialized (say $t_0$). The protocol samples an initial 128-bit seed (called $S_0$) and each successive seed ($S_{i+1}$) as well as the successive pseudorandom ID ($ID_{i+1}$) is generated using the following step

$$(S_{i+1}, ID_{i+1}) \leftarrow G(S_i),$$

where $G$ is a cryptographic pseudorandom generator using a hash function SHA-256. The first ID is $ID_1$, as the entire procedure starts with $S_0$. When a device receives a pseudo-random ID from a neighboring device, the app stores it in the local storage with a time-stamp as a pair ($ID, t$). At the time of reporting, the infected user needs to upload the seed value ($S^*$) that corresponds to the time-window at the start of the infectious period (2 weeks back), the starting time ($t_{start}$), and the ending time ($t_{end}$). This tuple gets added to the list of exposed seed values in the public server. These entries can be validated by attaching signature $\sigma$ and (optionally) a certificate ($cert$).

To check exposure, a user-app needs to download all such exposed seed value tuples, derive the corresponding ID-values for the durations indicated in the tuples, check for any matching with the local storage of received IDs, and compute the risk of exposure. The time-stamps are used for validation of the possible exposure events (e.g., to prevent relay attacks).

### 3.6.2 Highlights

The protocol minimizes the information a user needs to share with the central server. The users cannot upload any information to the server (before being or) once outside the infection period. It recommends that the app should upload the infection information to the central server with a slight time delay, which helps in reducing the chances of replay attacks. The protocol also prevents leakage of information regarding the app-joining date by fixing the length of the ID-sequence to a value $\Delta$. Additionally, strong integrity of the server data can be added to the existing mechanism by introducing an authentication-verification process to the uploading procedure using a pair of keys ($sk, vk$).

The white paper [28] discusses the ease of interoperability of decentralized protocols similar to East Coast PACT, West Coast PACT, and DP3T due to the same basic principle of uploading seeds to a server. However, it also points out that the only difficulty that may arise is due to differences in the reliability of privacy-protection of users uploading this data. This can be addressed by introducing a common API.
The ethics of achieving a balance between the privacy of users and transferring information to the authorities in the interest of safety of the larger population are also discussed. These considerations are likely to differ greatly between different countries. Finally, the accessibility of these services depends most importantly on the number of people owning a smartphone and installing the app, which greatly affects their effectiveness.

### 3.6.3 System Analysis

The system analysis of this protocol is almost identical to that of East Coast PACT due to the similarity between the two proposals.

#### 3.6.3.1 Security Analysis

- **Backend impersonation:** Possible and identical to the situation as in East Coast PACT or Apple-Google Framework.
- **Server data breach:** The risk and extent of deanonymization in this system are much more than that of any other decentralized protocols since here the starting seed value ($S^*$) is sufficient to generate all the ID values that have been broadcast by the infected user’s app within the window of infection. There is no other risk of disclosure as no personal or personally identifiable information of users gets stored in the public server.
- **Replay attack:** This could be feasible for the $dt$ time-interval within which the broadcast ID values remain constant.
- **Relay attack:** This attack is feasible.
- **Deanonymization of users:** The risk and extent of deanonymization in this system are much more than that of any other decentralized protocols since here the starting seed value ($S^*$) is sufficient to generate all the ID values that have been broadcast by the infected user’s app within the window of infection. This weakness also risks other data such as time of exposure or location of the infected user to be released. The possible mitigation provided in [38] involves only releasing the IDs and not seeds from the uploaded data, which still does not prevent dedicated adversaries from establishing a link.
- **Inverse-Sybil attack:** Possible and identical to the situation as in East Coast PACT or Apple-Google Framework.
- **Coercion threats:** This would be identical to the situation as in case of East Coast PACT or Apple-Google framework.
- **Address carryover attacks:** This is possible as nothing has been mentioned of how to prevent the same in BLE layer.
- **Miscellaneous:** Same as in East Coast PACT or Apple-Google framework.
3.6.3.2 Possible Open Areas

Apart from the vulnerabilities to the attacks mentioned in the previous subsection, the additional open areas of the protocol are as follows.

- The distinction between the responsibilities of different layers (API layer, cryptographic layer, and BLE layer) have not been mentioned. The specification does not provide the details of how to configure and use the BLE layer and how to make it compatible with different platforms and device manufacturers.
- Although the specification talks about the need for a common API to ensure interoperability among multiple decentralized systems, the mechanism of interoperability between multiple public servers using the same protocol for different states/countries or the details of the common API have not been described.

There are also certain open areas in architecture.

- The user registration process has not been described.
- The fault tolerance aspect of the framework at device end or server end has not been outlined.
- The specification does not outline the risk-scoring algorithm. It also does not specify the way to calibrate the signals emitted by the devices which can be used as a proxy to measure the approximate distance between two nearby devices (e.g., in the specification it talks about a distance of 6m to be used as the expected proximity).

3.6.4 A Re-Randomized Version

The authors of [28] have also proposed an alternative dual approach, where infected users, on authorization from the health authority, upload to the server not their broadcasted identifiers, but the received ones. A small advantage gained in this dual approach is the restriction of deanonymization and linkage attacks by the server if the received identifiers are uploaded in an encrypted form. However, it becomes very easy for malicious users to achieve replay, relay, inverse-Sybil, and deanonymization attacks in this case (say $C$ comes into contact with $B$ and sends the received identifiers to $A$ for upload when $A$ is diagnosed positive). The broadcast and upload of identifiers are proposed to be executed through the decisional Diffie-Hellman (DDH) protocol, by choosing a cyclic group $G$ with a generator $g$ known to all entities, and each user choosing one’s own secret key $s_U$. On contact with another user $B$, the device of user $A$ receives and stores the pair $(g^{r_i}, g^{r_i s_B})$. On positive diagnosis, $A$ gets authentication from the health authority and uploads all pairs $(x, y)$ received from such contacts to the server.

A possible mitigation for deanonymization attacks is for $A$ to choose a random element $r \in G$ and upload $(x^r, y^r)$ to the server instead of the received values. A user $B$ wanting to know one’s risk status can simply check whether $x^{s_B} = y$ for any of the
pairs downloaded from the server. This would prevent $B$ from deanonymizing $A$ in spite of (say) recording data such as a list of one’s contacts at the time the particular identifier was being broadcasted.

3.7 Temporary Contact Numbers (TCN)

Temporary Contact Number (TCN) is one of the earliest decentralized protocols that was introduced [9] on March 17 2020 by a group of researchers, engineers, private and public health experts through a collaborative effort between Stanford University and the University of Waterloo. It has been developed by the TCN coalition that includes the Covid Watch [7] and the CoEpi [4] teams. The updated version of the specification and its corresponding open-source reference implementation is available in the Github repository [32].

3.7.1 Framework and Protocol Details

Since there is a lot of similarity of TCN with the previously described systems like Apple-Google Framework, DP3T, East Coast PACT, and West Coast PACT, we would present it at a high level by primarily highlighting the specific features that are not observed in the other decentralized protocols.

Here, the proximity identifiers, referred as the Temporary Contact Numbers (TCNs), are generated as 128-bit pseudonymous pseudorandom numbers (at the device end) from a seed value. Like the rest of the decentralized systems, the reason for using the seed values for generating the TCNs, in contrast to the direct generation of pseudorandom numbers, is to optimize the size of data upload at the time of reporting (by users who are later diagnosed positive). We now describe the details of the steps through which the TCNs get generated.

At first, a pair of keys, called the Report Authorization Key (RAK) and the Report Verification Key (RVK) get generated using the Ed25519 signature scheme [21]. An initial Temporary Contact Key (TCK$_0$) is created using a SHA-256 hash function (called H_TCK), where TCK$_0 = H$ TCK(RAK). This initial TCK is used to generate a series of TCKs (TCK$_1$, TCK$_2$, . . . , etc.) which in turn generate a series of TCN values (TCN$_1$, TCN$_2$, . . . , etc.) using another SHA-256 hash function (called H_TCN).

\[
TCK_i = H_TCK(RVK \ || \ TCK_{i-1}) \quad \text{and} \quad TCN_i = H_TCN(\text{le}_u16(i) \ || \ TCK_i) \quad \text{for every integral } i > 0.
\]

The report generation process is extremely compact in TCN. The core idea comes from the observation that if RVK and a starting TCK value is provided along with the index values that correspond to the starting and ending TCN values, the receiving user-app can easily derive all the required TCNs. The specific details are now given
below.

\[
\text{REPORT} = (\text{RVK} || \text{le}_u(\text{s}) || \text{le}_u(\text{e}) || \text{memo}) \quad \text{and}
\]

\[
\text{SIGNED\_REPORT} = \text{REPORT} || \text{SIG}
\]

In the above calculations, \( s \) and \( e \) are the starting and ending indices (corresponding to the epochs within which the user is considered to be infectious and usually this period is considered to be 14 days), ‘memo’ is a free-form implementation specific message (of the form \( \text{TAG} \| \text{LENGTH\_OF(DATA)} \| \text{DATA} \)) and \( \text{SIG} \) is the signature on the \( \text{REPORT} \) (based on \( \text{RAK} \)) that would serve the purpose of authentication.

The specification has mentioned that a user-app should download the signed seed reports of the infected users on a daily basis from the server and evaluate the extent of exposure of the corresponding user.

The “encounter handshake” part of the protocol between the devices is implemented in both broadcast topology as well as through connected topology so that Android and iOS-based devices (running in either foreground or background mode) can be supported. The “infection reporting” part of the protocol is implemented using a secured HTTPS channel.

### 3.7.2 Highlights

- Like the rest of the decentralized protocols, no personally identifiable information is required at any stage for the protocol to function
- The protocol can work with either verified test results or self-reported symptoms
- Typically a signed report would be 134–389 bytes long
- The pair of keys RAK and RVK should be periodically rotated so that the TCN history does not form a single long reproducible chain. However, if it changes too soon, the scalability of the protocol may get impacted. In that case, the report upload also should be done in separate chunks and the specification has suggested a duration of not more than 6 h for every report.
- The design of the protocol has kept in mind the following needs:
  - Cross User-app and cross-platform (iOS and Android) device interaction
  - Not asking the user for location access from his/her device
  - Efficient usage of BLE power
  - Workability even in the background mode of the app
  - The duration for which one TCN value remains constant is a configurable parameter. It is recommended that it should be aligned with the change of MAC value as well to avoid linkage attacks—however, the specification has pointed out that the current implementation is not guaranteed to achieve that in Android or iOS due to the inherent issues of the operating systems.
3.7.3 Security and Architecture Analysis

- **Backend impersonation**: Like most of the decentralized protocols (like Apple-Google Framework, DP3T, East Coast PACT, etc.) TCN is also susceptible to this attack.
- **Server data breach**: Susceptible to this attack in a way similar to East Coast PACT.
- **Replay attack**: Susceptible to this attack in a way similar to East Coast PACT.
- **Relay attack**: This attack is feasible.
- **Deanonymization of users**: Susceptible to this attack in a way similar to East Coast PACT.
- **Inverse-Sybil attack**: Like most of the decentralized protocols (like Apple-Google Framework, DP3T, East Coast PACT, etc.) TCN is also susceptible to this attack.
- **Coercion threats**: Like most of the decentralized protocols (like Apple-Google Framework, DP3T, East Coast PACT, etc.) TCN is also susceptible to this attack.
- **Address and Shared Carryover Attacks**: TCN is susceptible to both these attacks and the specification has also talked about these possibilities.
- **Miscellaneous**: Possible. This would be identical to the analysis presented in case of Apple-Google framework.

3.7.3.1 Possible Open Areas in Security

Apart from the vulnerabilities to the attacks mentioned in the previous subsection, the additional open areas of the protocol are as follows.

- The distinction between the responsibilities of different layers (API layer, cryptographic layer, and BLE layer) has not been mentioned.
- There is no clear specification on how the backends may communicate with each other in a federated system of servers.
- There is a significant detailing done for cross-platform usage. However, no guidelines or experience sharing has been done for cross-app usage.
- The validation of reporting stage through Health Authority has not been explained enough.

3.7.3.2 Possible Open Areas in Architecture

- The user registration process has not been described.
- The fault tolerance aspect of the framework at device end or server end has not been outlined.
- The calibration of signal strengths for different manufacturers’ devices in the broadcast topology can be a critical success factor for the system to be effective—the calibration process (or any reference signal strength data) has not been shared along with the specification.
- There is no mention of how epidemiologists may get help from the system by tracking the progress of the disease, its transmission mechanism, or the dynamic nature of spread of infection across multiple locations.

3.8 The Epione Protocol

A team of researchers from UC Berkeley and the National University of Singapore has proposed the **Epione** protocol [36] for contact tracing and preventing the spread of SARS-CoV-2. The *registration phase* of this protocol begins with users installing the app and generating random tokens using a secret key $\kappa$, which are broadcast through Bluetooth in the *contact-broadcast phase*. Users store the received tokens on a local list on their devices. The health authority is considered to be a trusted entity and is responsible for identifying infected users and reporting the PRG seeds used to generate their broadcasted tokens to the server. The seeds are forwarded by the infected users to the health authority after encryption using a public key $\text{pk}$ (of a public key-secret key pair $(\text{pk}, \text{sk})$ generated by the server), which the health authority forwards to the server. The server is also trusted and can decrypt these seeds using its secret key, and use them to generate and store the broadcasted tokens in a (private) list. This constitutes the *reporting phase* of the Epione protocol, after which, the *exposure calculation phase* can take place when a user and the server securely compare the user’s received tokens with the list of tokens broadcasted by infected users stored on the server with a matching algorithm (that uses a secure two-party private set intersection cardinality (PSI-CA) computation). Although the central server has access to all the tokens received by any user, it is difficult for it to compute the neighborhood of any infected user (considered as a vertex of the proximity graph) due to the unavailability of tokens received by undiagnosed users. Therefore, Epione is a decentralized contact tracing protocol according to our classification.

3.8.1 Private Set Intersection (PSI)

Epione uses a secure two-party private set intersection cardinality (PSI-CA) computation to assess the risk exposure of querying users, by determining the size of the intersection set of two lists, one containing the tokens broadcasted by infected users saved in the server database, and another containing the tokens received by a user queried to the server. A user wanting to know one’s risk has full access to one’s own list but does not have direct access to the list stored on the server. The server, on the other hand, does not have direct access to the tokens received by the querying user. Algorithms that compute information about the intersection of two sets without complete knowledge of the elements in at least one of the two sets are called *private set intersection (PSI)* algorithms.
PSI algorithms could be of different types, based on the information computed about the intersecting set. These types range from computing the complete intersection to only finding the size of the intersection to simply stating whether intersection occurs or not. Suppose $A$ has a set $\{x_1, \ldots, x_n\}$ and $B$ has a set $\{y_1, \ldots, y_m\}$, which they do not wish to fully reveal to each other. The simplest way to find the intersection between these two sets is to apply a variation of the Diffie-Hellman key-exchange protocol on each element of both the sets [23]. $A$ and $B$ agree on a hash function $H$ (but do not share the hash key they choose). $A$ chooses an element $a$ belonging to the underlying field and computes $\{H(x_1)^a, \ldots, H(x_n)^a\}$, while $B$ chooses an element $b$ and computes $\{H(y_1)^b, \ldots, H(y_m)^b\}$. They exchange these lists, compute $\{H(y_1)^{ab}, \ldots, H(y_m)^{ab}\}$ and $\{H(x_1)^{ab}, \ldots, H(x_n)^{ab}\}$ respectively and again exchange these two lists.

If each participant randomly permutes one’s list in both the exchanges, then they only know the size of the intersection. If the lists are randomly permuted only in the first exchange, they would know the positions of the elements that are present in the intersection set but not their values. If the lists are randomly permuted in the second exchange, they would know the exact values that are common in both the lists. All these cases are subject to the collision properties of the hash function $H$, which, if not used, would directly reveal the complete sets to both participants.

The particular case of revealing only the size of the intersection set is called a private set intersection cardinality (PSI-CA) algorithm.

### 3.8.2 Highlights of the Epione Protocol

Epione stands out from centralized protocols through the token generation process, which takes place locally on the user-apps. The tokens received from other users are also stored locally on the users’ devices. However, it is also different from decentralized protocols due to the absence of any bulletin board maintenance by the server. Instead, users query their received tokens to the server, which computes their exposure risks. Since it implements the token-matching algorithm through a PSI-CA computation, the server only gains knowledge of the size of the intersection of the set of received token queried by a user and the set of tokens of infected users reported by the health authorities. This prevents the server from gaining information about any particular user, thus restricting a neighborhood-construction of any infected user in the proximity graph. It seems to be satisfactorily secure against most semi-honest adversaries.

### 3.8.3 Possible Cryptographic Issues with the Epione Protocol

Although this system may prevent attacks from malicious users (and possibly other non-user, non-authority entities) due to centralization, it places complete trust in the
health authority and partial trust in the central server. A corrupted health authority may quite easily carry out attacks involving false-positive claims. A server colluding with the health authority equipped with only a weakly secure implementation of a PSI-CA computation for its matching algorithm could also carry out (possibly) complete linkage attacks. These attacks have been mitigated in [36] by stressing on the trustworthiness of these two entities and on the requirement of not allowing them to collude.

Malicious adversaries with abilities to intercept or read various transmitted data could also launch quite strong attacks on the system. For example, if an adversary gains access to the communication between a group of users querying their received tokens to the server and the server’s corresponding answers, it could work out at least an incomplete neighborhood of infected users whose tokens were shared with multiple users of this group. Such an adversary might also track users to a large extent. Arbitrary queries by malicious adversaries have been proposed to be mitigated by use of a cryptographic hash on the tokens queried, although it must be noted that this is not a feature of the original system proposed.

### 3.9 An Approach to Avoid Inverse-Sybil Attacks

The work [34] proposes an interesting protocol to avoid attacks on these protocols similar in nature to inverse-Sybil attacks. It makes use of three hash functions and requires each user to maintain a hash-chain to validate one’s uploaded identifiers. The first two stages of the protocol are divided into epochs, at the start of which, the process begins afresh. After registration, the setup phase begins when each user chooses a hash key \( k \) and a starting (current) hash-chain head value \( h \). The first hash function, \( H_1 \) is used to generate subsequent hash-chain heads for each epoch. Next, the broadcast phase includes the following steps. When an epoch starts, a user samples a random value and broadcasts it along with the current hash-chain head. A user also receives such pairs from any other users in one’s vicinity. The received random values are stored in a set \( C \), which is constructed by hashing the triple of the user’s current hash head, the received hash head and the received random value by means of the second hash function \( H_2 \), and the sent random values are stored in an evaluation list \( L^{\text{eval}} \) by hashing the triple of the received hash head, the user’s current hash head and the sent random value by means of the third hash function \( H_3 \). A report list \( L^{\text{rep}} \) is created by recording pairs of each set \( C \) and the corresponding hash-chain head, which, in addition to the user’s hash key \( k \), would be reported to the server in the reporting phase if the user is tested positive for SARS-CoV-2. The server only proceeds if it can check that the hash-chain heads in the report list uploaded by a user indeed form a hash-chain. In this case, it stores the hashed values \( H_3(h_i, \sigma) \), for each \( \sigma \in C_i \), for each \((h_i, C_i) \in L^{\text{rep}}, \) in the server list \( L^{\text{ser}} \). In the exposure calculation phase, a user wanting to check one’s exposure status can download (the full or a part of) \( L^{\text{ser}} \) and check whether any entries match with its \( L^{\text{eval}} \). The user is considered at risk if any such matches exist.
This is clearly a non-interactive decentralized protocol as users need not interact with each other on coming into contact and the only broadcast of identifiers suffices, and since the central server does not get any information about the contacts of infected users and therefore cannot (honestly) construct their proximity graphs. A second protocol, which is decentralized and instead uses location data for chaining is also discussed in [34]. It is very similar to the one described here, but instead of simply storing hashed values of the hash-chain heads and random values broadcasted or received, a user’s location and time coordinates are also stored in the $L^{rep}$ and $L^{eval}$ lists on the device. This may cause loss of privacy (depending upon the security of the hash functions), but increases the integrity of the system, further tightening security against inverse-Sybil, replay, and relay attacks.

### 3.10 Conclusion

The decentralized contact tracing protocols discussed here all share key elements such as the minimization of information that a user needs to share with the central server, this uploading of information being a voluntary action that a user may take only on testing positive for SARS-CoV-2, and in no other case. The server broadcasts this information in a manner that maintains the anonymity of all infected users who have uploaded their IDs, for only the time period that these users may be deemed infectious. These users also cannot upload any information to the server (before being or) once outside the infection period. A user wanting to check one’s status of exposure need not share any information with the server at all and can compute this risk locally on the app.

All these protocols may eventually allow interoperability between different apps due to the similarity in their approaches. However, it must be noted that difficulties may arise due to differences in the reliability of privacy-protection of users uploading this data. This can be addressed by introducing a common API. Some unavoidable issues such as immediate identification of an infected user (say A) by another user (say B), if A is the only user to have established contact with B, are present in the systems. There is complete privacy for users who have not tested positive for SARS-CoV-2, but some privacy loss occurs whenever a user uploads one’s keys or identifiers to the server.

It may be worthwhile to expand the current scope of these systems by considering ways to perform contact tracing (with explicit approval from users) to secondary and tertiary levels and have a much broader view of the spread by analyzing the secondary or tertiary proximity graphs combined with the compartmental model of the infection system dynamics used by epidemiologists. Another question to ponder is the extension of the system’s functionality from a human-to-human transmission alerting function to flagging potential infectious-surface-to-human transmission events (the possible third way of transmission through aerosol particles may not be feasible to be captured by the current level of technology). It may also be worthwhile to investigate possible alternate implementations of the systems that do not require BLE or
the extension of some of the functionalities in feature phones as well. Furthermore, a user may want to perform certain “what-if” analyses of routes (combined with the GPS location map app) to get an apriori idea of the risk involved in traveling on a specific route. It would be of interest to inspect whether this information can be provided to some extent, with extreme caution, as it would decrease the privacy provisions of the system, despite its usefulness. Finally, the accessibility of these services depends most importantly on the number of people owning a smartphone and installing the app, which would greatly affect their effectiveness.

References

1. Apple-Google. Privacy-preserving contact tracing. https://www.apple.com/covid19/contacttracing.
2. Apple-Google. Privacy-preserving contact tracing. Exposure notification FAQs. https://blog.google/documents/73/Exposure_Notification_-_FAQ_v1.1.pdf.
3. Castelluccia, C., Bielova, N., Boutet, A., Kunche, M., Lauradoux, C., Le Métayer, D., Roca, V. (2020). DESIRE: A third way for a European exposure notification system leveraging the best of centralized and decentralized systems. ffhal-02570382 https://hal.inria.fr/hal-02570382/document.
4. CoEpi Website. https://www.coepi.org/about/.
5. COVID-19 Apps Wikipedia Page. https://en.wikipedia.org/wiki/COVID-19_apps.
6. COVIDSafe app. https://www.health.gov.au/resources/apps-and-tools/covidsafe-app.
7. COVIDWatch Wikipedia Page. https://en.wikipedia.org/wiki/Covid_Watch.
8. COVIDSafe. https://github.com/covidsafe.
9. CovidWatch Article. https://www.covid-watch.org/article/.
10. CovidWatch Website. https://www.covid-watch.org/.
11. DP3T Whitepaper. https://github.com/DP-3T/documents/blob/master/DP3T%20Whitepaper.pdf.
12. DP3T – Aims. https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Aims%20of%20the%20Project.pdf.
13. DP3T – Overview document. https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Simplified%20Three%20Page%20Brief.pdf.
14. DP3T Documents. https://github.com/DP-3T/documents.
15. DP3T Android App. https://github.com/DP-3T/dp3t-app-android-ch.
16. DP3T iOS App. https://github.com/DP-3T/dp3t-app-ios-ch.
17. DP3T - Interoperability decentralized proximity tracing specification (Preview). https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Interoperability%20Decentralized%20Proximity%20Tracing%20Specification%20(Preview).pdf.
18. DP3T - Security analysis. https://github.com/DP-3T/documents/blob/master/Security%20Analysis/Privacy%20and%20Security%20Attacks%20On%20Digital%20Proximity%20Tracing%20Systems.pdf.
19. DP3T - Authorization analysis. https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Upload%20Authorisation%20Analysis%20and%20Guidelines.pdf.
20. DP3T - Exposure score calculation. https://github.com/DP-3T/documents/blob/master/DP3T%20-%20Exposure%20Score%20Calculation.pdf.
21. Edwards-curve digital signature algorithm (EdDSA) – Wikipedia Page, Subheading Ed25519. https://en.wikipedia.org/wiki/EdDSA#Ed25519.
22. Exposure Notification – Wiki. https://en.wikipedia.org/wiki/Exposure_Notification.
23. Huberman, B. A., Franklin, M., Hogg, T. (1999). Enhancing privacy and trust in electronic communities. In *EC ’99: Proceedings of the 1st ACM conference on Electronic commerce. November 1999* (pp. 78–86). https://doi.org/10.1145/336992.337012.
24. Issues raised in GitHub repository for DP-3T. https://github.com/DP-3T/documents/issues.
25. PACT protocol specification - version 0.1 (4/8/2020) https://pact.mit.edu/wp-content/uploads/2020/04/The-PACT-protocol-specification-ver-0.1.pdf.
26. PACT MIT Webpage. https://pact.mit.edu/.
27. PACT – Mission and approach. https://pact.mit.edu/wp-content/uploads/2020/05/PACT-Mission-and-Approach-2020-05-19-.pdf.
28. PACT: Privacy-sensitive protocols and mechanisms for mobile contact tracing. https://arxiv.org/abs/2004.03544.
29. Pietrzak, K. Delayed authentication replay and relay attacks on DP-3T. https://eprint.iacr.org/2020/418.pdf.
30. ROBERT: ROBust and privacy-presERving proximity Tracing. Inria, France and Fraunhofer AISEC, Germany. https://github.com/ROBERT-proximity-tracing/documents.
31. Statement on the processing of personal data in the context of the COVID-19 outbreak – European Data Protection Board. https://edpb.europa.eu/sites/edpb/files/files/file1/edpb_statement_2020_processingpersonaldataandcovid-19_en.pdf.
32. Specification and reference implementation of the TCN Protocol for decentralized, privacy-preserving contact tracing. https://github.com/TCNCoalition/TCN.
33. Techcrunch.com News Article. Apple and Google release sample code, UI and detailed policies for COVID-19 exposure-notification apps. https://techcrunch.com/2020/05/04/apple-and-google-release-sample-code-and-detailed-policies-for-covid-19-exposure-notification-apps/.
34. The Crypto Group at IST Austria. Inverse-Sybil attacks in automated contact tracing. https://eprint.iacr.org/2020/670.
35. TraceTogether Wikipedia Page. https://en.wikipedia.org/wiki/TraceTogether.
36. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D. Epione: Lightweight contact tracing with strong privacy. arXiv:2004.13293.
37. Vaudenay, S. Centralized or decentralized? The contact tracing dilemma. https://eprint.iacr.org/2020/531.
38. Vaudenay, S. Analysis of DP-3T between scylla and charybdis. https://eprint.iacr.org/2020/399.pdf.