Chapter 2
Centralized Systems

Abstract  Digital Contact Tracing (DCT) protocols and systems usually rely on close-range communications between handheld devices (like smartphones and tablets), primarily through a 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 as centralized or decentralized. In Chap. 1, we have discussed some of the competing ways of classifying systems, eventually settling with the following broad-level understanding: for the scope of this book, we shall consider a system (or a protocol) centralized if the backend server(s) plays (or play) a dominant role in deciding the risk level (or risk-score) of an individual being infected. This chapter focuses on centralized systems and protocols like BlueTrace, TraceTogether, COVIDSafe, ROBERT etc. It also presents a generalized framework of centralized systems before going deep into the specific systems. The chapter ends with discussions on systems that lie in between the centralized and decentralized categories, thus setting the stage for decentralized systems in Chap. 3.

2.1 Introduction

The central server of any digital contact tracing system may have access to certain personal or personally identifiable information (like phone number, postal code etc.) of the user. If these datasets are not stored in an encrypted form, there is a significant risk of loss of users’ privacy. Hence, all DCT systems and protocols proposed so far make use of pseudonymous identifiers. These identifiers can be of two types—one associated permanently with an instance of user-app right from the registration phase, and the other generated for the purpose of sharing proximity identifiers between two nearby user-apps/devices. If the latter type of identifiers does not change over time, a malicious entity may be able to track the movement of a particular user by capturing
the unchanging proximity identifier emitted by the user-app and then associating it with the user’s device through some other means (e.g. through photographic or video recording of the user in some location). In order to avoid tracking through similar methods, all DCT systems and protocols are designed to change their proximity identifiers frequently.

In most centralized systems, the central server is usually responsible for generating these identifiers. This (possibly accompanied by other capabilities it may possess) may provide the central server in a centralized system to become aware of the list of contacts of an infected user; a compilation of such data from multiple users can be utilized to generate a social graph of users. Central servers and databases must be provided the highest level of data security so as to avoid any leakage or compromise of sensitive user-data.

### 2.1.1 Background

In Chap. 1, we have described the rationale behind the division of digital contact tracing systems in two categories—centralized and decentralized. We have also pointed out that due to the multiple ways of defining centralized or decentralized schemes, some centralized systems have many decentralized features and likewise, almost all decentralized systems, with certain functionalities delegated to the backend server, could be viewed as partially centralized systems. With more and more DCT systems being proposed, one may expect the eventual emergence of a continuum between the two extremes of fully centralized and fully decentralized systems, irrespective of how one may choose to define the categories.

This chapter starts with a description of the centralized protocol BlueTrace, followed by descriptions of COVIDSafe, PEPP-PT NTK and Aarogya Setu, and eventually concluding with a detailed study of a protocol that lies in the middle of the centralized-to-decentralized continuum, called ROBERT, and its evolved successor DESIRE, which moves further away from centralization and towards the decentralized end of this continuum. In the absence of standard design specifications or a documentation on the cryptographic algorithms used for centralized systems like PEPP-PT, NHS_COVID-19 app etc., we have provided high-level descriptions instead of detailed system analyses.

### 2.1.2 Characteristics of Centralized Systems

Let us now discuss the typical characteristics of a centralized digital contact tracing system. Centralized systems are designed as a natural extension of the manual contact tracing process, in which human contact tracers reach out to the primary, secondary or tertiary contacts after interviewing the index cases and provide the con-
cerned individuals with guidance, health advice and counseling (wherever required). Centralized systems also act as the nerve center of the entire operation by managing, controlling and limiting the spread of the infectious disease. The common characteristics of almost all centralized systems are as follows.

- A centralized agency like the Health Department or the Ministry of Health assumes the authority on the entire digital contact tracing system (including the data stored in the server).
- The backend server(s) and central database(s) are viewed as trusted entities that can securely manage the user-specific data and maintain the privacy of the users.
- If a user tests positive, then he/she would be expected to upload one’s proximity identifiers (shared by one’s user-app with the neighbouring devices and/or received from such devices over the last 14 days or so) to the backend server(s).
- The backend server(s) help (either proactively or based on request) in the identification of contacts who could be at risk of infection.
- The contact data is used by the backend to either reach out to the person (if evaluated to be at risk of being infected) via a human contact tracer or update the corresponding risk level (or the risk-score) to the central database(s), which can be retrieved by the user-app from time to time.
- In case the security of the server data is compromised and sensitive data of users fall in the hands of malicious actors or groups, the extent of damage could be significant. This makes the minimization of collection of personal or personally identifiable data a critical evaluation parameter in centralized systems and protocols.
- Epidemiological analysis can be done easily from the data stored in the backend server.
- It becomes important to track how the backend data is stored by the service provider (like Google Cloud Services, Amazon Web Services etc.) in the data centers, so that no other entity (e.g. the government of another country) may have legal rights or control on the data.

**Extensibility beyond one country, jurisdiction or region:** Since the backend server acts as the nerve center of the entire system, it becomes easy to design a federated system of cooperating and trusting servers that may communicate with each other to manage the contact tracing process across regional or national boundaries.

**Ease of modification:** Since most critical operations including evaluation of the at-risk or not at-risk status of users are carried out centrally, it may become easier in centralized systems to roll out changes, bug-fixes and enhancements (e.g. modification of the risk-scoring algorithm) as compared to decentralized systems.

### 2.1.2.1 Common Cryptographic Primitives

We now take a look at some important cryptographic primitives in centralized systems.
• **Proximity identifiers.** Frequently changing pseudonymous and pseudorandom identifiers that are exchanged between the user-apps when any two devices come near each other.

• **Encryption and authentication keys.** These keys are required during device-to-device, device-to-server or server-to-server communications. They may be symmetric or asymmetric, and can be exchanged by following any interactive or non-interactive key-exchange protocol.

• **Permanent identifiers.** A permanent identifier may be uniquely associated with each user-app and may correspond to other personal or personally identifiable information (like a user’s phone number).

• **Other codes.** This may include an encrypted country code, encrypted epoch number (signifying a specific start and end time) etc.

### 2.2 A General Framework of Centralized Protocol

A general centralized protocol for contact tracing is mainly characterized by the interaction of various entities with the central server. User-apps receive proximity identifier(s) from the server (in some cases, users may locally generate these identifiers), which they broadcast or exchange with other user-apps when in proximity. An infected user’s app uploads one’s received (and/or sent) identifiers to the server after authorization from a health authority, and another user intending to check one’s exposure status may either wait to receive a direct notification from the server (or from a person in case of a human-in-the-loop system like TraceTogether), or query the server by passing its sent (and/or received) identifiers and determine the risk level (or risk-score) of its user as computed by the server. The central server(s) of a centralized system thus plays a more prominent role in comparison to the central server(s) of decentralized protocols.

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

1. **Registration and Initialization Phase.** At the time of registration and initialization, the user-app may receive the first set of proximity identifiers to be used (or the keys from which such identifiers can be generated) from the server in a centralized DCT system. It may continue to receive new sets of proximity identifiers in a batch-mode after fixed time-intervals (e.g. after every 24h).

2. **Contact-Broadcast Phase.** There is no difference between centralized and decentralized systems in this phase.

3. **Reporting Phase.** In a centralized system, the app of a positively diagnosed user uploads the received (and/or sent) proximity identifiers (for, say, the past 14 days) to the central server (upon user’s consent and with the mandatory authorization by the health authority).

4. **Risk-level (or Risk-score) Computation Phase.** In some centralized systems, the server proactively identifies user(s) who may be at-risk of infection and notifies
them. This step may also be human-led as in the case of TraceTogether. Alternatively, the server computes the risk level (or risk-score) when a user-app sends an explicit request to compute the exposure risk.

Vaudenay proposes an interesting way of distinction between centralized and decentralized contact tracing systems in [39]. If the server generates and provides users with their ephemeral identifiers or if (possibly only infected) users communicate their own identifiers (or keys, pseudonyms, etc.) that allow the server to compute their identities, then the system is a centralized one. ROBERT [19, 20], TraceTogether [37], Arogya Setu [2–4], etc. are examples of centralized contact tracing systems.

2.2.1 A Naive Centralized Solution

A Hypothetical Insecure Protocol: If privacy were not a concern, it would be easy to build a smartphone-based centralized protocol. Whenever two phones come close enough (which can be detected through a simple Bluetooth program), both phones determine the other’s identity (say phone number). This information is pushed to the server on a regular basis. Whenever an individual (say A) is reported positive for SARS-CoV-2, the server can determine which other users were close to A. This does not provide any user privacy to other users as well as to the server, as personal identities (such as phone numbers) are disclosed to everybody involved in the process. Moreover, it also reveals the social interaction graph (much bigger than interaction with a positive patient) to the server.

Security Requirement: (User privacy) Any information related to the user’s identity should not be disclosed to the other without the consent of the owner.

This protocol allows server and users to get information beyond what is needed for the purpose of contact tracing. The following protocol realizes only a slightly higher level of sophistication than the last, and can provide privacy amongst users, which the previous method failed to achieve.

1. Each device would continually broadcast some random numbers and store these numbers locally.
2. Any device (say B), which is close to a device (say A), receives A’s random numbers and stores them.
3. On a regular basis, all devices will push their identity along with all the random numbers that they sent as well as received.
4. Whenever user A is reported positive, its identity details are sent to the server. The server can then easily identify all users (or devices), who were in closed contact of A.
Another simple alternative for the above protocol could be where the random numbers are generated (may be through a cryptographic primitive with a trapdoor) by the server. Every day, the central server push those numbers to users and the users broadcast those numbers through Bluetooth to all users who got close contact. Clearly, privacy between two users is achieved (as they only receive some random numbers). However, the server can easily track all individuals. In particular, the server still knows the social interaction of all users. That means, although there is privacy against users, there is no privacy against the server. This leads to a natural categorization of privacy of a user into two types:

1. privacy against users and
2. privacy against the central server or central administration.

However, one may allow the central server to identify all users who may have been in close contact with some SARS-CoV-2-positive user within a certain time window. Depending on situations, infrastructure, awareness of people of a country, one can consider some relaxed notion of privacy against the central administration. We call them centralized protocols. However, it is very important to know the entire extent of information that the server can gather even after behaving dishonestly. From cryptographic and privacy angles, it should not be relaxed to a level where surveillance of users is possible.

Restricting the data shared with the server: In the previous protocol, all the random numbers sent and received by each device are shared with the central server. This causes a significant loss of privacy as the central server can easily construct a social graph of the users at any point of time with the information shared by all the devices. On the other hand, if the server comes to know only the random numbers that are broadcast by all the devices (and not the ones that are received by the devices), it would not be able to deduce who came in contact with whom (and when) unless someone is tested positive, at which stage it would come to know of the random numbers received by the infected user’s device so that it may help in identifying or notifying the at-risk individuals.

2.2.2 Attack Scenarios

Examination of possible attack scenarios is important for analyzing the security of the specific systems and protocols in this chapter. A more elaborate version of these scenarios is presented in Chap. 4. Here, we divide the attacks under three broad categories, namely (1) Integrity Attacks (2) Privacy Attacks and (3) Non-cryptographic Attacks.
2.2 A General Framework of Centralized Protocol

2.2.2.1 Integrity Attacks

The integrity of a contact tracing system may be compromised through false positive alerts, i.e. when a user’s device triggers an alert in spite of the user not having been in the proximity of a user (in, say, the past 14 days) who is later diagnosed as positive. Called a *trolling attack* by Yaron Gvili [18], a simple way to generate false positives is one in which an adversary “borrows” the device of a user diagnosed positive for SARS-CoV-2 and takes it close to many other unsuspecting users. These users will likely get alerted for exposure risk in spite of no exposure to the actual infected patient. Although it may not be feasible to mitigate such an attack easily, such an attack may not be possible on the large scale, and may not pose a very serious threat; more efficient attacks that do pose serious threats, which might be carried out on a large scale are reviewed in the following discussion.

**Replay and Relay Attacks:** The proximity identifiers shared or broadcasted by a user who is later diagnosed positive may be misused by malicious entities by replaying them to other users who were not in contact with the infected user. If such identifiers are collected by an adversary (or a colluding group of adversaries) and later replayed to other devices, this is called a *replay attack*. Vaudenay [40] provides an interactive modification of the DP3T protocol that helps in preventing such attacks. A variation of this attack is the *relay attack*, where the identifiers collected from infected users are relayed immediately to other users in different locations, thereby proving the use of timestamps as possible safeguards against replay attacks useless. These attacks are very difficult to prevent. Inclusion of coarse location data as a preventive measure may work out partially—however it could turn out to be a computationally costly solution, and may also compromise users’ privacy to a certain extent. In [29], the author provides a novel solution for preventing replay and relay attacks on DP3T through a non-interactive modification.

**Inverse Sybil Attacks:** *Inverse Sybil attacks*, also called *crown attacks* in [34], are attacks carried out by a computationally strong adversary (e.g. a terrorist group), where many malicious users act as the same user. In (mostly decentralized) contact tracing systems where users send their broadcasted identifiers to the central server on positive diagnosis, multiple malicious users could broadcast the same identifiers to different people at different locations, causing all the receiving users to be falsely alerted of exposure if even one of them reports as a positive case to the server. In (mostly centralized) contact tracing systems where users report the received identifiers to the server on being infected, it is even easier to carry out this attack as it is sufficient for the adversaries to merely broadcast the same identifiers to a large population of users. Protocols that employ interactive exchange of proximity identifiers instead of allowing solely for broadcast are comparatively better protected from such attacks, while potentially incurring other costs (like loss of privacy, false negatives, etc.).
2.2.3 Privacy Attacks

Loss of privacy is another issue that needs to be considered when designing contact tracing systems, particularly when mitigating integrity risks. Inclusion of too many meta-data (e.g. a user’s time, coarse location, postal code etc.) may result in a partial loss of privacy or even complete identification. Preventing such loss of privacy is also important for encouraging a large-scale adoption through voluntary usage of the system by citizens without any fear of state surveillance.

**Linkage/Deanonymization Attacks:** A malicious adversary may monitor communication channels (and an honest entity may also receive some communication data), which may allow for identification (deanonymization) of the communicating users. Secure communication channels, and algorithms with strong privacy should be employed in order to avoid such *linkage attacks*.

**Single-entry Attacks:** The single-entry attack is a special type of the deanonymization attack, in which a malicious user keeps one’s device near the device of another user (without physically coming in proximity) and switching off the device otherwise. In case the malicious user’s app declares the user at-risk, it would lead to a conclusion that the targeted user is infected.

This attack may also result in the inadvertent deanonymization of a user if that is coincidentally the only user with whom a person has come into contact over a particular period of time, and has received an exposure alert.

2.2.4 Non-cryptographic Attacks

**False Negatives:** False negatives may usually occur due to discrepancies in the system such as failure to detect some devices, distorted communication between various entities (user-to-server, health authority-to-server, server-to-user, etc.), inaccurate computation of exposure score, etc. These issues are mostly non-cryptographic and unavoidable to some extent.

**Denial of Service (DoS):** Denial of service, especially by the central server (and possibly health authority) could occur because of factors like a high volume of communication, which may cause a drain on the storage capacity or even the power of the server, defective or incompatible settings in the device, bluetooth or other software, etc. Measures such as use of less costly algorithms may help in reducing these faults.

We now look into the details of some centralized systems that are already in use in some countries. We also analyze a few protocols that are actively being considered for implementation in some other countries.
2.3 BlueTrace, OpenTrace and TraceTogether

2.3.1 Framework

BlueTrace [8, 10] is an open source application layer protocol developed by the Singapore Government Digital Services for digital contact tracing systems. OpenTrace [5] is a reference implementation of the BlueTrace protocol under the GPL-3.0 license, which includes the generic code-base for the app (Android and iOS) as well as for the server. TraceTogether [37] is a specific implementation of OpenTrace that is being used by the Singapore Government for digital contact tracing and it supports the efforts of the Ministry of Health’s contact tracers. In this section, we primarily focus on the framework of BlueTrace protocol as it constitutes the basic layer on top of which the reference implementation and the apps are built.

2.3.2 Design Principles

A stated design principle of BlueTrace is that it is “designed around privacy” [8, 10]. The principle has been elaborated using the following salient points.

1. The BlueTrace peer-to-peer messages contain temporary identifiers that change frequently. So a third party (i.e., a party other than the app that sends the message and the central server) sniffing such messages, would be unable to identify the device or track the movement of the registered user.
2. The only personally-identifiable data point required by the protocol is the user’s phone number. The number is securely stored in the central server by the health authority.
3. The protocol stores the proximity information (or the encounter history) locally in each user’s device as collected through the peer-to-peer messages. This encounter history stored within a device is shared with the central server only when (a) the user is found to be infected and (b) the user chooses to share it.
4. A user can decide to revoke one’s consent of usage of the app at any point of time. As soon as the user withdraws consent, the server deletes all personally identifiable or link-able information about the user. Hence, encounter history from other installed apps can no longer be linked to this user.

The other core design principle of BlueTrace is that it is built around the supervision of the health authority. This is implied in the following ways.

1. The central server is expected to be administered by a trusted health authority as it is responsible for registering the user through personally identifiable information (phone number, etc.).
2. The temporary identifiers are generated and supplied to the apps by the central server.
3. The encounter history of the infected user is uploaded to the central server which then finds out the possible primary contacts by matching the encounter history with registered users’ database and helps the health authority to notify the corresponding users.

4. A federated system of central servers administered by multiple health authorities (in multiple countries/regions/jurisdictions) can communicate among each other to identify and notify possible primary contacts who may have traveled across countries/regions/jurisdictions.

It must be noted that although we have placed BlueTrace under centralized digital contact tracing systems, the documentation of BlueTrace considers its architecture to have both centralized and decentralized components where the decentralization aspect comes from the way peer-to-peer messages are passed and the encounter history is stored in the devices.

### 2.3.3 Protocol Details

The following sequence diagram (Fig. 2.1) can be used to describe the BlueTrace protocol.

There are three entities in this diagram—(a) the reporting server (b) the client (central) and (c) the client (peripheral). The **reporting server** is the same as what we have referred to as the central server so far. The **client (central)** and **client (peripheral)** can be viewed as two apps or two users communicating through peer-to-peer messages built on top of BLE (Bluetooth Low Energy), where one party plays the role of **central** by scanning for presence of other devices, while the other plays the role of **peripheral** by advertising its presence. When a **central** discovers a **peripheral**, it records the peripheral’s packet and sends its own packet to the peripheral. Each device plays the role of central vs peripheral in a 1 : 4 ratio of time-slices in a duty cycle. However, there could be some devices that always remain in the peripheral state. The communication that we have described just now falls under Device-to-Device Communication Protocol (DDC) part of BlueTrace, which we describe next.

#### 2.3.3.1 Device-to-Device Communication Protocol (DDC)

These messages are created using UTF-8 encoded JSON data interchange format and include the information about temporary ID, device model, signal strength (when operating as a central), Health Authority Identifier (HAI) and BlueTrace protocol version. These messages are sent in an unencrypted fashion. The signal strength is coded using Received Signal Strength Indication (RSSI) format. Since temporary IDs are not link-able to the users of the devices and these IDs are constantly rotated, there is seemingly no immediate security or privacy risk of this disclosure even in the presence of malicious apps or software listening to such packets.
Table 2.1  A sample packet

```json
{
  "id": "FmFISm9nq3PgpLdxxYpTx5tF3ML3Va3wqgY9DGIXt1uPbw+1ZtqAdqbvR1nSvr+ILXPG==",  // TempID
  "md": "iPhone X",  // Device model
  "rc": -60,  // Signal strength
  "o": "IJ_HAI",  // Health authority identifier
  "v": 2  // Protocol version
}
```
A sample packet is shown in Table 2.1. Once the app receives such a packet from an encounter, it stores the processed information in the local database of the device for 21 days post which, it is removed. The app also blacklists the other device for two consecutive duty cycles to avoid immediately getting engaged with the same device.

2.3.3.2 Device-to-Reporting-Server Communication Protocol (DRSC)

The DRSC protocol can be divided under three different functions:

1. initial registration of the user,
2. supply of temporary IDs to the registered app from time to time in the form of forward-dated batches, and
3. upload of encounter history once a user is found to be infected and chooses to share the proximity history.

DRSC is built on top of Firebase, which is “a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014” [9].

**Initial registration of the user:** At the time of initial registration, the app uses Firebase phone authentication service to bind the phone number with the device. The Firebase cloud service stores the mapping of a static and unique user ID to that phone number. Subsequent calls from the app are authenticated using this ID.

**Generation of temporary IDs:** The temporary IDs are requested by the app from the server using the function getTempIDs(). The temporary IDs are generated in a batch of 100 and each temporary ID is expected to be used for 15 min implying a batch of 100 such IDs may last for 25 h. The format of a temporary ID is shown in Fig. 2.2.

The server encrypts a part of this ID (Fig. 2.2) using a symmetric encryption key using AES-256-GCM standard and then converts it into a Base64 encoded string. The secret key of the symmetric encryption is known only to the server and hence that part of the ID can neither be deciphered nor tampered by any other party. The IV part of the ID is generated using a PRNG (Pseudo-Random-Number-Generator) algorithm and the Auth Tag is used for integrity check.

| UserID (12 bytes) | Start time (4 bytes) | Expiry time (4 bytes) | IV (16 bytes) | Auth tag (16 bytes) |
|-------------------|----------------------|----------------------|---------------|--------------------|

Encrypted with AES-256-GCM

84 byte Base64 encoded string

**Fig. 2.2** Format of temporary ID
Upload of encounter history: The upload of the encounter history starts with the invocation of function getHandshakePin() to obtain an authenticated PIN for uploading the data to the cloud server. Next, getUploadToken() function is called to get a single-use token for the upload. A JSON file is created with the upload token and the encounter history of the last 21 days by using the writeToInternalStorageAndUpload() function. Finally, the file is uploaded to the Google cloud using uploadToCloudStorage() function. There is no separate encryption applied to this file at the time of upload and the stored file’s security is based on the settings of Google Cloud Storage encryption.

2.3.3.3 Contact Tracing at the Server End

Once the encounter history is uploaded by an infected user’s app, the server analyzes each of the packets to check if the HAI (Health Authority Identification) within the packet belongs to its own jurisdiction or some other Health Authority.

If the HAI corresponds to its own jurisdiction, it inspects the packets and decrypts the temporary ID using the symmetric secret key to detect if the encounter timestamps are within the valid duration (based on the start time and expiry time marked inside the packet) of the messages. It ignores a packet if it is not within the valid duration—this prevents the possible replay attacks on the protocol. For valid packets, it retrieves the relevant information like the signal strengths and correlates multiple packets to decide the duration of the encounter with any specific user. Epidemiological parameters (like the duration of contact and the proximity based on signal strength) are used to decide if the user from whose device a set of messages were received by the infected user, can be considered a close contact or not. If the encounter is indeed considered a close contact (close encounter), the phone number is retrieved based on the user ID contained within the packets. The contact tracing team of the health authority would then reach out to the relevant user associated with the phone number and continue the contact tracing process as followed in the manual version of contact tracing. The OpenTrace implementation has not automated this step of notification and has let it remain a human-centric and human-led process.

2.3.3.4 Server-to-Server Communication Protocol

If the HAI belongs to a different health authority, the server (belonging to health authority A in Fig. 2.3) passes the encrypted temporary IDs to that server under health authority B, so that the server B may decrypt the same by referring to its data-store B to identify the relevant user ID and returns a PseudoID (salted cryptographic hash) of the user ID back to server A. Server A uses such PseudoIDs to identify the possible close encounters and notifies the PseudoIDs that can be considered close contacts back to health authority server B.
2.3.4 Highlights and Characteristics

We now look at some of the highlights and characteristics of the proposal.

- **“Human-in-the-loop” system**: BlueTrace has been designed as a protocol to extend the manual contact tracing process of a Government or health authority by decentralized automated recall of contacts through a digital contact tracing system while maintaining the contact tracing of probable close contacts of infected individuals centralized under the supervision of the Government’s or health authority’s contact tracing team.

- **Minimal collection of personally identifiable data**: Apart from phone numbers, no other personally identifiable data of the users get stored/tracked in the system.

- **A user’s identity is stored only in the data-store of the health authority to which the user belongs**: Neither the device to device communication nor the server to server communication reveals the static user ID or phone number of a user.

- **Encounter history tracks who has been in contact with but not where**: The protocol does not require any access to the location of a user’s device at any point of time. So, GPS or any other location information (like WiFi scanning) is neither checked nor stored by the protocol.

- **Data upload happens in presence of a contact tracer (for TraceTogether)**: The authentication mechanism (of BlueTrace protocol) and the process in place for TraceTogether app, together ensure that the encounter history upload step can happen only in presence of a genuine contact tracer.

- **Configurability and extensibility**: There are many parameters of the protocol that are configurable. For example, the number of days for which the encounter history is stored in a device can be modified as per epidemiologic need. In the current OpenTrace implementation and for TraceTogether it is kept as 21 days.
Similarly, the Bluetooth signal strength (RSSI readings) is used as a proxy for the distance between two devices. In TraceTogether, the actual implementation has been done after doing a careful calibration of various mobile devices from different manufacturers and models as used in Singapore. This calibration data is provided as a baseline dataset and any implementation of BlueTrace protocol can potentially refine it as per the need in a different country / region. Similarly, the final step of alerting users about possible contact event(s) can also be automated by implementing additional notification APIs between a device and the server. Implementations of BlueTrace under separate health authorities can also modify the ratio of scanning and advertising periods within a duty cycle. Similarly, the algorithm for generating Temp IDs and PseudoIDs can be tweaked as per need of the health authority.

- **BlueTrace uses BLE in Connected Topology mode:** Bluetooth enabled devices can communicate through BLE in two modes [36]—broadcast topology mode and connected topology mode. In the broadcast topology mode, a broadcasting device can simultaneously broadcast messages to multiple observers. However, the BlueTrace designers opted for the connected topology mode in which a device while playing the role of a central needs to establish connection with another peripheral device before exchanging BLE packets between each other.

### 2.3.5 System Analysis

This section provides analyses of the protocols from different aspects.

#### 2.3.5.1 Security and Privacy Analysis

The security and privacy of BlueTrace protocol hinges upon two key design elements:

1. the centralized symmetric key encryption (and decryption) at the server-end of temporary identifiers and
2. the decentralized logging of such IDs during the encounter of two devices.

The encounter history of a user remains unknown and hence private from the health authority unless the user is tested SARS-CoV-2 positive. The identity of a user (whether infected or not) remains unknown and private to any other user and any other authority. The location information is unknown to any entity and hence the protocol cannot be used on its own to track any user. The personally identifiable information (static unique user ID and phone number of the user) are stored in the central data store. The security of any such data depends upon the security scheme adopted between the central server and central data store and the symmetric key encryption used in the DRSC protocol. There is no separate encryption scheme used in BLE packets; the local data storage does not include any additional security or privacy considerations other than what is guaranteed by Android or iOS apps in their respective operating environments.
Cryptographic Primitives and Encodings Used

- **AES-256-GCM.** AES-256-GCM is the 256-bit keyed initialization of the Authenticated Encryption Standard that uses the Galois Counter Mode. This mode of operation is capable of authenticated encryption of a message as well as verifying the integrity and authenticity of additional authenticated data, and is specified in the NIST Special Publication 800-38D [22].

- **Base64 encoding.** In order to avoid corruption of data during transmission due to reasons such as interpreting binary data as text, control characters, etc., it is encoded before sending over any channel. One such method of encoding is the base64 encoding, which is a binary-to-text encoding scheme that translates the input into a radix-64 representation (every 6 bits of the input message is encoded into a 64-bit code).

- **IV generation.** The term IV stands for initialization vector (also starting vector). As its name suggests, it is a fixed-length input provided at the initiation of cryptographic implementation. The main purpose of this input is to provide randomization to the implemented scheme or construction, and is generated through a pseudorandom permutation so as to satisfy the requirement of uniqueness—no IV must repeat under the same key.

- **Authentication tag.** An authentication tag or a Message Authentication Code (MAC) is a cryptographic scheme of message authentication. An authentication tag is used to prevent forgery of messages; it maintains the authenticity of the message (i.e. the message was truly sent by the sender as claimed) as well as its integrity (i.e. the message received is unchanged and the same as the message sent), when sent over a communication channel.

- **Salted cryptographic hash.** A salt is a random value input to a cryptographic scheme, usually to passwords before hashing, implementing one-way functions, etc. It is similar to an IV, but holds an important distinction with it in that it may be repeated, as it is generated by a pseudorandom function.

Protection Against Security Attacks

- **Backend impersonation.** Since the backend server for any implementation is expected to have a well-known domain name with proper certification, the scope of a backend impersonation attack is limited. However, if the communication channel between the app and the server is secure and authenticated, the system would be more robust against such attacks.

- **Server-end data breach.** The protocol can be subjected to such an attack since the server is the nerve center in the framework of this protocol and stores personally identifiable data.

- **Replay attack.** The chance of such an attack is limited to within 15 min at maximum as the temporary IDs expire after that time.

- **Relay attack.** BlueTrace is susceptible to relay attack.
• **Deanonymization.** This is not possible from device end alone without additional contextual data (like side channel attack). However, if the server data is compromised, the user data may easily get deanonymized.

• **Coercion threats.** User-end devices do not store any personally identifiable information, and hence the extent of damage through coercion threats would be limited. However, if the device of an infected user is in the possession of a malicious third party, there is a significant chance of it being misused to create a large number of false positives in the system.

• **Other attacks.** BlueTrace cannot be subjected to the inverse-Sybil attack as multiple devices cannot impersonate the same user. One-entry attack is also possible in BlueTrace, OpenTrace and TraceTogether; the extent of damage through such an attack on these protocols would be limited since the alerting mechanism is a human contact tracer-led process.

It is important to keep in mind the possible open areas. Based on this analysis, it may be inferred that server-end data breach is probably the most significant risk in BlueTrace. Apart from that, backend impersonation, deanonymization and relay attacks may also need to be handled by putting some counter-measures in place.

### 2.3.5.2 Architecture Analysis

**Use of the BLE connected topology mode:** While this topology provides the advantage of symmetric communication of encounter messages between two devices in proximity, the downside is that it is not as “easy or fast to use” [36] as compared to the broadcast topology. The additional advantage of encrypting the packets in connected topology mode has not been utilized in BlueTrace.

**Upload of encounter data by the infected user’s app:** The advantage of this scheme is the optimization of data upload since only data that need to be shared with the health authority is uploaded and all other encounter history remains private; the disadvantage is that the infected user’s device can become the single-point-of failure. If the device crashes or the local data becomes corrupted/inaccessible for some reason, the entire contact history would be lost.

**Centralized security of sensitive and personally identifiable data:** This ensures that if the centralized storage mechanism is made robust and secure (at the equivalent level of any sensitive Government data), the possibility of it getting compromised would be remote. At the same time, such a centralized store is subject to single-point-failure and dependent upon the safety and security mechanisms implemented by private entities like Google or Apple. As pointed out by Robert Valk in his article [38], for OpenTrace implementation, Google Cloud Services “stores your data, manages the encryption keys, links the mobile number and unique ID of every user, executes the server-less functions, provides the cryptographic libraries as part of the cloud function’s runtime, and controls the Android app distribution channels.”
Challenge faced by iOS users: This is an important issue to be considered as a possible open area. Apple restricts the usage of the connected topology mode of BLE by not allowing it to run in the background in iOS. Due to this, iPhone users need to run TraceTogether in the foreground. This creates inconvenience. TraceTogether team has mentioned in an article [17], “To help users keep the app running in the foreground while minimizing battery usage, the TraceTogether team included a power saver mode setting in the code-base. If you are an iPhone user, all you have to do is keep TraceTogether open but place the phone upside down in your pocket or face down on the table. That will trigger the power saver mode, allowing the app to regularly scan the environment for other TraceTogether users.”

2.4 COVIDSafe

2.4.1 Framework

COVIDSafe [13] is the official digital contact tracing app released by the Australian Government on 26 April, 2020 and is based on the BlueTrace protocol designed and developed by the Singapore Government Digital Services. The code-base for both the Android and the iOS systems have been released as open source repositories in github [6, 7].

In this section, we primarily describe the app-specific customizations and changes adopted at the implementation stage that are not a standard part of the BlueTrace protocol. For all other aspects (especially for highlights, characteristics and system analysis) we refer to the sections on BlueTrace, OpenTrace and TraceTogether.

2.4.2 Design Principles

The stated design principle of BlueTrace of “designed around privacy” [8, 10] continues to be emphasized in the case of COVIDSafe. The privacy policy has been well-documented [30] and an extensive Privacy Impact Assessment Report [14] as well as the Department of Health’s response to it have been made public. All the design principles of BlueTrace apply to the implementation of COVIDSafe except the following-

1. There are quite a few personally-identifiable data points captured by the app at the time of initial registration. These are-
   - Mobile phone number. This is used later for contacting the individuals who are identified by the system as “at-risk of being infected”.
2.4 COVIDSafe

- **Name.** This is mentioned later to ensure that the right individual is contacted by the health officials from the State or the Territory. If uncomfortable sharing one’s real name, a user is free to mention any “pseudonym or fake name”.
- **Age range.** This data point is used by health officials to decide on the prioritization of who should be contacted with urgency.
- **Postcode.** This data point allows the health department to assign the appropriate health official from the designated state or territory corresponding to that post code to contact the individuals and decide on which area to be declared as hotspot etc.

2. If a user decides to delete the app and discontinue its usage, the website provides the following information [13]-

“You can delete the COVIDSafe app from your phone at any time. This will delete all COVIDSafe app information from your phone. The information in the secure information storage system will not be deleted immediately. It will be destroyed at the end of the pandemic. If you would like your information deleted from the storage system sooner, you can complete our request data deletion form.”

One may have a look at [14, 41] for privacy and legislative issues in this regard.

2.4.3 **Protocol Implementation Details**

There are not many differences in the implementation of the protocol as compared to what has already been described under OpenTrace and TraceTogether, except the following-

- The backend of COVIDSafe runs on the Amazon Web Services (AWS) platform (not on Google Cloud) and the server data is stored in AWS Sydney region data center so that the Australian Government may have the legal jurisdiction over the data center.
- A user wanting to change one’s registration information may do so by deleting the app and reinstalling it with the desired changes. A user may also modify information such as one’s name, age-range and postcode with uninstall-reinstallation process, but not the phone number. A user may also contact the Department of Health and submit a relevant form to delete all of one’s data from the system.
- The encrypted user ID is the basis on which the “de-identified reports about uptake of COVIDSafe” and “analytical data from iTunes and Google Play about COVIDSafe including the number of downloads, average use time and deletions” would be collected by Digital Transformation Agency of Government of Australia.
- Certain bugs [32] that originally got detected and then fixed in OpenTrace, reappeared in the COVIDSafe implementation; these were eventually rectified in a subsequent release (13 May, 2020).
- Currently, the implementation does not allow any user to register using a phone number other than that from Australia (with country code of +61).
2.4.4  **Highlights, Characteristics and System Analysis**

The highlights, characteristics and system analyses for COVIDSafe being almost identical to those in the corresponding subsections of BlueTrace, shall not be mentioned separately. However, there are certain differences for the following functionalities:

- **Personally identifiable data.** Apart from phone numbers, a few other personally identifiable information that is collected by the app includes the name, postcode and age-range. However, it is allowed to share a pseudonym or a fake name in place of the actual name of the user.

- **Data upload process for a user tested positive.** Once a user chooses to upload the contact details captured by one’s app over the last 21 days, a health official initiates the authentication process. As part of the authentication mechanism, the health official sends a PIN number (through the system) to the user’s registered phone number (over SMS) so that the person may use that to upload the relevant contact information from the device to the backend server.

2.5  **Centralized Systems with Private Specifications**

The following systems and protocols have user-level documentation and in some cases client-side source code in the public domain; detailed technical specifications have not been explained in details in certain cases. We have already given an outline of the Aarogya Setu app in the previous chapter. We add a few more details in this section.

**Aarogya Setu:** At the time of initial registration, Aarogya Setu asks for the user’s mobile number, name, gender, profession, age and the list of countries visited in the last 30 days. In addition, it also requests the user to mention if he/she is willing to volunteer in the times of need. The information collected from the user at the time of registration is securely stored in the central server. The server also generates a unique pseudorandom Device iDentity (DiD) number for each user and associates the same with the encrypted personally identifiable set of information at the backend database. For all subsequent communications between the devices coming in proximity with each other or between the device and the server, this DiD is used. At the time of recording the contact events, a user app also records the (time, duration, location and distance) corresponding to each nearby device’s app (in addition to the unique randomized identifier shared by that device).

All the local device data gets encrypted using Advanced Encryption Standards (AES) before storing. Any device data that is older than 14 days gets automatically deleted. Similarly, the longest period for which the user data remains undisturbed at the server end is 60 days.
Whenever the user takes a self-assessment, that data along with the location of the user gets transferred and securely stored in the backend server. Whenever a person is tested positive, that data from the Indian Council of Medical Research (ICMR) is shared with the Aarogya Setu server. If the infected user has the Aarogya Setu app installed, then a notification is sent to the user-app for changing the color to “Red”, the proximity data captured by that person’s device would be uploaded to the server so that the contact tracing process can be started by identifying the individuals who might be at-risk based on their exposures. Individual user-apps also calculate the risk status by communicating with the server to discover if the owner was in proximity with any known infected user or not. It has been mentioned in the FAQ document that Aarogya Setu collects a user’s location for two purposes, namely-

1. to identify the possible hot-spots where infections could be spreading and
2. to understand the routes traversed by the infected individuals so that appropriate measures like sanitization and isolation of affected individuals may be implemented easily.

PEPP-PT NTK: The Pan-European Privacy-Preserving Proximity Tracing Need-To-Know System [1] has been designed jointly by a number of European institutes and organizations, including the Fraunhofer Institute for Telecommunications, Robert Koch Institute, Technical University of Berlin, TU Dresden, University of Erfurt and Vodafone Germany.

In this protocol, the temporary IDs that are shared by a device as proximity IDs have a mapping with the user’s persistent ID. This mapping is maintained at the server end. When a user is diagnosed positive, all the received temporary IDs from his/her device get uploaded to the server. The server decrypts the temporary IDs to identify the persistent IDs of the users who have significant risks and notifies them through a push or pull mechanism. The risk computation depends on the number of contacts of a user with the infected users and the durations and distances (as decoded from the signal strengths of the received signals) between the devices in each such contact event.

As per current information [44], it appears that only Georgia has implemented this protocol in their Stop Covid contact tracing system.

NHS_COVID-19: The NHS_COVID-19 app had received significant attention before the UK Government decided to abandon the initiative [24] in favor of the Exposure Notification APIs jointly developed by Apple and Google. The main reason behind this decision [42] appears to be the experience from the pilot roll-out, in which the app was able to detect only 4% of the iPhone devices and 75% of the Android based mobile phones, whereas Google/Apple system could catch 99% of the devices.

Released with the free-ware licensing scheme in the United Kingdom on 6 May, 2020, this app was developed by NHSX with the help of VMWare and a team of scientists and doctors, and launched as a pilot test release in the Isle of Wight. The client-side code-base for both the Android and the iOS systems were released as open source repositories in github [26, 27].
We outline some characteristics of this protocol, particularly as it may be reasonable to expect that the new app being developed using Google/Apple APIs may retain some of these features.

- An interesting difference of this app from other apps was that it allowed a user to notify NHS in case he/she felt unwell (to some extent this is similar to the self-assessment option of Aarogya Setu), which in turn could trigger alerts from NHS to other users of the app who might have come in close contact of this user in the past 28 days.
- The contact risk model developed by NHS was being constantly updated to enhance its accuracy.
- No personally identifiable data was collected by the app. However, the anonymized data sets were planned to “be used for NHS care, management, evaluation and research”. The types of data collected by the app included (a) first half of the postal code of the user’s area and (b) the make and model of the mobile device.
- Users were allowed to delete the app whenever they wanted and the NHS was committed to follow the data regulations for the usage of the anonymized data.
- The app did not collect any location information.
- Certain questions [31] were raised about the possible security and privacy issues of the collected data and usability challenges (like the requirement of keeping the app in the foreground).

### 2.6 ROBERT and DESIRE: Centralized to Decentralized

Considering the overall framework, ROBERT and DESIRE both have the essence of decentralized as well as centralized systems. The ROBust and privacy-presERving proximity Tracing (ROBERT) [19, 20] scheme is a digital contact tracing protocol specification that has jointly been designed by the PRIVATICS project team from Inria, France and Fraunhofer, AISEC, Germany under the aegis of the Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT/PEPP) initiative [28]. DESIRE [12] has been proposed as an evolution of the ROBERT protocol by the PRIVATICS project team from Inria, France. Both ROBERT and DESIRE have been designed with the intent of bringing the best of centralized and decentralized systems. Hence, it can be placed right in the middle of these two approaches. Precisely for that reason, we position the description of this pair of protocols at the end of the chapter on centralized schemes before we venture into the decentralized digital contact tracing systems.

In each of the following subsections we first describe the ROBERT protocol and then present the changes that have been incorporated in the DESIRE proposal.
2.6.1 Design Principles

The security and privacy requirements and the design goals for ROBERT and DESIRE are stated in the respective specification documents [12, 20]. However, the design principles are not separately articulated. In our view, there are three prominent design principles that can be interpreted from the specifications, which stand out as the guiding factors behind the design decisions. These are as follows.

1. **Hybrid approach.** The first design principle is that centralized as well as decentralized features must be judiciously combined in a digital contact tracing system instead of creating either a fully centralized framework, which may compromise on privacy requirements or a fully decentralized framework, which may compromise on security and robustness requirements. This is evident from the statement in the abstract of the ROBERT specification that mentions, “Although it might seem attractive in terms of privacy to adopt a fully decentralized solution, such approaches face inherent challenges in terms of security and robustness against malicious users” [20].

2. **Adaptable design.** The second design principle that can be interpreted from the specifications is that the protocols propose to construct the framework in the form of configurable components in such a way that the entire system can be adapted towards a fully centralized or a fully decentralized system. We shall elaborate more on this principle when we describe the details of the DESIRE protocol in this section.

3. **Generalized adversarial model:** ROBERT assumes the threat of malicious users and an honest-but-curious authority (server) against whose attacks the protocol needs to be robust whereas DESIRE considers the adversarial model to be a fully generalized one in presence of malicious users, malicious authorities and a collusion of both the parties.

Some of the design goals of ROBERT are subjective (or loosely defined) in nature while the rest are specific and verifiable. The subjective goals are:

- The system must be simple to understand and use.
- It must be transparent.
- It should be possible to deploy the system with minimal infrastructure.

The specific goals that can be verified in any implementation are:

- **Maintenance of anonymity.** Neither the client app side nor the server side of the protocol may collect or store any personal (or personally identifiable) data.
- **Federated infrastructure.** The system is expected to scale beyond one country or region and the server side architecture and primitives are designed assuming that the multiple trusted authorities can manage the different servers that would communicate and cooperate with each other.

These design goals are not separately listed down in the DESIRE specification. We may however assume that all these design goals are also valid for DESIRE, consid-
ering that it is an extension of the ROBERT protocol (except the federated infra-
structure, since the protocol, rather surprisingly, has not retained any such provision).

Apart from the above design goals, the security and privacy requirements are as follows:

- The proximity data must be accurate and reliable. This is a subjective requirement
  and is not quantified in the specification so as to derive the extent to which false
  positives or false negatives can be tolerated.
- No user or central authority (in general) should be able to identify in real-time or
  discover through post-processing, the identity of another user (whether infected
  or not). Similarly, no such party should be able to detect the location of another
  user or reconstruct a social graph around another person.

### 2.6.2 Protocol Details

Both ROBERT and DESIRE are designed to use BLE in broadcast topology mode
and not in the connected topology mode. DESIRE is more explicit in its description
of packet structure and the mechanism of using the broadcast topology, but the
specification for ROBERT does not elaborate the BLE. Hence, at the lowest level
of the protocol stack, they are expected to behave differently from BlueTrace. The
pros and cons of this choice has not been discussed in the specification. The efficacy
of this choice can be analyzed only when a reference implementation is tested or an
actual implementation of ROBERT (or DESIRE) is rolled out for any country. We
first focus on ROBERT in depth and then describe the ways in which DESIRE differs
from ROBERT by explaining the new primitives, processes and functionalities.

A device running an app based on ROBERT broadcasts HELLO messages through
BLE, which other nearby devices can capture and store. Similarly, it simultaneously
captures HELLO messages broadcasted by other devices.

#### 2.6.2.1 Structure and Usage of the HELLO Message

The HELLO message is 128-bit long and consists of four parts:

- **ECC**: Encrypted Country Code (8-bits)—described in detail below.
- **EBID**: Ephemeral Bluetooth IDentifier (64-bits)—described in detail below.
- **Time**: 16-bit truncated (less-significant-bits) part of the current time-stamp incor-
  porated to prevent replay attacks.
- **MAC**: Message Authentication Code (40 bits) and is intended to prevent integrity
  attacks.

The most important part is the Ephemeral Bluetooth IDentifier (EBID). Within the
context of digital contact tracing Systems, the term ‘ephemeral’ has been used for
the first time in case of DP-3T protocol. As the term suggests, these IDs are meant to
be short-lived. It serves the purpose of being a pseudorandom number corresponding to an instance of the app and for a specific time-interval. EBIDs change at regular intervals. This number is designed to not reveal any personally identifiable user information. Its value constantly changes in a random fashion—hence, no one can possibly track a user from a series of these numbers emitted by the app installed in the user’s device. However, these numbers would eventually play a crucial role in the identification of proximity events from which the ROBERT protocol-based system can deduce whether a user might have been infected or not.

Let us now understand how an EBID is constructed and used. EBID is derived as

$$\text{EBID} = \text{ENC}(K_{\text{Server}}, i | ID_A).$$

Here, ENC stands for an encryption algorithm and as per the protocol specification it is expected to be implemented as a 64-bit block cipher like SKINNY-64/192 [9].

The parameters $K_{\text{Server}}, i$ and $ID_A$ are used as a key to the algorithm. $ID_A$ is a permanent identifier attached to user A and is known only to the server to which the app registers initially at the time of installation. The integer $i$ stands for an epoch number (achieved by discretization of time-intervals) and EBID value remains unchanged for an epoch typically of a duration of 15 min as per the recommendations of the Bluetooth specifications [11].

$K_{\text{Server}}$ is an $L$-bit long server key ($L \geq 128$) known only to the server. From this description, it is evident that only the server can create this identifier and the app on its own can neither create it nor decrypt any of its parameters including the permanent identifier $ID_A$. $\text{ENC}^{-1}$ is the inverse of ENC, which means it is the decryption algorithm using which the server can retrieve back $ID_A$ and the epoch number $i$ from EBID using the server key $K_{\text{Server}}$.

Encrypted Country Code (ECC) is derived from a publicly known 8-bit Country Code (CC) of the user’s country by encrypting it using a federation key $K_G$, known only to the trusted set of federated servers from the participating European countries and also the EBID value allocated to the user for the corresponding time-interval or epoch ($i$).

Once an app receives a HELLO message from a nearby device, it first checks if the time value is within a tolerable range of the current system time-stamp (to be precise the truncated version of the system time-stamp) and if the time value is permissible, it stores the entire HELLO message along with the current system time-stamp in its $\text{LocalProximityList}$, which is maintained for a certain number of configurable days (e.g. 14 days) and deleted permanently thereafter.

After a regular number of intervals (in terms of epochs), the server, upon explicit request from the app, sends a list of $T$ pairs of (EBID, ECC) where each pair is designated for a specific epoch ($i$). For all practical purposes, the EBID and ECC values are nothing more than pair of random numbers to the client-side of the app. Whenever the app receives a HELLO message, it can demarcate the positions of the bits corresponding to ECC, EBID, Time and MAC and also verify whether it has been tampered with or not by checking the MAC; however it can not decipher the
contents of EBID or ECC. This makes the peer-to-peer messages tamper-proof and secured even in the unsecured broadcast topology mode of BLE (Bluetooth Low Energy) protocol.

The usage of EBIDs in case of a user who is tested positive would be as follows.

2.6.2.2 Exposure Status Reporting

Once a user (say $A$) is diagnosed as SARS-CoV-2 positive, she may choose to declare the EBIDs her app has captured during the last CT days where CT represents a time-period for which she could have been contagious for others who have come in her close proximity. She would require to get the relevant authorization (e.g. a pre-approved token) from a health authority to upload the (HELLO, Time) pairs from the app’s $LocalProximityList$. The exact process of upload authorization has not been elaborated in the specification and we may assume that it could vary from one country to another.

It is interesting to note that during the upload process, $A$ does not reveal any of her EBIDs. Moreover, the specification has recommended that the upload of the entire list of (HELLO, Time) pairs should not be done in a batch mode. In the batch mode of upload, there could be a possibility of a curious authority constructing the social graphs of contact events at the server end. This may eventually lead to re-identification of user identities with the help of circumstantial meta-data for the events. Hence, the specification has suggested that the upload step may use (a) some type of Mixnet or proxy or (b) some trusted intermediate server (like a hospital’s server) that would mix the $LocalProximityLists$ of multiple users diagnosed as positive and later upload the data to the backend server or (c) have a secured hardware component at the server end that can be accessed only via a set of secured APIs, which can mask any information about the originating apps/devices that had uploaded the lists. We have mentioned earlier that the server can retrieve back the permanent identifier ($ID_{User}$) of a user and the corresponding epoch number ($i$) from any one of its EBIDs using the decrypting function $ENC^{-1}(\ldots)$. Hence, after receiving the (HELLO, Time) pairs in the upload process from an app, the server resolves the permanent identifiers for the users that got exposed to $A$ and the corresponding epoch numbers when those users came in close contact with $A$. The server adds these epoch numbers in a $List of Exposed Epochs (LEE)$ maintained in a table (called $IDTable$) indexed by the permanent identifiers along with some more information about each user’s app. Since the server does not store any personally identifiable information (like phone number) against a permanent identifier, it can’t (and is not expected to proactively) inform about the exposure event to any user’s app. The user’s app has to make any explicit request about the exposure status to the server in order to know about its current risk of being infected by the virus.

Next, we describe the process of exposure status detection.
2.6.2.3 Exposure Status Detection

The user-app needs to query the server to know the user’s exposure status. In this query (called ESR_REQUEST), the app sends its current set of values comprising EBID, epoch number \(i\) and time-stamp along with a message authentication code to avoid the possibility of an adversary tampering with the request packet. The server resolves the permanent identifier of the user \(\text{ID}_{\text{User}}\) from the EBID and finds out if the LEE list corresponding to \(\text{ID}_{\text{User}}\) is empty or not.

If the list is empty the server responds back with a binary flag bit of 0 (indicating that the user is not at risk). On the other hand, if the list is non-empty, then it uses a publicly known algorithm (as per the guidance of epidemiologists and the health authority) that uses the stored data (primarily from LEE) to identify if the risk probability is above a threshold and if it is above such a threshold, declares the user at risk by returning a bit 1 for the binary flag. At the same time, it marks the UN (User Notified) flag against the permanent identifier as true (1). Irrespective of the status (at risk or not at risk), the server updates the SRE (Status Request Epoch) field with the epoch number contained in the ESR_REQUEST as long as the request is a valid one.

An ESR_REQUEST is considered valid if:

- the UN flag was not already set to 1 earlier,
- the request has been received at least after a desired (configurable) minimum number of epochs after the previous request,
- has a correct epoch number,
- has a time-stamp within the tolerable range of current network time, and
- has the correct message authentication code, signifying that there was no tampering with the message.

Once the app receives the at-risk flag in ESR_REPLY from the server set to 1, it stops sending any further ESR_REQUEST to the server, notifies an appropriate message at the application level to the user (so that he/she can take the next steps like getting in touch with a health agency for testing etc.) and keeps sending HELLO messages to the nearby devices.

2.6.2.4 Initial Registration

When a user (say A) decides to use an app based on the protocol, she downloads the app from an approved store like Google app store (if her device has Android operating system) or from Apple store (if her device is iOS based) and installs it in her device. At the time of user registration of her app, no personal information of A (like phone number) or device information (like IMEI number or location) is shared with the backend server and the server ensures that she is assigned a permanent identifier \(\text{ID}_{\text{A}}\) in the IDTable maintained at the server-end which remains known only to the server and the user is aware of her unique user-id required for signing in to the app.
Although it has not been mentioned explicitly anywhere in the specification, there seems to be an one-on-one mapping maintained at server end between the user-id (presumably chosen by $A$) and ID$_A$.

At the time of registration, the server also keeps a record of additional information for $A$ including (and not limited to) an authentication key ($K$-Auth$_A$) that would be used to authenticate the messages coming from $A$’s app, an encryption key ($K$-Enc$_A$) that would be used to encrypt sensitive information to be shared with $A$’s app, UN-flag (User Notification) for $A$ (initially set to false), SRE (Status Request Epoch) to store the epoch when $A$’s app would have sent the most recent ESR_REQUEST and LEE (the List of Exposed Epochs) for $A$ which is initially empty. The server also sends an initial list of (EBID, ECC) code pairs that $A$’s app would use for a certain number of epochs before it requests for an additional list of such pairs from the backend server.

### 2.6.2.5 Risk Scoring Approach

The specification defers the actual description of the risk scoring algorithm to the implementation stage. It mentions that the methodology shall eventually be guided by recommendations from health authorities and epidemiologists. Thus, it may evolve along with the changing nature of the virus or its transmission mechanism at different stages of the pandemic. The designers have argued that the centralization of risk scoring procedure makes the protocol more flexible as compared to decentralized systems since the algorithm can be changed in one place to seamlessly alter the behavior of the rest of the parts including the client side apps. As the backend is expected to be implemented as a secured and a reliable centralized server (under regular auditing for security and privacy by independent regulatory bodies) with a well-known domain name and certification, the entire system is resilient to attacks that may try to tamper the risk scoring mechanism because unlike decentralized systems, the risk evaluation is not done at individual device level in ROBERT (and DESIRE). To influence the risk score of a user’s app, either the malicious user needs to expose himself/herself to an infected person or devise ingenious mechanisms to break robust well-known cryptographically secured communication between the app and the server.

At this point, the specification talks about a binary value to be returned to the app in ESR_REPLY message signifying a status of at-risk (binary value of 1) or not at risk (binary value of 0). The designers have also kept a provision open for returning a probability value instead of a binary value or returning a value of 1 even for a small percentage of users who are not at risk, at random. The latter approach can be used to prevent a one entry attack, in which an adversary may plan to place just one entry (of the target victim) in its LocalProximityList and hence when its app is declared at-risk by the server, the adversary would still not able to identify with certainty whether the target victim is diagnosed positive or not. However, this approach would increase the false positives to a limited extent like 5–10%.
2.6.2.6 Federated System of Servers

The protocol has been designed to be used across countries in Europe where each country can implement its own version of the app following the ROBERT (or DESIRE) framework and each country can have its own backend server. As long as the HELLO messages follow the structure where the first 8 bits contain the Encrypted Country Code (ECC), the federated systems can inter-operate in the following manner:

The specification illustrates a situation where two users, one from Germany (named Uta) and another user from France (called Bernard) happen to come in close proximity of each other and their apps interchange the HELLO messages on a particular epoch \((i)\). If at a later point of time, one of them (say Uta) is diagnosed positive, his app may follow a process (as recommended by the German authorities) to upload the HELLO messages (including the one received from Bernard) upon Uta’s consent to the German server. Once the German server decodes (with the help of the Federation Key, \(K_G\)) the first 8-bits of the HELLO message sent by Bernard to Uta and realizes that it corresponds to a user’s app registered under France, it forwards the corresponding HELLO message to the France’s server. Eventually, the server in France adds a record in the LEE corresponding to Bernard’s permanent identifier (\(ID_{\text{Bernard}}\)) about the exposure epoch \((i)\), which can be used during the risk score computation for Bernard.

2.6.3 Protocol Details: Differences in DESIRE

We now describe the specific differences in DESIRE protocol vis-a-vis the ROBERT protocol.

2.6.3.1 Private Encounter Tokens (PETs)

DESIRE proposes the storage of Private Encounter Tokens (PETs) by the devices instead of storing the received EBIDs. The PETs are generated in the following manner:

Suppose \(A\) and \(B\) happen to come in close proximity to each other. Both \(A\)’s app and \(B\)’s app independently generate pseudorandom numbers that remain constant for the duration of an epoch. For example, if \(A\)’s app has generated a pseudorandom number \(x\) (which is only known to \(A\) and hence a secret key), it shares the computed value of \(g^x\) (public key) as the EBID to the nearby devices including \(B\)’s device. Let us assume on the other hand \(B\)’s device generates a pseudorandom number \(y\) and shares the computed value of \(g^y\) as the EBID to \(A\).

Next, each of the apps records a computed hash value of the EBID received to the power of its own secret as the PET token from the given encounter. Hence, for both the devices this event gets recorded as the same value \(H(g^{x\cdot y})\) where \(H\) is a
cryptographic hash function such as SHA-256. The DESIRE specification also refers to the optional storage of additional metadata like speed, signal strength etc.

From the cryptographic point of view, it has been assumed that both the apps follow the discrete logarithm of elliptic curve (Curve25519) with the same group structure of order $p$ and generator $g$. In fact, this is the basic idea of Diffie-Hellman key exchange protocol. While this protocol is secure in classical computational model under some usual assumptions, this is absolutely insecure in quantum computational paradigm. Fortunately, commercial quantum computers are still elusive and there are several protocols which are secure in the quantum model too.

At a later point of time, if $B$ happens to be diagnosed SARS-CoV-2 positive, then upon his consent and health authority’s requisite approval, all the PET tokens stored in $B$’s device for the last CT days get uploaded to the backend server which adds these to a global $EList$, containing the exposed list of PETs. Sometime later, $A$ enquires the server to know her risk status and at the time of sending the query also uploads the PETs received from other devices (including those that were sent by $B$’s device) in the last 24h. The server finds a match of the list of exposed PETs as uploaded by $B$ with some PETs uploaded by $A$. The server then maintains a record of this encounter by adding the epoch numbers and other details corresponding to the encounter in a List of Exposed PET Metadata (LEPM) against the permanent ID of $A$. The server also computes the risk score based on the configured risk scoring algorithm depending upon how many of $A$’s LEPM entries appear in the global $EList$ and returns the flag at-risk (if the risk threshold has been crossed) or (otherwise) not-at-risk to $A$’s app.

The actual implementation of PET tokens, as proposed in the specification of DESIRE, is slightly more complex than what has been described above. It suggests that a device maintains two lists simultaneously where one is called the ETL (Exposure Token List) and the other is called the RTL (Request Token List). For every encounter and computed value of $g^x \cdot y$, it stores two PET values $H("1\" \vert g^x \cdot y)$ and $H("2\" \vert g^x \cdot y)$ in such a way that the PET value that gets stored in $A$’s device in ETL is the same as the PET value that gets stored in $B$’s device in RTL and vice-versa. $B$, once diagnosed positive, uploads the PET values from its ETL while $A$ uses the PET values contained in her device’s RTL while sending the ESR_REQUEST message to the backend server. This has been done so that a curious server would not be able to build a social graph for any infected user as the uploaded PET tokens of $B$ that his app uploads after him being diagnosed positive would be different from the PET tokens he would have sent earlier in his app’s ESR_REQUESTs.

2.6.3.2 Initial Registration

At the time of initial registration of the app, an Authorization Token (AT) is generated at the server end and sent to the user’s phone number via SMS. This AT is used by the app to register at the backend. The phone number is not stored by the server.

The server also generates an encryption key (EK) specific to the user’s app, encrypts all the entries in IDTable corresponding to the user’s permanent ID (except the ID itself) with EK, sends ID and EK to the app and finally deletes the EK.
2.6.3.3 Message Structure

DESIRE does not use any message structure (like a HELLO message), instead it proposes the direct sending of EBIDs in the broadcast payload followed by the payload in the scan/response message. Even though no time value is shared, the chance of a replay attack is minimized by converting the encounter event from an asymmetrical to a symmetrical one.

2.6.3.4 Risk Scoring

In DESIRE, the Exposure Risk Score (ERS) for every app is maintained in IDTable along with the LEPM (List of Exposed PET Metadata), and the at-risk or not-at-risk status is determined based on whether this score has crossed a threshold value (decided based on guidance of epidemiologists and health authorities).

2.6.4 Highlight and Characteristics

**No personal or personally identifiable information is collected:** ROBERT does not collect any personal information or personally identifiable information at the time of app installation and registration. By suggesting the usage of Mixnet or proxy or intermediate trusted server (like that of a health authority) or a dedicated hardware device at the server end, it proposes to take care of masking the network identify (like IP address) of the devices at the time of data upload corresponding to the users who are diagnosed positive. In case of DESIRE, although the phone number is collected at the time of registration for ensuring that the user has the device corresponding to the phone number, it is later deleted and not stored at the server-end.

**Server and app jointly own every activity:** The EBIDs (Ephemeral Bluetooth IDs) that are sent, and received by the devices as part of their HELLO messages are created at the backend server for ROBERT. For DESIRE, the EBIDs are directly generated and broadcast from the device-end. The proximity details (e.g., the EBIDs) of nearby devices are stored in each device; the identification of close-contact events, their corresponding epochs and risk scoring are all performed at the server-end. Once again, the retrieval of such information (like whether a user is at-risk or not at-risk) is done by the app. Hence, we can say that the backend server and the app jointly own every activity in ROBERT and DESIRE protocols.

**Cryptographic algorithms are abundantly used:** Every communication, whether between two nearby devices or between a device and the server or between two federated servers uses encryption and authentication. Both symmetric as well as asymmetric keys have been used during encryption. In ROBERT all key-exchanges are interactive in nature while in DESIRE both interactive and non-interactive key exchanges have been used.
**Broadcast topology mode of BLE is used:** Both ROBERT and DESIRE make use of the broadcast topology mode of the BLE protocol. Since app-level encryption and authentication are used, the transparent nature of packets in broadcast topology does not pose any security or privacy threat. On the other hand, the broadcast topology possibly ensures faster exchange of messages between devices coming in proximity as compared to the connection topology mode. However, it has been proposed that the broadcast topology be used in a different way for DESIRE as compared to that for ROBERT. In case of ROBERT, the exchange of packets could be asymmetric which means if device $A$ captures the BLE packets (of a ROBERT protocol based app) from device $B$ in any interaction event, it is not necessary to assume that the device $B$ also should captures BLE packets from $A$ as well (even if that happens in majority of the cases). However, in DESIRE it is mandatory to have a symmetric exchange of packets between the two devices.

**Configurability and extensibility:** Although it has not been described in the specification in detail, it can be inferred that both ROBERT and DESIRE have several configurable elements (e.g., duration of an epoch, contagious period, tolerable time difference of a received message from current network time, risk scoring algorithm, user notification flag reset procedure etc.) that can be modified at the time of initial installation of the system or at a later point of time.

**DESIRE is more decentralized than ROBERT:** In DESIRE, there is no longer a dependency on the server to generate the Ephemeral Bluetooth IDs. The mobile devices generate such pseudorandom IDs independently for sending to each other and store those in such a fashion (using a non-interactive key exchange protocol) that those can be transparently matched at the server end to identify the possible close-contact events without any possibility of recovering the original pseudorandom EBIDs that got exchanged between the original devices, which came in close proximity to each other.

### 2.6.5 System Analysis

We divide the analysis of ROBERT and DESIRE in two parts—first we focus on the security and privacy analysis of the frameworks, and next we concentrate on the architecture analysis of the systems. We point out the possible open areas within each of these parts.
2.6.5.1 Security and Privacy Analysis

Let us first look at the system parameters, message encodings and cryptographic primitives used in ROBERT in each of the following subsections and then highlight the changes in DESIRE.

System Parameters
The system parameters are as follows:

- \( T_{\text{ptsstart}} \): Start time of the proximity tracing service in the country to which the backend server belongs. It is expressed as the NTP (Network Time Protocol) \( \text{seconds} \) value \[23\]. The server also maintains an epoch number that starts with the value 0 for the first \( \text{epoch\_duration\_sec} \) duration (usually 15 min translated to number of seconds) starting from \( T_{\text{ptsstart}} \). DESIRE does not mention this parameter.
- \( \text{epoch\_duration\_sec} \): This is a configurable parameter that represents the duration of an epoch in seconds for the entire system.
- \( \Delta \) (\( \delta \)): Time tolerance for acceptability of a HELLO packet. It can typically be a few seconds. This is not applicable for DESIRE as Time is not sent as part of the message packet.
- \( \text{ContStart}_A \) and \( \text{ContEnd}_A \): These are derived values (in seconds) for a user A’s start time (\( \text{ContStart}_A \)) and end time (\( \text{ContEnd}_A \)) of being contagious. The derivation algorithm would use a configurable parameter (CT) for the number of days (like 14 days or 21 days) as per the guidance from epidemiologists or the health authority.
- \( M \): The number of epochs between two consecutive requests from app end to the server for getting the next set of (EBID, ECC) pairs. This is not applicable for DESIRE.
- \( T \): The minimum number of epochs that must elapse between the consecutive ESR_REQUEST messages from the app to the server. This should be applicable for DESIRE as well. However, the specification has not commented on this.

Message encodings
The codes used in different messages are as follows:

- \( \text{CCS} \): Publicly known 8-bit country code values for the countries that agree to participate in the federated system around the protocol (ROBERT or DESIRE). In the HELLO messages this code is used in an encrypted form (described later) and is called the Encrypted Country Code (ECC) which is also of 8-bit length. This code is not applicable for DESIRE as it is silent about the usage of Federated system of servers.
- \( \text{UNA} \): A flag (called User Notified) that can have either a \text{true} or \text{false} value. When the user for app A is notified to be at-risk, this flag is set to \text{true}.
- \( \text{LEEA} \): This a list maintained for app\( A \) to contain the list of epochs in which the user corresponding to the app\( A \) got exposed to someone who later was diagnosed to be positive. In case of DESIRE the two lists that are important at server end are—the global ELList (Exposure List) and the user specific LEPM\( A \) (List of Exposed PET Metadata).
- Request type: There are four types of requests and their corresponding codes. HELLO message has code 1, ESR_REQUEST has code 2, Unregister has code 3 and DeleteHistory has code 4. Among these, Unregister and DeleteHistory have not been described in the specification. For DESIRE, ESR_REQUEST is applicable, however no other message type/code has been described.

Cryptographic items
The different keys and items are as follows:

- $K_S$: $L$-bit long (where $L \geq 128$) server key (initialized during server set-up) that is used by the server to generate the EBIDs. This primitive is not applicable for DESIRE.

- $K_G$: $L$-bit long (where $L \geq 128$) federation key shared among the backend servers of the countries participating in the federated agreement. This primitive is not applicable for DESIRE.

- $(sk_S, pk_S)$: An asymmetric key-pair (called the registration key-pair) where $sk_S$ is the secret key known only to the server and $pk_S$ is the public key distributed to all the apps. This registration key-pair is generated at the server set-up time over the elliptic curve NIST-P256 [15], where $pk_S = sk_S \cdot G$, where $G$ signifies the base point on the prime order of the indicated elliptic curve. This key-pair is also a necessary ingredient to form a SharedSecret between the server and a particular instance of the app, which we describe next. This primitive is not applicable for DESIRE.

- SharedSecret: User’s app ($A$) generates an ephemeral asymmetric key-pair $(ske_A, pke_A = ske_A \cdot G)$ and then transmits $pke_A$ to the server. app $A$ computes $ske_A \cdot pk_S$, while the server generates $sk_S \cdot pke_A$. Clearly, both are the same and this primitive forms the SharedSecret between the app $A$ and the server. This primitive is not applicable for DESIRE.

- $K - Auth_A$: This is the authentication key (of length $\geq 128$ bits) that is used to authenticate app $A$ messages and both the server and the app side generates it from the SharedSecret by $K - Auth_A = HMAC_{SHA256}(\text{SharedSecret, } \text{authentication key})$. This primitive is not applicable for DESIRE. However, an encryption key called $EK$ is generated and shared to app $A$ by the server.

- $K - Enc_A$: This is the encryption key (of length $\geq 128$ bits) that is used to encrypt sensitive messages sent by the server to app $A$ and both the server and the app side generates it from the SharedSecret by $K - Enc_A = HMAC_{SHA256}(\text{Shared Secret, encryption key})$. This primitive is not applicable for DESIRE.

- $ERS_A$: Exposure Risk Score (ERS) would be maintained at server end for each registered app ($A$) in DESIRE. This is not mentioned in ROBERT specification.

- $ID_A$: This is a 40-bit unique identifier generated by the server (for both ROBERT and DESIRE based systems) using a random drawing process (without replacement) to uniquely identify app $A$ and is not shared with app $A$.

- $HELLO_A$: We now dissect the encoding structure of a HELLO message emitted by the app $A$ at an epoch $i$. The $HELLO_A$ message consists of an information
part ($M_A$ of length 88 bits) and a Message Authentication Code part ($MAC_A$ of length 40 bits). This primitive is not applicable for DESIRE.

Now we have the following process:

$$HELLO_A = [M_A | MAC_A]$$

$$MAC_A = \text{HMAC\_SHA256}(K - Auth_A, 00001111 | M_A)$$

$$M_A = [ECC_A, EBID_A, Time]$$

$$ECC_A = \text{MSB}(\text{AES}(K_G, EBID_A | 0^{64})) \oplus CC_A$$

$$EBID_A = \text{ENC}(K_S, i | ID_A),$$

where ENC is a 64-bit block cipher

Here $Time$ is less significant 16 bits of the current system time (NTP Seconds) and ENC can be implemented as SKINNY-64/192 [14].

Security and privacy issues
The security and privacy of the framework can be analyzed by referring to the above primitives and by considering the following adversarial models of attacks.

- **Backend impersonation:** Since the backend server would have a well-known domain name with proper certification and the communication channel between the app and the server is secured, backend impersonation attack would not be feasible by a malicious third party.

- **Server data breach:** In case of ROBERT, server data breach may have a risk of disclosing certain sensitive information like LEE list in the IDTable (although the user details like ID is a pseudonym). However, in case of DESIRE the risk is even lesser as all IDTable entries are encrypted by the $EK_A$ keys that are known only to the corresponding apps.

- **Replay attack:** As the HELLO message contains a Time field and the entire message is protected against any tampering by using a MAC, a replay attack cannot be mounted against ROBERT. For DESIRE based systems, the symmetric exchange of PET tokens would prevent any replay attack.

- **Relay attack:** The specifications of ROBERT and DESIRE do not mention any measure against relay attacks. This could be feasible, however it may be feasible only within a limited time-span.

- **Deanonymization of users:** No personal or personally identifiable information is collected either by the app or by the server. Hence, even if the data at any end gets compromised or leaked, it would not lead to deanonymization of users. However, there could be other means (like side-channel attacks through contextual information beyond what is captured in ROBERT or DESIRE framework) through which deanonymization can still happen. One-entry attack can also lead to deanonymization. However, the extent of this attack would be limited as the system does not allow an app to send any subsequent ESR\_REQUEST message once its UN flag is set to ‘true’ at the server end unless it is later reset through a certain approval process (not outlined in the specifications).
• **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 in ROBERT but not in DESIRE (due to symmetric nature of the contact events being recorded through PET tokens).

• **Coercion threats:** For ROBERT, the data stored in a device based on the exchange of HELLO messages are not decipherable without the help of the server, hence the chance of coercion threats in these systems is limited. However, there could still be a chance of an infected person’s device being 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 as those users may get notified to be at-risk by the server once the data gets uploaded. The latter issue is applicable to DESIRE as well.

### 2.6.5.2 Possible Open Areas

We have already discussed that the protocols do not currently guard against certain attacks like relay attacks, inverse-Sybil attacks etc. as described in the previous subsection. That is, there are certain open areas that might need to be studied in a disciplined manner in future.

• The ROBERT specification does not mention the exact algorithm for generating $(ID_A)$.  
• The ROBERT specification does not mention whether the server key $(K_S)$ is generated as a pseudo-random number.  
• The ROBERT specification does not mention how the federation key $(K_G)$ is generated and shared among the federated servers.  
• The DESIRE specification does not mention how the Encryption Key $(E K_A)$ for each app is generated by the server.  
• (In ROBERT) No information shared on the choice of NIST-P256 for asymmetric registration key-pair generation on the server side, on the choice of HMAC_SHA256 for generation of $K – Auth_A$ and $K – Enc_A$ on both the server side and app side, on the choice of AES in ECC, on the choice of SKINNY-64/192 in EBID and the choice of HMAC_SHA256 in MAC generation.  
• (In DESIRE) No information has been shared on the choice of Curve25519. Also, it has not been mentioned why SHA-256 has been used for the cryptographic hash function while generating the two types of PET tokens.

### 2.6.5.3 Architecture Analysis

**Bluetooth Low Energy layer used in broadcast topology:** This ensures that the communication is “easy or fast to use” [36]. Since the message size is longer than what can fit in a single broadcast packet, subsequent packets must be requested by the receiver from the broadcasting device (using Scan/Response or Fragmentation approach).
Secured communication at app level: The app uses security and authentication measures on every communication to preserve privacy and security of information.

Precaution against identification of users by the server while uploading the proximity data: The server is designed to not know who uploaded what part of the proximity data. In case of DESIRE the device end EBIDs are completely unlinkable to the server as it can only observe the PET tokens.

Simple interaction among federated servers: The federated system of servers are expected to function independently and the data exchange is kept minimal to pass information corresponding to only those users who might have traveled from one country to another, came in close contact with another user from the visited country and later at least one of them is diagnosed as positive.

2.6.5.4 Possible Open Areas

- Two message types—Unregister and DeleteHistory have not been described in ROBERT specification.
- It is not clear how DESIRE can be extended to multiple authorities.
- It is not mentioned how UNA can be reset in the ROBERT framework once it is set (to true). This has been described in greater detail in the DESIRE specifications.
- (In ROBERT) If a user’s app crashes and the user does a fresh registration after installing the app again, would the previous permanent ID be linkable to the present permanent ID? If not, would the previous history of LEE be lost from the user’s perspective?
- (In ROBERT) Once UN is set (to true) for a particular user (say A), the specification mentions that her app would keep on sending HELLO messages to nearby devices. However, nothing is mentioned on whether her app would continue to receive and store HELLO messages from other devices. That would be important from A’s perspective as there could be a chance of her being diagnosed as negative while some other user whose device came in close proximity to her (after A’s UN flag being set as ‘true’) is later tested positive.
- The calibration of signal strengths for different manufacturers’ devices in the broadcast topology would be critical for the success of such systems.
- The specifications talk about other metadata like signal strength or duration being used in the risk scoring algorithm, but it has not elaborated on how that would be captured.
- In Fig. 1 of the ROBERT specification [20], C is assumed to infect A as C is diagnosed positive before A gets diagnosed. However, that need not be true. A might have infected C and the symptoms shown by C might have appeared before it happened for A, which may have led C to be tested before A.
- In ROBERT’s specification [20] as well as in DESIRE [12], the terms at-risk and at risk of exposure have been used interchangeably. However, at-risk may typically mean at risk of being infected, which is different from at risk of exposure.
In Sect. 2.1 of ROBERT’s specification [20], it is mentioned that the risk score is a binary number (at-risk or not at risk). However, that call should only be taken at the medical personnel level. For example, the same duration and proximity of exposure of a person with another infected individual may be considered to have different levels of risk based on the age of the user. For a young asymptomatic individual, a health advisor may not suggest immediate testing to be done—however, if the person is aged (although asymptomatic so far), the guidance could be different. So only communicating a binary value would be too limiting for a medical professional to take a call on what needs to be done. This issue may arise in case of DESIRE protocol as well.

It appears that once an application is labeled at-risk—it remains in that state forever. However, in reality that would not be the case. If A and C came in close contact 14 days back and C is tested positive today, A can be falsely considered at-risk the present time (day) and not anymore from tomorrow. So if C’s application notifies the server of the fact that C is tested positive and A’s application is marked at-risk today, if A’s application happens to not be running at the present day and it wakes up tomorrow and sends an exposure status request to the server tomorrow—the server should not declare A to be at-risk. How is this checked in the system? There can also be cumulative effects of exposure of an individual repeatedly over a number of days—it may not be fair to record that as one instance of at-risk.

2.7 Conclusion

In this chapter we have discussed some of the contact tracing protocols in details, mostly from the aspects how a centralized server participates in such designs. That is, we have considered the centralized systems in this chapter. The basic framework and protocol details are discussed. We describe the BlueTrace, OpenTrace and TraceTogether protocols from different aspects. Then we consider the COVIDSafe proposal. Finally we note that the boundary between the centralized and decentralized protocols might not be very stringent in certain cases. That is why we have discussed ROBERT and DESIRE in great details that show how the designs move towards decentralized domain via the hybrid route. Based on this understanding, in the next section, we will discuss the decentralized protocols in details.

References

1. A high level overview of PEPP-PT_NTK. [https://nadim.computer/res/pdf/PEPP-PT_NTK_High_Level_Overview.pdf](https://nadim.computer/res/pdf/PEPP-PT_NTK_High_Level_Overview.pdf).
2. Aarogya Setu App. [https://www.mygov.in/aarogya-setu-app/](https://www.mygov.in/aarogya-setu-app/).
3. Aarogya Setu App Forum. [https://www.mygov.in/task/aarogya-setu-app-covid-19-tracker-launched-alert-you-and-keep-you-safe-download-now/](https://www.mygov.in/task/aarogya-setu-app-covid-19-tracker-launched-alert-you-and-keep-you-safe-download-now/). Aarogya Setu App.
4. Aarogya setu wikipedia page. [https://en.wikipedia.org/wiki/Aarogya_Setu](https://en.wikipedia.org/wiki/Aarogya_Setu).
5. Angius, E., & Witte, R. (2012). OpenTrace: An open source Workbench for automatic software traceability link recovery. In 19th Working Conference on Reverse Engineering, Kingston, ON, 2012, pp. 507–508. https://doi.org/10.1109/WCRE.2012.63. https://github.com/opentrace-community.

6. AU-COVIDSafe/mobile-android Github repository. https://github.com/AU-COVIDSafe/mobile-android.

7. AU-COVIDSafe/mobile-ios Github repository. https://github.com/AU-COVIDSafe/mobile-ios.

8. Bay, J., Kek, J., Tan, A., Hau, C. S., Yongquan, L., Tan, J., & Quy, T. A. BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders. https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf.

9. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., & Sim, S. M. (2016). The SKINNY family of block ciphers and its low-latency variant MANTIS. Cryptology ePrint Archive: Report 2016/660. https://eprint.iacr.org/2016/660.pdf.

10. BlueTrace protocol: Privacy-preserving cross-border contact tracing. https://bluetrace.io/.

11. Bluetooth Core Specification Version 5.1 Feature Overview. https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-v5-1-feature-overview/.

12. Castelluccia, C., Bielova, N., Boutet, A., Cunche, M., Lauradoux, C., Le Métayer, D., & Roca, V. DESIRE: A third way for a European exposure notification system leveraging the best of centralized and decentralized systems. https://hal.inria.fr/hal-02570382/document.

13. COVIDSafe App Website by Australian Government Department of Health. https://www.health.gov.au/resources/apps-and-tools/covidsafe-app.

14. COVIDSafe application privacy impact assessment on the COVIDSafe website by Australian Government Department of Health. https://www.health.gov.au/resources/publications/covidsafe-application-privacy-impact-assessment.

15. Digital Signature Standard (DSS). FIPS PUB 186-4. Federal Information Processing Standards Publication. Information Technology Laboratory, National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

16. Firebase wikipedia page. https://en.wikipedia.org/wiki/Firebase.

17. GovTech Singapore Website. 6 things about OpenTrace, the open-source code published by the TraceTogether team. https://www.tech.gov.sg/media/technews/six-things-about-opentrace.

18. Gvili, Y. (2020). Security analysis of the COVID-19 contact tracing specifications by Apple Inc. and Google Inc. Cryptology ePrint Archive, Report 2020/428, 2020. https://eprint.iacr.org/2020/428.

19. Inria, France and Fraunhofer AISEC, Germany. ROBERT: ROBust and privacy-presERving proximity Tracing – Summary. https://github.com/ROBERT-proximity-tracing/documents/blob/master/ROBERT-summary-EN.pdf.

20. Inria, France and Fraunhofer AISEC, Germany. ROBERT: ROBust and privacy-presERving proximity Tracing – Specification. https://github.com/ROBERT-proximity-tracing/documents/blob/master/ROBERT-specification-EN-v1_1.pdf.

21. Inria, France and Fraunhofer AISEC, Germany. ROBERT: ROBust and privacy-presERving proximity Tracing – Documents. https://github.com/ROBERT-proximity-tracing/documents.

22. NIST Special Publication 800-38D. (2007). Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology, November, 2007.

23. Network Time Protocol Wikipedia Page. https://en.wikipedia.org/wiki/Network_Time_Protocol.

24. Next phase of NHS coronavirus (COVID-19) app announced. https://www.gov.uk/government/news/next-phase-of-nhs-coronavirus-covid-19-app-announced.

25. NHS_COVID-19 App website by the government of UK. https://www.nhsx.nhs.uk/covid-19-response/nhs-covid-19-app/.

26. nhsx/COVID-19-app-Android-BETA Github repository. https://github.com/nhsx/COVID-19-app-Android-BETA.
27. NHSX/COVID-19-app-iOS-BETA Github repository. https://github.com/nhsx/COVID-19-app-iOS-BETA.

28. Pan-European privacy-preserving proximity tracing. https://www.pepp-pt.org/.

29. Pietrzak, K. Delayed authentication replay and relay attacks on DP-3T. Cryptology ePrint Archive, Report 2020/418. https://eprint.iacr.org/2020/418.

30. Privacy policy for COVIDSafe app language translations on the COVIDSafe website by Australian Government Department of Health. https://www.health.gov.au/resources/translated/privacy-policy-for-covidsafe-app-other-languages.

31. Concerns on the NHS_COVID-19 wikipedia page. https://en.wikipedia.org/wiki/NHS_COVID-19#Concerns.

32. Independent analysis on the COVIDSafe wikipedia page. https://en.wikipedia.org/wiki/COVIDSafe#Independent_analysis.

33. OpenTrace on the BlueTrace wikipedia page. https://en.wikipedia.org/wiki/BlueTrace#OpenTrace.

34. The crypto group at IST Austria. Inverse-Sybil attacks in automated contact tracing. https://eprint.iacr.org/2020/670.

35. The COVIDSafe application privacy impact assessment agency response. COVIDSafe document by Australian Government Department of Health. https://www.health.gov.au/sites/default/files/documents/2020/04/covidsafe-application-privacy-impact-assessment-agency-response.pdf.

36. Townsend, K., Cufí, C., Akiba, & Davidson, R. Getting started with Bluetooth Low Energy. https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch01.html.

37. TraceTogether wikipedia page. https://en.wikipedia.org/wiki/TraceTogether.

38. Valk, R. Contact tracing apps - can we trust the tech? https://platform.deloitte.com.au/articles/contact-tracing-apps-can-we-trust-the-tech.

39. Vaudenay, S. Centralized or decentralized? The contact tracing dilemma. https://eprint.iacr.org/2020/531.

40. Vaudenay, S. Analysis of DP-3T between Scylla and Charybdis. https://eprint.iacr.org/2020/399.pdf.

41. Website of the Federal Register of Legislation, Australian Government. Biosecurity (Human biosecurity emergency) (Human coronavirus with pandemic potential) (Emergency requirements–Public health contact information) Determination 2020. https://www.legislation.gov.au/Details/F2020L00480.

42. Why the NHS Covid-19 contact tracing app failed. https://www.wired.co.uk/article/nhs-tracing-app-scraped-apple-google-uk.

43. Wikipedia page on NHS_COVID-19. https://en.wikipedia.org/wiki/NHS_COVID-19.

44. Wikipedia page on COVID-19 Apps. https://en.wikipedia.org/wiki/COVID-19_apps.