PriMera Scientific Engineering
Volume 8 Issue 6 June 2026
DOI: 10.56831/PSEN-08-275
ISSN: 2834-2550

PriMera Scientific Publications

Data Encoding for Distributed and Federated Contact Tracing

Type: Review Article
Received: May 16, 2026
Published: May 31, 2026

Citation:
Hari TS Narayanan. "Data Encoding for Distributed and Federated Contact Tracing". PriMera Scientific Engineering 8.6 (2026): 64-79.

Hari TS Narayanan*

Founder, Senior Consultant NetProwise Consulting, Gokulam Colony, Chennai 600033, India

***Corresponding Author:** Hari TS Narayanan, Founder, Senior Consultant NetProwise Consulting, Gokulam Colony, Chennai 600033, India.

Abstract

In digital contact tracing, a user's device exchanges proximity data with nearby devices, and this data includes identifiers associated with the user. When a user tests positive, the proximity data stored on their device is analysed to identify others who may have been exposed. Once these individuals are identified, manual contact tracing is initiated for further follow-up. Multiple contact-tracing solutions exist today: some are designed for closed user groups, some for national populations, and others are open systems without clearly defined boundaries. However, these isolated solutions are insufficient for a world with high levels of global mobility. Ideally, a contact-tracing system should be global, interoperable, and capable of supporting seamless mobility across regions. Designing such a system presents several challenges, including scalability. In this study, we propose a proximity-data encoding approach that enables scalable centralized, distributed, and federated contact-tracing architectures. Our federated and distributed models support the coexistence of diverse identifiers and proprietary encryption methods. The federated approach enables user mobility while allowing each country to maintain control over its citizens' data.

Keywords: Contact Tracing; Encoding; TraceTogether; Exposure Notification; Proximity Identifier; Federated; Distributed; Bluetooth Low Energy; Global mobility

Introduction

A digital contact-tracing system, like any other digital system, enhances manual contact tracing by providing greater agility and scalability. It collects, stores, and processes data to identify cases and contacts associated with one or more infectious diseases. A subject is considered a *case* when they are infected and exhibit disease-specific symptoms or clinical signs. A subject is considered a *contact* when they are likely infected but have not yet developed symptoms or signs associated with the infection. A case or contact may arise when a subject has been in the presence of an infected individual. Such presence can take the form of physical contact, close contact, or proximity contact. Physical contact involves the exchange of bodily fluids. Close and proximity contacts are defined in guidelines issued by the U.S. Centers for *Disease Control and Prevention* (CDC) [1].

User identification is a critical component of both digital and manual contact-tracing systems. Existing designs can be assessed by examining the characteristics of their proximity-identification mechanisms [2]. In general, the user identifier in a contact-tracing system must remain anonymous to protect privacy. Several anonymization approaches have been documented in survey literature [3-5]. While most countries mandate the use of anonymized identifiers, at least one country employs a non-anonymous identifier scheme [6]. A globally deployed contact-tracing framework must therefore satisfy privacy requirements, scale effectively, and support the coexistence and interoperability of diverse identifier formats. The following sections evaluate how current digital contact-tracing solutions perform relative to these expectations.

Two primary anonymization methods are found in existing contact-tracing implementations [7-12]. In the first approach [7], a user's identifier (such as a phone number) is mapped to a unique random string on a secure server. This random string serves as the user's anonymous identifier for all proximity-related operations. To reduce the risk of wireless tracking, the identifier is rotated periodically—a feature common to all anonymization methods reviewed here. A secure server manages both anonymization and de-anonymization processes. When necessary, the server deanonymizes the identifier and correlates the user's proximity records to determine whether the user qualifies as a contact. Because this method requires storing and retrieving multiple anonymized identifiers for every user, it depends on random-access storage operations, which can create bottlenecks when scaled to large populations [2], particularly in centralized architectures.

In the second method [8-10], the anonymous proximity identifier is generated directly on the user's device (typically a smartphone). In this approach, the relationship between the user's actual identifier and the anonymous identifier is implicit. De-anonymization can occur only on the user's device unless all participating devices are engaged in every de-anonymization operation. Although this method avoids random-access storage requirements, the de-anonymization process is resource-intensive at the device level and introduces significant overhead [2]. Moreover, the results produced can be incomplete or unreliable due to events such as network loss or device unavailability [13]. These issues are particularly common when smartphones serve as the primary devices. Appendix A illustrates some of these challenges using COVID-19 data from the United States [14, 15].

Proximity data can be exchanged over wireless technologies in two ways:

1. Broadcasting discovery advertisement messages, or
2. Establishing one-to-one connections with discovered peers.

Solutions that rely on peer-discovery broadcasts scale more effectively than those requiring explicit connections between every device pair. Some wireless technologies, such as Bluetooth, support random non-resolvable source addresses (distinct from the user's identifier). This feature is highly desirable, as it helps protect users from wireless tracking.

Bluetooth on smartphones is widely used for proximity-data exchange. It supports peer discovery and *non-resolvable address* features, and it benefits from a significantly larger installed base than cellular technologies. Proximity exchange can continue even in the absence of cellular connectivity. A Bluetooth device can share proximity data in two ways: by establishing a direct connection with another device or by broadcasting data to all nearby peers.

The connection-based approach requires forming a link between every pair of devices, involving at least six message exchanges per connection. This method does not scale effectively under rapidly changing load conditions. In contrast, the broadcast-based approach periodically transmits proximity data in advertisement messages that are scanned by nearby devices. This method is non-intrusive and scales more efficiently, though it is constrained by smaller payload sizes and the inherent unreliability of broadcast delivery. When used, it requires a control mechanism capable of ensuring an acceptable level of reliability under dynamic load conditions. A scheduling algorithm for Bluetooth-type technologies, which coordinates scanning and broadcasting to maintain required reliability under varying load, is proposed in [16]. This schedule adapts with agility, adjusting its parameters based on the most recent load conditions observed by the device. There is a centralized solution [11] that employs BLE connections to exchange proximity data once an encounter exceeds a specified duration threshold (e.g., 5 minutes). This strategy reduces the number of required connections and

lowers post-processing workloads.

Modern smartphones support multiple versions of Bluetooth, and these versions are generally backward compatible with a substantial installed base [17]. A key difference relevant to proximity-data exchange is the maximum advertisement-broadcast payload size. Bluetooth Low Energy (BLE) supports an advertisement payload of 31 bytes, which restricts proximity-data formats to 31 bytes or less. Bluetooth 5.0 and later versions increase this limit to 244 bytes. Current proximity-data designs either fit within the 31-byte BLE constraint [6, 8-10] or exceed it [7]. Designs that remain within the 31-byte limit can operate across all Bluetooth versions. Designs requiring larger payloads can use advertisement broadcasts only on Bluetooth 5.0 and later and doing so requires significant architectural modifications. In practice, a contact-tracing system is expected to operate across coexisting Bluetooth versions. Additional challenges apply to all digital contact-tracing systems. Distance estimation based on Bluetooth signal-strength attenuation is inherently limited in accuracy [18]. At the time of this study, no alternative wireless technology offers comparable ubiquity or suitability.

None of the current solution is designed for global operation or interworking. Ideally, a contact-tracing system should be global, providing seamless coverage to support mobility. Designing such a system presents significant challenges. It requires cooperation among countries and mechanisms for exchanging data without compromising national control over citizens' proximity information. The system must be flexible enough to accommodate proprietary features, including country-specific privacy requirements and diverse identifier encodings. Membership-management procedures must remain simple to support epidemic and pandemic response. All components of the system, except the inter-country data-exchange layer, should remain under the control of the respective countries.

The encoding proposed in this paper is motivated by the absence of a scalable, interoperable global solution. The encoding fits within the 31-byte BLE payload, enabling scalable and non-intrusive proximity-data exchange. It can evolve to support more robust designs as larger Bluetooth payloads become available. The encoding provides two levels of security to support data exchange both within a member system and across federated systems. Each member system may operate in either a centralized or distributed configuration, depending on capacity requirements. The encoding avoids bottlenecks observed in earlier designs [7] while supporting diverse identifier formats and proprietary encryption schemes. Data loss due to device loss is minimized, and reliable results can be produced even when only partial data is available [13]. Membership management of federated system incurs minimal overhead.

The remainder of this paper describes the proposed proximity-data encoding and its suitability for a global contact-tracing architecture. To our knowledge, this is the first encoding designed for such a purpose. Section 2 presents a centralized design, including its data encoding, functional entities, and operational procedures, with distinguishing features summarized at the end of the section. Section 3 introduces a distributed design derived from the centralized model, along with its differentiating characteristics and configuration-management procedures. Section 4 describes a federated design capable of hosting multiple interworking centralized and distributed systems. Section 5 presents an optional enhanced encoding for Bluetooth 5.0 and discusses coexistence between BLE and Bluetooth 5.0. Section 6 summarizes the distinguishing features and limitations of the proposed design.

Centralized Contact Tracer

This section presents a centralized contact-tracing architecture, including its proximity-data encoding, functional entities, and operational procedures. The encoding scheme supports anonymization, encryption, and routing. Identifiers are rotated using multiple keys and multiple anonymizing functions. The resulting identifier is sufficiently compact to operate with both intrusive and non-intrusive BLE-based exchanges. The use of multiple keys minimizes data loss in the event of key exposure and provides a controlled mechanism for phasing in new keys and retiring older ones.

A proximity record (PR) originates at a server cluster and returns to the same cluster for post-processing after undergoing the transformations illustrated in Fig. 1. The transformed PR retains enough information to be mapped back to its original content. This study employs three identifiers. Manual contact-tracing activities use the user's MSISDN, although any identifier—including random values mapped to user IDs— may be used when scalability is not a concern.

An anonymizing function maps the MSISDN to a 12-byte anonymized proximity identifier (AP-ID). This function reduces recognizable patterns in the MSISDN while scrambling the bit structure. The AP-ID, along with two additional fields—cluster number (CNO) and anonymizing-function pointer (AFP)—is encrypted using any 128-bit block-cipher method. The resulting 16-byte encrypted string serves as the anonymous user identifier for proximity-data exchange. This group of fields is referred to as the rotating proximity identifier (RP-ID). The CNO identifies the server cluster responsible for post-processing. The AP-ID is the anonymized user identifier, and the AFP identifies the anonymizing function used.

A server entity constructs a type-1 proximity record (PR1) containing the RP-ID and an encryption-key index (KIX). Multiple PR1s may be generated for a device using different anonymizing functions and encryption keys. These PR1s are delivered asynchronously and periodically to the device. Upon receiving a PR1, the registered App constructs a type-2 proximity record (PR2) and exchanges it with peer devices. When a smartphone scans and receives a PR2 broadcast, it constructs a type-3 proximity record (PR3) and stores it in its proximity-data log after consolidation. Consolidation aggregates proximity distance and duration based on the RP-ID and the source address.

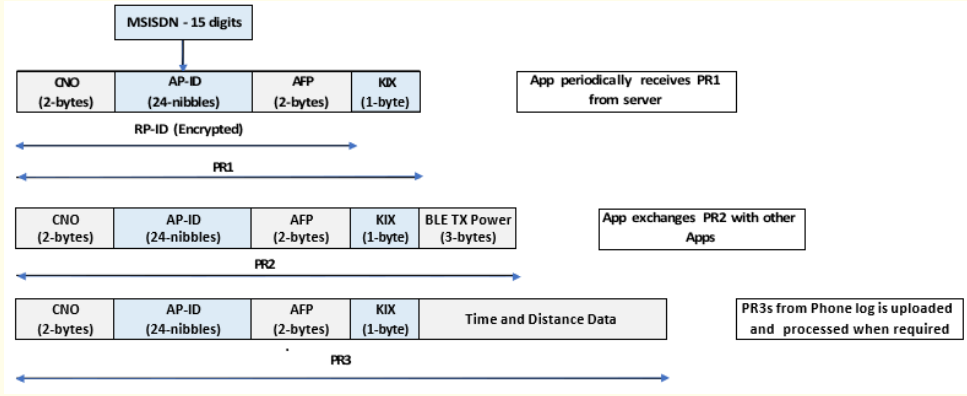


Figure 1: Proximity Records with Proximity Data Encoding.

Anonymizing functions rotate over time. Such a function may consist of a hashing algorithm, an encryption operation, a sequence of logical transformations, or any combination thereof. The key index (KIX) identifies the encryption key and its parameters. The key referenced by KIX encrypts the preceding 16 bytes. Encryption may use standard or proprietary block-cipher methods. Storing only function pointers and key indexes preserves privacy and anonymity while enabling the server to reconstruct the original content during post-processing.

The App constructs PR2 from PR1 and broadcasts it in the BLE ADV_IND payload. Devices within operational range scan and receive this message. The exchanged proximity data (PR2) includes the RP-ID, the KIX, the received signal strength, and other BLE overhead. These fields are encoded using a length-type-value (LTV) structure within the BLE ADV_IND payload, which also includes a UUID uniquely identifying PR2. Appendix B provides the encoding details for the 31-byte BLE ADV_IND payload. The proximity data uploaded for post-processing (PR3) includes PR2, timestamp, proximity duration, and proximity distance. The design and the logic for proximity-data consolidation may be App vendor specific.

Figure 2 presents the functional entities that constitute centralized architecture. The server cluster comprises six primary components: the registration server, anonymizer, proximity-data (PD) processor, PD dispatcher, status-action dispatcher, and a semi-manual uploader. Their roles are summarized below.

1. ***Registration Database***: Upon application registration, the user's MSISDN and associated metadata are stored in the registration database.
2. ***Proximity Database***: This repository maintains all uploaded and processed PR3 records and serves them to downstream processing entities.
3. ***System Tables***: Three tables support the core operations: - ***Key table***, which stores block-cipher keys and their parameters; - ***Anonymizing-function table***, which maintains the set of anonymizing functions and their parameters; - ***Dispatcher-address table***, which maps each cluster number (CNO) to the corresponding post-processing server.
4. ***Registration Server (Registrar):*** The registrar acts as the system's entry point, handling registration requests from user devices. It populates and serves the MSISDN and other application-specific data stored in the registration database.
5. ***Anonymizer***: The anonymizer sequentially retrieves MSISDNs from the registration database and transforms each into an AP-ID using an anonymizing function. The AP-ID is then encrypted with a designated key to form PR1, which is delivered asynchronously to the user device. Each device receives a batch of PR1s during this periodic sweep.
6. ***Proximity-Data (PD) Dispatcher***: The PD dispatcher uploads PR3 records associated with newly reported cases to the proximity database. It subsequently decrypts the records and forwards them securely to the appropriate PD processor.
7. ***Proximity-Data (PD) Processor***: The PD processor recovers the MSISDN from the AP-ID using the corresponding de-anonymizing function. It then applies consolidation logic to summarize proximity-distance and duration for each MSISDN. After consolidation, the set of MSISDNs meeting CDC-defined exposure criteria is forwarded to the status-action dispatcher.
8. ***Status-Action Dispatcher***: This entity notifies users of their exposure status and prescribes the required follow-up actions (e.g., isolation or quarantine). At this stage, automated digital tracing transitions into manual public-health workflows.
9. ***Semi-Manual Uploader***: This component enables manual insertion of proximity data into the system, supporting partial data recovery when a reported case does not have an associated device.
10. ***Anonymizing Function***: Each anonymizing function maps an MSISDN (in BCD format) to a 24-nibble AP-ID while removing recognizable MSISDN patterns and scrambling bit structures. The anonymizing and de-anonymizing functions share a common identifier.
11. ***Key***: Proximity data encryption employs either standard or proprietary block-cipher algorithms. The anonymizer performs encryption, and the PD dispatcher performs decryption. Each server maintains local copies of the system tables in addition to the persistent master copy.
12. ***Data Analytics***: Advanced analytics and data-mining applications fall outside the scope of this study.

App Registration

The registration procedure is initiated once the user installs the application on a smartphone. The app locates the registrar server through a DNS lookup, using either the MSISDN's country code or a combination of country code and service-provider identifier. This step constitutes a synchronous, Internet-based interaction between the device and the server. If that fails it can be executed with asynchronous interaction using something like short message service (SMS).

Upon receiving the request, the registrar validates the MSISDN, collects the user's profile information, and establishes a subscription that binds the app to a designated anonymizer. The registration response includes an initial batch of PR1 identifiers. After this initial exchange, the app continues to receive PR1s asynchronously from the anonymizer.

To maintain this binding, the app periodically renews its subscription. The renewal interval is intentionally set to be an order of magnitude longer than the PR1 distribution period. If the anonymizer does not receive renewal messages for two or three consecutive renewal cycles, the subscription is considered expired, and PR1 transmission to the app is discontinued. Conversely, if the app does not receive PR1s for two or three successive distribution periods, it proactively issues a renewal request.

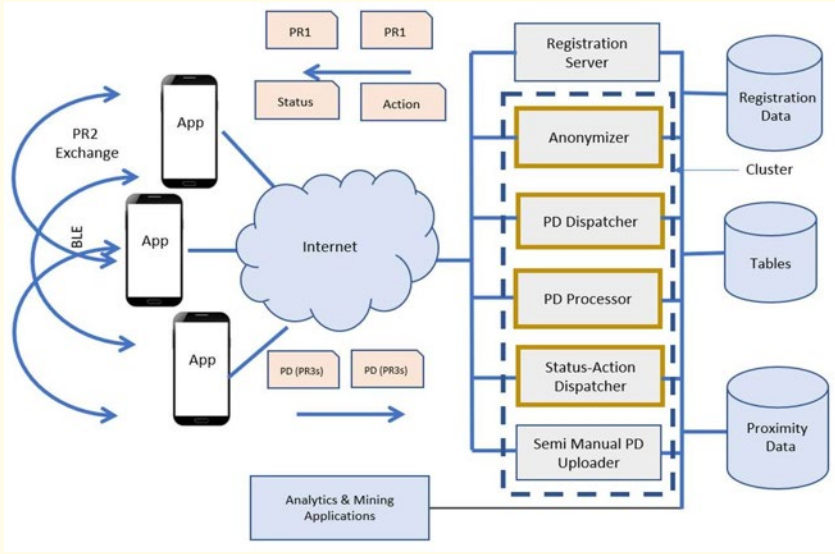


Figure 2: Functional Entities of Centralized Contact Tracing System.

Renewal requests are distinguished from first-time registrations by the presence of an anonymizer identifier. Initial registrations either omit this identifier or include a placeholder value. If the anonymizer identifier is outdated or incorrect, the system treats the request as a fresh registration: the supplied data is added to the database, and a new batch of PR1s is issued immediately. Renewal records are appended to the database sequentially in batch mode.

The app may also reinitiate registration when its profile changes—for example, due to roaming or other mobility-related conditions that alter its operational context.

Rotating Identifier Generation and Distribution

The anonymizer is responsible for constructing and distributing PR1 identifiers to all registered applications. Together, the anonymizer and the PD processor form the two central functional entities of the design. During each sequential sweep of the registration database, the anonymizer iterates through the list of MSISDNs and delivers a batch of PR1s to every registered app. Devices may retain and use multiple batches concurrently in an interleaved manner; receipt of a new batch does not require retirement of an existing one. This design—batch distribution combined with extended PR1 lifetimes—ensures that the anonymizer has sufficient time to complete each sweep without starving applications of identifiers. Renewal updates generated by apps are also posted to the database during this sweep. Each app manages PR1 lifetimes locally using timestamps, in contrast to other centralized architectures [3], where identifier lifetimes are dictated by the server.

For each MSISDN retrieved from the database, the anonymizer selects an anonymizing function and encryption key using a scheduling mechanism that draws from the corresponding system tables. It then constructs a set of PR1s by applying the chosen mapping function and encrypting the resulting AP-IDs. Once the batch is generated, the anonymizer transmits the PR1s to the app using asynchronous message transport. This process is repeated for every MSISDN in the registration database, and any pending updates—such as subscription renewals—are committed to the database during the sweep. The sweep is executed periodically.

The use of asynchronous, pub-sub-based message transport provides reliable and efficient delivery of PR1s. It eliminates the need for the anonymizer to locate devices or maintain synchronous communication channels. Both Android and iOS platforms support

encrypted asynchronous messaging, enabling an intermediary message server to store PR1s and deliver them whenever the device becomes reachable.

Proximity Data Exchange and Logging

Each arriving PR1 is time-tamped by the application and stored in its local buffer. This timestamp is scoped to the app and is used to determine when a PR1 should be retired. Applications may use PR1s in an interleaved manner, and a PR1 is retired when its timestamp exceeds a predefined threshold. For every active PR1, the app constructs a PR2 and broadcasts it to nearby devices. Other apps within operational range receive these broadcasts through periodic scanning. The exchanged proximity record (PR2) contains the transmitting device's PR1 and the BLE transmit-power level.

Upon receiving a PR2, the app appends a local timestamp and stores the record along with the received signal strength indicator (RSSI). The app then performs consolidation to compute both proximity duration and estimated distance. The proximity distance to the broadcaster is derived as a function of signal attenuation. Each consolidated PR3 stored in the log contains the PR1, the computed proximity duration, and the estimated proximity distance. Consolidation relies on both the transmitter's hardware address and the rotating identifier.

Post-processing consolidation is conditional and performed only when the user is identified as infected. In that case, the consolidation step aggregates all records associated with the same MSISDN.

Proximity Data Upload & Processing

When a subject is confirmed to be infected, the health authority—acting with the subject's consent— initiates the upload of proximity data from the subject's device to the PD dispatcher. The uploaded dataset consists of the proximity-data log covering the full incubation period. Upon receipt, the dispatcher decrypts each PR3 entry in the log and forwards the resulting records to the PD processor.

The PD processor applies the anonymizing function to recover the MSISDN associated with each record and performs consolidation of the proximity data. Records that satisfy CDC-defined exposure criteria are then transmitted to the status-action server. The status-action server communicates the exposure status and recommended follow-up actions to the corresponding applications. At this stage, automated digital tracing transitions into manual public-health workflows, such as counselling of exposed individuals.

Centralized Contact Tracer Design Summary

The centralized architecture described above addresses the three challenges outlined in Section 1. By employing a compact PR1 format, the design supports both intrusive and non-intrusive BLE-based proximity exchanges. The use of multiple rotating encryption keys replaces the single-key mechanism adopted in TraceTogether [3]. Keys can be phased in or phased out seamlessly, and the system can immediately discontinue the use of any compromised key during PR1 generation without interrupting normal operation—an ability not feasible in single-key designs. Key management is uniform across all servers, with a dedicated functional entity responsible for maintaining the key and anonymizing-function tables. These management activities operate independently of the core contact-tracing workflow, ensuring uninterrupted system performance.

The design also eliminates the need for random database access by anonymizing and encrypting user identifiers before distribution. During post-processing, the de-anonymizing functions reconstruct the MSISDN directly from the corresponding AP-ID through procedural mapping, avoiding random disk lookups.

The registrar server serves as the system's frontend and can be configured to distribute load across multiple clusters. This distribution model differs from those described in [4-6], offering greater flexibility in scaling the system according to application requirements. For example, clusters may be organized such that each one manages a distinct geographic region or administrative domain.

Distributed Contact Tracer - Design and Operation

The distributed design comprises multiple server clusters (Fig. 3) and retains the same data-record formats—PR1, PR2, and PR3—as the centralized architecture. The cluster-number field identifies the originating cluster within the distributed system: the most significant 8 bits denote the distributed contact-tracing system, while the least significant 8 bits identify the specific server cluster. No modifications are required on the application side; instead, servers incorporate additional functionality to interpret and operate on the extended cluster-number field.

Each server cluster contains enhanced instances of the core functional entities: one or more registration servers, an anonymizer, a PD processor, a PD dispatcher, a status-action dispatcher, and one or more semi-manual uploaders. A PR1 generated in one cluster may later arrive at a different cluster for post-processing as a PR3. To support this, the PD dispatcher includes a forwarding mechanism that routes PR3 records to their home cluster. The recommended default behaviour is as follows: if a PD dispatcher cannot process a received PR3 locally, it forwards the record either to its home cluster or to a designated processing cluster (described in Section 4). A dedicated table server provides key-table and anonymizing-function-table services to all clusters over a secure network.

The anonymizer and PD dispatcher incorporate client-side capabilities to request either the full key table or specific key-table entries from the table server. Similarly, the anonymizer and PD processor can request the complete anonymizing-function table or individual entries. The table server may also push update notifications to these components when table contents change. The PD dispatcher additionally supports forwarding proximity data to other dispatchers across clusters. A secure CT network enables inter-cluster message exchange.

Embedding the cluster number within encoded identifiers ensures that identifier spaces remain independent across clusters. This allows each cluster to use any 12-byte identifier—whether derived from user identifiers or randomly generated values mapped to user IDs—without risk of collision. By distributing processing across multiple clusters, the design effectively mitigates scalability constraints.

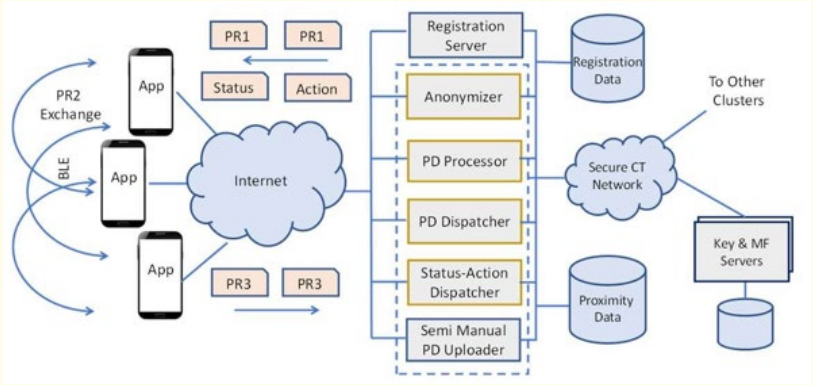


Figure 3: Functional Entities of Distributed Contact Tracing System.

The PD dispatcher first determines whether each PR3 record in the uploaded log belongs to its own server cluster. After decrypting each PR3, the dispatcher compares the cluster number embedded in the record with its local cluster identifier. Records that match are forwarded to the proximity-data (PD) processor within the same cluster for further processing, while non-matching records are routed to the designated PD dispatcher responsible for the corresponding cluster.

Each server cluster maintains its own registration and proximity databases. As a result, inter-cluster communication becomes an additional resource requirement. This inter-cluster traffic primarily consists of proximity-data records forwarded to their home clusters and periodic interactions with the table server. Other forms of traffic on this network are sparse and fall outside the core operational flow.

Except for the cluster-management procedures described in Section 3.1, all remaining cluster operations mirror those of the centralized contact-tracing system.

Distributed Contact Tracer - Cluster Management

The core contact-tracing workflow remains identical to that described in the centralized design, with the exception of proximity-data forwarding. The distributed contact tracer introduces additional cluster-management capabilities that enable the system to create, resize, and retire clusters as needed. When the database grows to a point where a full sweep of AP-ID distribution risks exceeding the cycle time, these cluster-management operations allow the system to scale without disrupting ongoing pre- or post-processing activities.

1. ***Creating a new server cluster***: A new cluster is instantiated and activated. Once operational, the registrar begins binding newly registering applications to the anonymizer of this cluster. From that point onward, the assigned apps receive PR1s from the new anonymizer.
2. ***Migrating to a scaled-up cluster***: A scaled-up cluster is activated with its registration database initialized by copying data from the existing cluster. The registrar then binds all new registrations and renewals to the anonymizer of the new cluster, while buffering renewal entries for later removal from the old cluster. During the next sweep, the new anonymizer begins distributing PR1s to the assigned apps. Any PR1s received from the old anonymizer are ignored once the app receives its first batch from the new cluster. The app may complete or discard its current PR1 set before switching to the new identifiers. The old anonymizer is phased out after the registration-expiry period, and the remaining servers retire after one full incubation period. Importantly, changing the anonymizer does not affect post-processing of proximity data. The decommissioned cluster number may be reused.
3. ***Splitting a large cluster into two smaller clusters***: MSISDNs selected for migration are marked in the original cluster's database. These marked entries are copied to a new cluster, which is then activated to accept new registrations. At the end of the next PR1 distribution cycle, all marked apps are bound to the new anonymizer. Marked entries are removed from the original database after the renewal-expiry period. The new anonymizer then begins its distribution cycle. An inactive app may be unaware of its reassignment and may fail to receive PR1s when it becomes active; upon attempting renewal with the old anonymizer identifier, it is treated as a first-time registration and issued a fresh batch of PR1s.

In summary, the distributed architecture enables flexible partitioning and allocation of processing loads across multiple clusters. Each cluster functions as an autonomous system, and post-processing may be performed locally, in the originating (home) cluster, or in a designated processing cluster. The centralized contact tracer is therefore a special case of this architecture, consisting of a single cluster.

The distributed design evolves directly from the centralized model while preserving its non-intrusive data-exchange framework. As a result, it inherits the core operational characteristics of the centralized system. The principal enhancement is the introduction of PR-forwarding capability. The table server added in this design is lightweight, and its table-management operations remain decoupled from the primary contact-tracing workflow. Compared with other distributed approaches [4-6], this architecture offers greater flexibility in distributing processing loads, relying on dedicated servers rather than user devices. Consequently, the excessive resource consumption, data-loss-related unreliability, and substantial overheads reported in [4-6] are avoided.

Although the distributed design is well suited for deployment within a large national environment, it is not adequate for global operation, where each cluster must retain sovereignty over its proximity data. The flexibility that allows any cluster to process any PR is acceptable within a single jurisdiction but may expose citizens' data to clusters located in other countries. The next section introduces

the global design, which extends the distributed architecture to address these cross-jurisdictional privacy and control requirements.

Federated Contact Tracer Design - Operation and Evaluation

The federated contact-tracer architecture extends the distributed design by enabling multiple distributed systems to interoperate as a unified federation (Fig. 4). To support this evolution, the encodings of CNO, AFP, and KIX are retroactively revised across all three designs (Fig. 5). The resulting federated model satisfies the requirements of a global, evolvable solution, accommodating revisions without disrupting existing deployments.

The federated system introduces a two-level hierarchy, providing two layers of security and two layers of data-routing control. The anonymizing function using KIX-IN protects user identity when proximity data traverses other distributed systems. The encryption with KIX-OUT ensures confidentiality against external attacks.

As proposed in Section 3, the cluster number of Fig. 1 is divided into two fields. The most significant 8 bits uniquely identify a distributed system within the federation, while the least significant 8 bits identify a specific cluster within that distributed system (Fig. 5). Operationally, the cluster number maps to the PD dispatcher responsible for handling proximity-data exchanges.

Within each distributed system, every PD dispatcher serves as a gateway for inter-cluster proximity-data exchange. Additionally, one designated PD dispatcher—identified by a cluster-number value of zero—acts as the global gateway, enabling proximity-data exchange with other distributed systems in the federation. In this encoding, the size of the anonymizing-function pointer is reduced to 1 byte, reflecting the streamlined hierarchical structure.

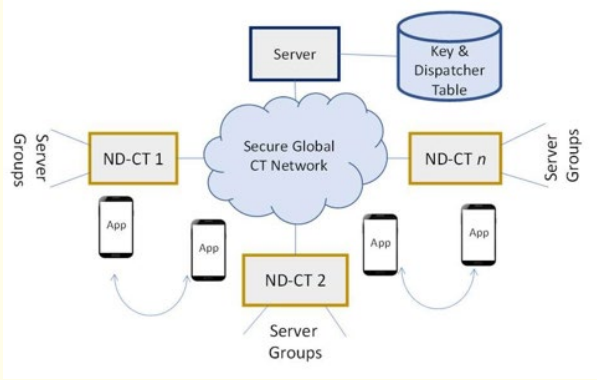


Figure 4: Federated Contact Tracing System.

Two proximity-data dispatcher address tables are maintained in the federated architecture: one globally scoped and one locally scoped within each distributed system. The global dispatcher table maps the most significant 8-bit field of the cluster number to the corresponding global gateway. In contrast, the least significant 8-bit field is meaningful only within an individual distributed system. This encoding enables a PD dispatcher in any distributed system to forward proximity data (PR3) to any other system in the federation while preserving user anonymity and concealing the address of the originating PD dispatcher.

Because the global dispatcher table and the global key table must be trusted by all participating systems, their administration requires a neutral international authority, such as the World Health Organization (WHO). A distributed contact-tracing system becomes part of the federation by adding an entry to the global dispatcher table, and its membership terminates when that entry is removed.

The 12-byte tuple consisting of MSISDN, AFP, and the 1-byte cluster number is first anonymized using a designated function and then encrypted with the cipher identified by KIX-IN. Any 64-bit block cipher supporting ciphertext stealing [19-24] may be used, allowing encryption without padding and ensuring that ciphertext length matches plaintext length. The encoding supports 2^{16} combinations of anonymizing functions and cipher selections, and rotating across this large space significantly strengthens anonymization robustness. Both proprietary and standardized ciphers are permissible.

Two key tables are maintained:

- KIX-IN keys are managed locally within each distributed system.
- KIX-OUT keys are managed globally and are accessible to all clusters in the federation.

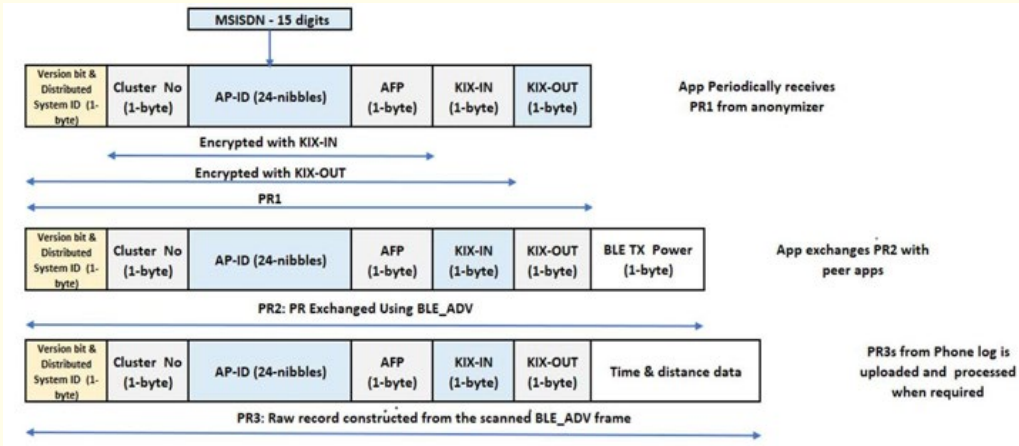


Figure 5: Enhanced PR1, PR2, and P3 Encodings.

The PD dispatcher incorporates additional logic to interpret and act on the cluster number. When processing a PR3 record, the dispatcher applies the following decision sequence: if the most significant 8 bits of the cluster number differs from its own, the PR3 is forwarded to the system's federation gateway. If they match, the PR3 is either processed locally or routed to its home cluster using the least significant 8 bits of the cluster number. When a global gateway receives the record, it consults the global dispatcher table to forward the proximity data to the appropriate gateway within the federation. To minimize overhead, proximity data is transmitted in batches.

The federated contact-tracer design provides decentralized control over proximity data while enabling seamless, anonymous exchange of proximity information among participating distributed systems. Membership management is performed through simple table updates, making it a low-overhead operation. The expanded BLE 5.0 payload capacity can be leveraged to support an enhanced proximity-data encoding that incorporates 128-bit ciphering and a message digest, strengthening both security and integrity.

BLE 5.0 Migration

The Bluetooth LE (BLE) broadcast payload is limited to 31 bytes, constraining the size of the proximity record (PR) and restricting the design to 64-bit block ciphers with 64-bit keys. Bluetooth 5.0, however, supports payloads up to 244 bytes while remaining backward-compatible with BLE. This section describes a backward-compatible PR encoding for Bluetooth 5.0 (Fig. 6) that enables the use of 128-bit block ciphers and 128-bit keys. In this enhanced encoding, the 15-digit MSISDN is mapped to a 22-byte value—representing the only significant change relative to the earlier encoding (Fig. 1). When 64-bit encryption is used with KIX-IN, ciphertext stealing is

unnecessary. A broadcast-and-scan coexistence model for the old and new encodings is presented in [16].

An application running on Bluetooth 5.0 can exchange both the legacy and enhanced PR2 formats. Each Bluetooth 5.0 app receives both record types within the same message distributed by its anonymizer. Server components and database entities require incremental updates to support the coexistence of the two PR formats. The anonymizer must be capable of generating two types of PR1s; the PD dispatcher must be able to download, process, and forward two types of PR3s; and the PD processor must decrypt and remap the MSISDN for both PR3 formats. A Bluetooth 5.0 app must therefore support exchanging both PR2 types, whereas a BLE-only app requires no modifications.

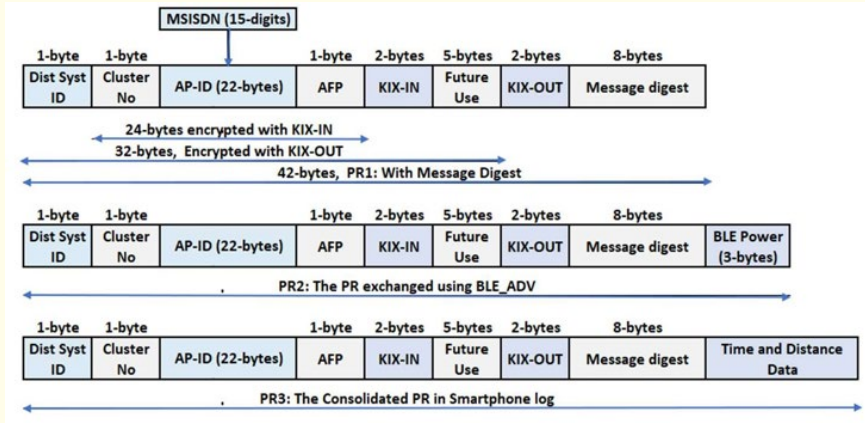


Figure 6: PR1, PR2, and PR3 Encodings for Bluetooth 5.0.

Summary

This study presents a unified proximity-data encoding framework applicable to both distributed and federated contact-tracing systems. The distributed architecture enables flexible allocation of processing load across multiple clusters, and the federated model extends this capability to interworking systems operating under different administrative domains. Conceptually, the distributed design generalizes our earlier centralized architecture, retaining its core principles while adding scalable routing and anonymization features. The proposed encoding and its associated procedures support large-scale deployments that can interoperate seamlessly without relinquishing local control over their respective proximity data.

Table 1 compares the proposed design with several earlier approaches [2-6]. The compact proximity-data encoding fits within the 31-byte BLE broadcast advertisement, enabling efficient wireless exchange. Moreover, de-anonymization of proximity data does not require random access to storage, a property essential for building scalable systems. Together, these two features form the foundation for high-performance, large-scale deployments.

In this design, the anonymized rotating proximity identifier is derived as a function of the underlying user identifier. The proposed encoding allows interworking systems to substitute any suitable identifier— such as an IP address or UUID—without altering the overall architecture. For privacy, the identifier is encrypted within the proximity record using one of several block-cipher options; both proprietary and standardized ciphers are supported.

Wireless tracking risks are mitigated through interleaved rotating identifiers generated using multiple anonymization functions and encryption methods. This diversity significantly reduces the predictability of identifiers and strengthens protection against tracking attacks.

S. No.	*Feature*	*Exposure Notification* [4-6]	*TraceTogether* [3] *& TraceCORONA* [11]	*This Study*
1	PD payload can be exchanged within BLE broadcast	Yes	No	Yes
2	Potentinal operational bottlenecks with random IDs	No	possible	No
3	Support for different ID types including Random ID	Random ID	Random ID	Yes
4	Encoding supports inteworking systems	No	No	Yes
5	Rotating ID requires encryption	No	No	Optional
6	Multi-Key Support	Not required	Single Key*	Present
7	Proprietary encryption supported	No	Possible	Yes
8	Proximity data routable for interworking	No	No	Yes
9	Support for flexible capacity distribution for scaling	No	Centralized design	Present
10	Support for interleaved rotating ID	No	No	Yes
11	Unreliable result when device is lost or not present	Possible	Not an issue	Not an issue
12	Coexistence of different type of ID encodings	No support	No support	Yes
** TraceCORONA allows negotiation of keys for certain procedures*				

Table 1: Comparison with other designs.

References

1. Public Health Guidance for Community-Related Exposure. COVID-19 Guidance, Center for Disease Control and Prevention, US (2020). https://www.cdc.gov/coronavirus/2019-ncov/php/public-health-recommendations.html.
2. Narayanan Hari TS. "Contact Tracing Solution for Global Community (October 12, 2022)". Computer Journal, Oxford Academic 64.10 (2021): 1565-1574.
3. Martin., T et al. "Demystifying COVID-19 Digital Contact Tracing: A Survey on Frameworks and Mobile Apps". Hindawi, Wireless Communications and Mobile Computing (2020).
4. Ahmed N., et al. "A Survey of COVID-19 Contact Tracing Apps". IEEE Access 8 (2020): 134577-134601.
5. Muhammad Shahroz., et al. "COVID-19 digital contact tracing applications and techniques: A review post initial deployments". Transportation Engineering 5 (2021): 100072.
6. Contact Tracing App – India (2020), Arogya Setu Official site: https://www.mygov.in/aarogya-setu-app/
7. Jason Bay., et al. "BlueTrace: A privacy-preserving protocol for community-driven Contact Tracing across borders". A White paper from Government Technology Agency, Singapore (2020). https://TraceTogether.io/static/TraceTogether_whitepaper-938063656596c104632def383eb33b3c.pdf
8. Exposure Notification API Preliminary 1.2. Apple & Google joint Contact Tracing Initiative (2020). https://covid19.apple.com/contacttracing
9. Exposure Notification Bluetooth Specification 1.2. Apple & Google joint Contact Tracing Initiative (2020). https://covid19.apple.com/contacttracing
10. Exposure Notification Cryptography Specification preliminary 1.2. Apple & Google joint Contact Tracing Initiative (2020). https://covid19.apple.com/contacttracing
11. Thien Duc Nguyen., et al. "Digital Contact Tracing Solutions: Promises, Pitfalls and Challenges". IEEE Transactions on Emerging

Topics in Computing 12 (2024): 483-495.
12. PWC Automatic Contact Tracing: https://www.pwc.com/us/en/products/check-in.html
13. Narayanan Hari TS. "Loss of Smartphones, and its Implications to Contact Tracing (July 20, 2020)". International Journal of Engineering Research and Technology (IJERT) 9.7 (2020).
14. Worldometer on Corona Stats. A Web portal that reports COVID-19 Stats and other useful information (2020). https://www.worldometers.info/coronavirus/country/us/
15. Is The Data on Your Business' Digital Devices Safe? Steve Olenski. Forbes article (2017). https://www.forbes.com/sites/steveolenski/2017/12/08/is-the-data-on-your-business-digital-devices-safe/?sh=7d5b47274c6a
16. Narayanan Hari TS. and Narayanan Spatika. "Scheduling Proximity Data Exchange for Contact Tracing". International Journal of Swarm Intelligence and Evolutionary Computation, Walsh Medical Media 11.8 (2022): 1000267.
17. Bluetooth Trends in Smartphones and the Implications for Medical Device Software, Medical Device and Diagnostic Industry (2021). https://www.mddionline.com/digital-health/bluetooth-trends-smartphones-and-implications-medical-device-software
18. Bohan Zhang., et al. "DISTERNING: Distance Estimation Using Machine Learning Approach for COVID-19 Contact Tracing and Beyond". IEEE Journal on Selected Areas in Communications 40.11 (2022): 3207-3223.
19. Wheeler David., et al. "TEA, a tiny encryption algorithm". Lecture Notes in Computer Science. 1008. Leuven, Belgium: Fast Software Encryption: Second International Workshop (1994): 363-366.
20. Matthew D Russell. "Tinyness: An Overview of TEA and Related Ciphers". Archived (2007).
21. Roger M Needham and David J Wheeler. Tea extensions (1997). http://www.cix.co.uk/~klockstone/xtea.pdf
22. Treyfer, A Block Cipher/MAC for Algorithm: https://en.wikipedia.org/wiki/Treyfer
23. James L Massey. "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm". Fast Software Encryption (1993): 1-17
24. Phillip Rogaway, Mark Wooding and Haibin Zhang. The Security of Ciphertext Stealing, Fast Software Encryption 7549 (2012).

Appendix A

The following illustrations compare the behavior of different anonymization methods. Post-processing operations are executed in batch mode at two-hour intervals. The number of new cases used in these examples corresponds to the average case count per batch period (U.S. data for January 7, 2022). The average number of proximity records with unique RP-IDs per day is estimated from the broadcast schedule described in [12], and this value naturally varies with population density.

The parameters used in the illustrations are as follows:

- ***Average number of new cases per post-processing interval***: 75,000 ([13]).
- ***Incubation period***: 15 days (CDC, USA).
- ***Average number of proximity records with unique identifiers in a user log per day***: 1,000 ([12]) - This value depends on several factors, including identifier-rotation frequency, device encounters within the operational range, broadcast and scanning schedules, and exchange frequency.
- ***Average storage access time for a random 150-byte record***: 0.02 ms - Assumes SSD performance is ~100× faster than HDD for random access.
- ***Number of smartphones in the U.S.***: 270 million ([14]).

Server Generated Identifier [3]: When an individual is identified as infected, the proximity records stored on the person's smartphone are consolidated, de-anonymized, and correlated to determine close contacts. Consolidation aggregates contact attributes (time and distance) for each rotating identifier; de-anonymization maps each rotating identifier to its corresponding user identifier; correlation then summarizes contact data per user. These operations are executed in ***two-hour batches***.

For an illustrative workload of ***75,000 new cases***, with each device storing ***15,000 proximity records*** (corresponding to a 15-day

incubation period), the anonymization time is:

$$75{,}000 \times 15{,}000 \times 0.02 \text{ ms}$$

This exceeds ***6 hours***, whereas the operational cycle requires completion within ***2 hours*** to avoid bottlenecks.

Device Generated Identifiers [4-6]: In this model, when a subject tests positive, the anonymization data for the entire incubation period is broadcast from a centralized server to all ***270 million smartphones***. The scale of this operation is:

$$270 \times 106 \times 15 \times 75{,}000 \times \langle \text{daily anonymization-data size} \rangle$$

This distribution occurs every ***two hours***. Each smartphone must then locally recompute anonymization functions for all ***75,000 patients*** and compare the results against its own proximity log. If each validation requires matching ***1,000 records***, the computational load scales with:

$$75{,}000 \times 15 \times 1{,}000$$

Devices must also perform correlation and consolidation on any matched records. Although matches are rare, the checking must be repeated on ***every device***, every ***two hours***, to maintain reliability.

A smartphone offline for more than two hours cannot complete this process. When a lost device is replaced, the new phone requires ***15 days*** before its negative results (no contact) become reliable; positive results remain unaffected. These limitations are inherent to this design.

De-anonymization in the Proposed Study: In the proposed architecture, when a server receives proximity records for a newly identified case, it first consolidates the records by rotating proximity identifier (time and distance). It then performs decryption and de-anonymization on the consolidated set, followed by a second consolidation step based on user identifiers. Crucially, ***none*** of these operations require random storage access.

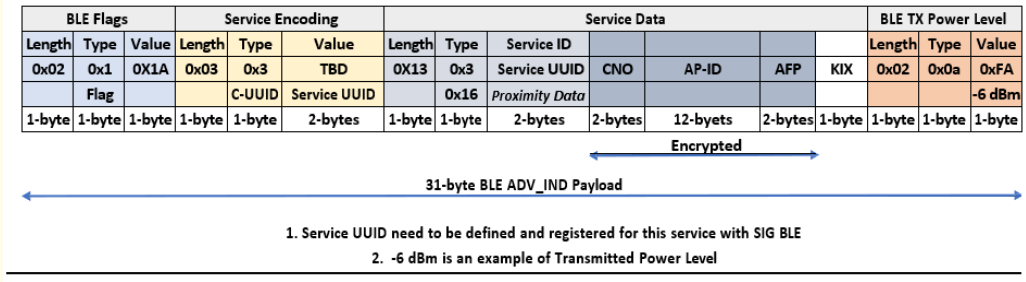
Processing is performed on ***dedicated distributed servers***, with each U.S. state capable of operating its own specialized node. A single server may handle data for ***5,000 new patients***, each with ***15,000 proximity records***, yielding a workload proportional to:

$$5{,}000 \times 15{,}000$$

This load is well within the capabilities of a server-class system, which is inherently more reliable and available than user devices. However, because the anonymized and encrypted identifier is ultimately a function of the user identifier, the design retains a degree of vulnerability.

Appendix B

Figure 7 illustrates how the 31-byte proximity record is encoded within the BLE ***ADV_IND*** payload. The 17-byte Service Data field carries the proximity record defined in Figure 1. The ***BLE TX Power Level*** encodes the transmitter's signal strength, which the receiver uses to estimate its distance from the source. The proximity-data format of Figure 5 can also be represented by splitting the AFP into two single-byte fields. Together, these two encodings support all three system architectures—***centralized, distributed, and federated***. The BLE Flags and Service Encoding fields constitute BLE-specific overhead.



BLE Flags			Service Encoding			Service Data							BLE TX Power Level		
Length	Type	Value	Length	Type	Value	Length	Type	Service ID					Length	Type	Value
0x02	0x1	0X1A	0x03	0x3	TBD	0X13	0x3	Service UUID	CNO	AP-ID	AFP	KIX	0x02	0x0a	0xFA
	Flag			C-UUID	Service UUID		0x16	*Proximity Data*							-6 dBm
1-byte	1-byte	1-byte	1-byte	1-byte	2-bytes	1-byte	1-byte	2-bytes	2-bytes	12-byets	2-bytes	1-byte	1-byte	1-byte	1-byte

Figure 7: Proximity Data Encoding with BLE ADV_IND Message.