CONTACT TRACING WITH BLUETOOTH LE - A STATUS REVIEW

Hari T.S. Narayanan

Abstract- The manual contact tracing process enables health care authorities to identify subjects to be tested for containment of an infection. The technology component of this process has gained traction recently due to the prevalence of smartphones and the presence of Bluetooth LE (Low Energy) in these phones. The technology component adds agility and scalability to manual effort by logging proximity information and identifying potential subjects. Every smartphone can record the identity of other smartphones that have come in its proximity using Apps and other services. A positively tested patient’s smartphone log is used to find the other smartphone users who were in the proximity of the infected subject recently. Once identified, the people in the group are contacted for testing. This needs to be done without compromising the privacy of any of these subjects, without revealing the location and other private information that are sensitive. In all possibility, we will be seeing many more applications based on such proximity data. The current breed of contact tracing applications is built with Bluetooth LE technology for data gathering and complemented by secure server technology. In this paper, we evaluate two contact tracing initiatives built with Smartphone Bluetooth LE technology for their privacy and performance. We also identify the list of research and development tasks ahead of us in this emerging area based on this review.

Key Words: Contact Tracing, Bluetooth LE, Proximity, Exposure Notification, Encounter Message, Privacy, Performance

I. INTRODUCTION

The term Contact Tracing refers to the process of collecting, storing, and processing data to identify the subject group which needs to be tested for an infection. Contact Tracing is not a complete solution; it enhances the manual contact tracing [1] by adding agility to tracking and identifying potential subjects. Contact Tracing Apps are put to use in China, South Korea, Australia, Singapore, India and many other countries with the current spate of COVID-19[2]. Almost all these initiatives are built around Bluetooth LE technology [3] present in smartphones. This paper is topical, wherein, we review two recent Bluetooth LE based initiatives that include privacy requirements.

The Bluetooth LE is one of the popular Personal Area Network (PAN)[4] wireless technologies. A device built with Bluetooth sends out periodic advertisement messages to indicate its presence. This message includes device identifications – both logical and hardwired. This message enables other devices to locate the advertising device for connectivity. Earlier versions of Bluetooth were power hungry, and due to this, the Bluetooth in smartphones were recommended to be activated only when in need. This changed with Bluetooth Low Energy (BLE) or Bluetooth 4.0 with its power optimization features. The current version of Bluetooth is 5.0[5]. The contact tracing type of applications require Bluetooth to be active all the time, even when the connectivity to cellular network is not there (Example: during flight mode).

In this paper, we are reviewing two contact tracing initiatives that preserve privacy requirement: the first one is, Exposure Notification[6], a joint initiative by Apple and Google; the second one is, TraceTogether[7], an initiative from Singapore Government’s Ministry of Health. Both initiatives assume the presence of Bluetooth 4.0 or more for their respective solutions. This paper is divided into five sections. The next section is a short introduction to Bluetooth LE and Unix Time. Both are key components of both the solutions that we review in this paper. The

1 NetToolsConsulting, Chennai, TN, INDIA
initiatives themselves are reviewed in Section 3 and Section 4 respectively. The privacy and performance issues are presented in the respective sections at the end.

II. BLUETOOTH OVERVIEW

Bluetooth LE (BLE) is designed for Personal Area Network that operate in low tens of meters range. Often, devices with BLE adapters operate in Master-Slave mode. These devices use BLE advertising and scanning procedures to establish connectivity before exchanging data. The Advertisement messages are sent by Slave to indicate its presence in the neighborhood. The Scanning is employed by the master to find the slave device its user is looking to connect. Every Bluetooth device is capable of advertising and scanning. There are restrictions on advertisement frequency and advertisement message size to conserve power on battery operated devices. Smartphones have less of this burden; thus, they can advertise and scan more liberally than slave devices like wearables.

2.1 BLE Advertisement

Bluetooth physical layer operates over 2.4GHz band (Figure 1). There are 40 channels in all, starting from 0 to 39. Channels 37, 38, and 39 are dedicated for advertisement packets and all other channels are used for data packets. In BL 5.0, the advertisement can go over all the 40 channels. The channels 37, 38, and 39 are spread over the Bluetooth frequency band as shown in Figure 1.

![Figure 1. Bluetooth Advertisement and Data Channels](image)

2.2 BLE Advertisement Interval

A device, specifically a slave device gets into advertising mode, when turned on. It starts sending advertisement packets periodically on each advertising channel. A master can also be configured to advertise. The advertisement sequence is illustrated in Figure 2. The *AdvInterval* is the minimum duration between two advertisements. This value is in the range 20 milliseconds to 10.24 seconds in steps of 0.625 milliseconds. The predictability of *AdvInterval* is eliminated by adding a random delay to it. The delay is a pseudo-random value from 0 to 10 milliseconds. This random value helps to reduce the possibility of lock-step collisions between advertisements of different devices. This arrangement gives a wide range of possible values for advertisement frequency. There are guidelines to configure advertising frequency. The normal *fast* advertisement interval is in the range 100-500 milliseconds; *aggressive* interval is in the range 20-50 milliseconds. Once connected to a master this rate can be drastically reduced. Low advertisement frequency and small payload conserve power. On the other hand, low frequency could cause delay in connecting to a slave device. In general, 500-1000 milliseconds are a sweet spot for advertisement frequency in most products.

The advertisement channels carry seven different PDU types including the four advertising PDU types listed here. The one advertising type that is relevant for the discussion is *ADV_IND* (Connectable undirected advertising) type shown in Figure 3. This packet type supports connectable, undirected advertising and it is used when a Slave/Peripheral device is powered up for the first time and it is looking for other nodes in the proximity. This typically represents a factory default state.
1. **Connectable directed advertising**: Devices that need to quickly connect to a specific device use this type. It has already initiated some communication with this master.

2. **Non-connectable undirected advertising**: Devices that only transmit, such as beacons, use this type. Smartphones do that.

3. **Connectable undirected advertising**: Most other devices use this advertisement type. Slaves send these messages before they send type 1 message.

4. **Scannable undirected advertising**

![Figure 2. BLE Advertisement - Timing Diagram, Advertisement PDU Types](image1)

![Figure 3. ADV_IND (Connectable undirected advertising) PDU Format](image2)

The PDU format of ADV_IND is shown in Figure 3. The advertisement packet has up to 31 bytes for payload that can be used to advertise additional information about the device. The first 6-bytes are advertiser’s ID (MAC address). While advertising, the device can indicate if it is using a random MAC address or using its own MAC address. This becomes important in protecting the identity of the user when another device wants to send a message to the actual MAC address. The fields in payload are coded in the form of **Type-Length-Value**. A comprehensive list of payloads and their descriptions are available at Bluetooth SIG site [8]. The most common payloads are the following.

- Complete Local Name
- Shortened Name
- Power Level

2.3 **BLE Scanning**

Unlike advertising, the scanning is dominated by software features that implement it. Scanning is bound to a suitable **callback function**, this callback function is executed on the occurrence of various scanning events, including the one that finds a matching advertisement. Scanning can be started and stopped using the appropriate library calls. The **proximity** is often measured as a function of difference in signal strength between the source and destination.

Scanning is also a power draining process. Both Android and iOS reduce the power drain by reducing the scanning frequency when the App goes from foreground to background. Both operating systems will generate less
advertisement discovery events; for instance, reporting multiple discoveries of the same device is done only once. In foreground scanning operation, power is conserved by stopping the scan the moment the peripheral device is found, or the set time limit has reached. Scanning interval needs to be big enough to reduce usability issue and users’ frustration. Scanner can look for a specific type of device (slave) or devices that fit certain filter category. The scanning parameters in Android are used to vary the intensity of scanning. Android allows the developer to make use of scanning parameters for optimizing the scanning procedure. Refer to the latest Android API for a complete list of these parameters and their applications.

2.4 Unix Epoch Time

The Unix Time is used in both initiatives presented in this paper for various purposes, and so a short overview of Unix Time is in order. The Unix epoch or Unix time is the number of seconds that have elapsed since January 1, 1970 00 hours UTC, not counting leap seconds. There is API support in all languages to get Unix time. Unix Time is represented as a 32-bit integer. The initiatives presented here, use Unix time for Timestamping, specifying the valid duration of messages, and more importantly to generate Proximity Identifiers that anonymizes the user. Network Time Protocol (NTP) is another Timing service with an epoch of January 1, 1990 00 hours UTC. NTP includes leap seconds unlike Unix time. The NTP gives an accurate value of current time. There are APIs to convert NTP time to Unix Time. The NTP time is offered as Internet service using atomic stratum clock.

III. CONTACT TRACING WITH EXPOSURE NOTIFICATION (APPLE & GOOGLE INITIATIVE)

In early May of this year, Apple and Google announced a version of contact tracing initiative built with Bluetooth LE. A new broadcast payload type, Exposure Notification, is added to Bluetooth advertisement message PDU to collect and compute privacy preserving proximity data. The initiative includes the specification of the Exposure Notification message and the related broadcast, scanning procedures needed to build the contact tracing applications. The initiative also specifies a crypto system to generate an anonymised rotating proximity identifier (RPI). This identifier and encrypted Bluetooth vendor information are the content of Exposure Notification payload.

The App component, using Bluetooth transmits Exposure Notification periodically. This notification is received and stored by instances of the same App or interworking App in other phones in the proximity. If a subject is tested positive for an infection, then this data is retrieved to identify other subjects for pro-active testing. This is done with subject’s approval. Subject can refuse to share this information. In either case, the subject’s decision is protected. None of the data contains anything that points to the subject or the subject’s location at any time. If there is no issue with the subject, then the data in the phone is aged out periodically. Rest of the section describes the operational procedure of this initiative.

3.1 Exposure Notification Message

The user downloads the App from Internet and installs it on his/her smartphone. The App generates a master key, Temporary Exposure Key (TEK). This is done with the crypto system running on smartphone with no outside help. All other identifiers are derived from this key and in some cases with discretized Unix Time. The TEK, as the name suggests, is replaced periodically (every 24 hours) for privacy consideration. The Exposure Notification message is a normal Bluetooth advertisement message (ADV_IND) differentiated by Exposure Notification in the 31-byte payload section. This data (Figure 4) includes two key items (besides unique identifiers) Service Data Part (24-bytes): 16-bytes of Rolling Proximity Identifier (RPI) and 4-bytes of Associated Encrypted Metadata (AEM).
The Rolling Proximity Identifier (RPI) is derived from TEK and recycled on the average every 15 minutes. This is the identifier that is used in finding the proximity match between two users or subjects. The AEM includes an encrypted piece of vendor data to compute the proximity distance more accurately when required. The values are coded using Type-Length-Value (TLV) that is native to Bluetooth protocol. The Exposure Notification message is given a unique 16-bit Service UUID of 0xFD6F to differentiate it from other messages. The same ID (0xFD6F) is used for Tagging the Service Data of RPI and AEM. The Advertiser address in the header part of Exposure Notification is configured to be anonymous and the RPIs are recycled every time the MAC address is recycled. The Exposure Notification message broadcast period is recommended to be in the range 200-270 milliseconds.

3.2 Exposure Notification Broadcast Sequence

The Figure 5 illustrates an abbreviated Exposure Notification broadcast sequence. The AppLocal, BLE Local, and Crypto Local on the left side of dotted vertical line represent smartphone subsystems (App) that is sending Exposure Notification messages. The Crypto Remote and BLE Remote on the right side of dotted vertical line represent smartphone subsystems (App) that is receiving (scanning) Exposure Notification messages.

When an App (built with EN) is installed, it takes runtime permission for Exposure Notification from the user. Then the first TEK, 16-byte value, is generated using Cryptographic Random Number Generator (CRNG) on the phone. A DiscretizedUnix Time is associated with this TEK. This random number and Unix Time are then used to derive Rotating Proximity Identifier (RPI). The RPI is the anonymous time dependent value that identifies subject’s proximity in contact tracing. The RPI and AEM are sent over in Exposure Notification messages for about 15 minutes and then a new RPI is generated and used for about next 15 minutes and so on. This is repeated for 24 hours, and then a new TEK is generated and associated to a new i and the EN broadcast continues with new TEK for the next 24 hours and so on. Figure 6 illustrates the relationship among TEK, RPI, AEM, and EN payload. HKDF is a simple key derivation function (KDF) based on a hash-based message authentication code (HMAC). Unix Time is discretized by dividing the 32-bit value by 600 (10 minutes), to convert the unit from seconds to steps of 10 minutes.
3.3 ENScanning Sequence and Contact Tracing

The Figure 7 captures an abbreviated Exposure Notification scanning sequence. The BLE Local, and Crypto Local on the left side represent smartphone subsystems that send Exposure Notifications. The RemoteApp, BLE Remote, on the right side represent remote smartphone subsystems that receive (scanning) Exposure Notification messages. When a broadcast message from another node is received, the payload is stored locally along with its arrival time (Unix Timestamp) and RSSI value. The RSSI stands for Received Signal Strength Indicator; RSSI is used along with the AEM to order the proximity data for assessment when needed. Every scanned message is stored to compute contact duration. The data in received messages (in micro SD) cannot be mapped to its source (smartphone).

The Exposure Notification maintains privacy by using RPI as proximity identifier; RPI is computed with a Random number (TEK) and rotated about every 15 minutes to eliminate wireless tracking. The TEK itself is recycled every 24 hours. TEK is stored in individual’s phone. This is a brilliant design when it comes to privacy. The RIP can only be mapped to a smartphone using that phone!

RPI is a function of both TEK and Discretized Unix time. The RPI is not bound to any hardware address or fixed logical address on the smartphone. Every phone sends periodic broadcast of RPI and encrypted vendor data (AEM) in EN. Exposure Notifications are sent with random source MAC. An RPI value changes exactly when the randomized MAC value changes. Every smartphone in the neighborhood scans for Exposure Notification and stores the scanned record micro SD. The App does not send, record, or share any location information. The information recorded (RPI, AEM, TS, RSSI) can only provide proximity and contact duration between two RPIs.

If a user is found to be infected then with this user’s permission, the list of diagnostic keys (TEK, i) are uploaded to cloud. Besides diagnostic key, no other information from the phone is uploaded. The uploading is done by the user with health authority provided Pin. There is nothing in the uploaded diagnostic key(s) that identifies the user. App downloads the list of diagnostic keys of all the infected users (for the day) from the cloud at a suitable time of the day. The keys are then used to generate RPIs and check against the proximity data in their respective storages. This is done in individual’s phone. If match is found, the data is assessed for its physical proximity and contact duration. A recommendation is made to the user. If there is no match, then no action is suggested.
3.4 Privacy Requirement with Exposure Notification

The scanned data in the Micro SD is safe and unusable in case the smartphone gets into the wrong hand. User’s consent or refusal for sharing the diagnostic keys is also protected.

3.5 Security & Other Issues

It is possible for two different phones to generate the same TEK (16-bytes) on the same day. If this happens on different days, then there is no issue, otherwise this could lead to false positive when only one of the two subjects are infected. Segmenting data based on geographical areas could mitigate this issue. A complete analysis of this issue leading to better deployment is one of the suggested areas of study.

The other likely issue is replay attack. A hacker can capture and replay an EN in the same proximity or elsewhere. Replaying in the same neighborhood while the same original EN is still live is not an issue. If it is played after the original EN expires, then it could lead to incorrect, elongated contact period. The replay of EN elsewhere could again lead to false positive conditions.

There is no authentication to EN, this poses a series vulnerability problem; malicious Apps or Applications can broadcast tailored EN or replay captured EN. This could lead to false positive and shortage of computing resources. Solutions are needed to address all these problems.

3.6 Exposure Notification - Performance & Performance Issues

Once installed, an App can work on smartphone with no support from any outside services. This is different from other designs where there is dependency to a centralized Internet Server(s).

- An App generates one TEK and approximately 96 RPIs each day. If an App sends one EN every 500 milliseconds, then 1800 ENs are sent with each RPI (over a period of approximately 15 minutes; on 3 advertisement channels 37, 38, and 39). The optimal broadcast frequency could be evaluated using analytical and simulation models to optimize power and other factors.
- An App uploads approximately 20 <TEK, i> (~400 bytes) to a cloud server when the subject is infected. This upload is done once; no issue here.
- If 1000 cases are uploaded to cloud on a day, then each App will be pulling 40,000 bytes of data (diagnostic keys) from the cloud. If there are 1 million App users, then this data adds up to 40 GB. The cloud server
needs to be engineered to handle this volume. There is a possibility that all the Apps are likely pulling this data in the same 1-hour-window around midnight (while the user is sleeping). Server needs to be engineered to support this large distribution of data during a short window.

- Each App will receive 1000x20 diagnostic keys for a day for matching proximity data (based on the above example). The last 20 days of scanned data need to be matched against these 1000 diagnostic key sets (each with 20 keys). Each diagnostic key generates 6 RPIs (not all are used in EN) for an hour. Each of these RPIs need to be matched against data scanned over approximately 20 minutes. Assuming there were 10 other phones that were in the same proximity during those 20 minutes, the number of records to be matched is 10x60x20x5. This is assuming every phone does 5 broadcasts per second. The total number of matches to be done assuming 10 users in every interval is 10x60x20x5x6x24x20x1000. This value is in $10^{11}$ range! This area requires optimization considering this sort of computation is beyond the capacity of a typical smartphone.

- The data scanned includes RPI (16-byte), AEM (4-byte), TS (4-byte), and RSSI (1-byte). Thus, each EN record requires 25 bytes without counting the storage (indexing) overhead. Assuming there are 10 other phones on the average in the scanning proximity during each hour, storage space needed is 10x25 bytes. Each phone sends 5 ENs per second, or in other words, 15x60x5 ENs in each MAC roll-over period of 15 minutes and 4x24x15x60x5 ENs per day. Storage space needed per day is 10x25x4x24x15x60x5 bytes. This is approximately 100 Million bytes/day and 1.5 G for 15 days. Assuming 16 GB micro SD, the storage volume needed by the App is close to 10% of the capacity of the micro SD. This area requires investigation to optimize storage without compromising the contact tracing requirement.

This completes the review of the first initiative, we will move on to the second one from Ministry of Health, Singapore Government.

IV. TRACETOGETHER

TraceTogether is one of the first contact tracing initiatives deployed country-wide [8]. It was developed by Singapore Government’s Ministry of Health to augment manual contact tracing. It includes BlueTrace, the privacy preserving BLE protocol, Apps for Android and iOS, and a reference implementation, OpenTrace.

BlueTrace uses Bluetooth LE technology present in smartphone to collect proximity information of other smartphones. Later, this information is used to identify group of people who need to be tested for containment. The Bluetooth technology is common to all contact tracing initiatives. However, there are key differences in how it is made use of. In the coming sections, we will review the operational aspects of BlueTrace and evaluate it against Exposure Notification initiative of Apple-Google.

4.1 BlueTrace Operation and Message Format

The BlueTrace operation starts with the user downloading and installing the App on his/her smartphone (Figure 8). User completes the installation by registering the phone number with the TraceTogether back-end server over Internet. The user is provided with a unique, randomized UserID and it is associated with the user’s phone number. All such bindings are stored in a back-end server. The phone number is used to contact the user when needed.

![Figure 8. User Registration – App Installation](image)

Once the installation is completed, the App starts logging encounters with the same App running on other phones. The information logged includes a Temporary ID (TempID) that is generated with the UserID (Figure 9). Each
TempID comprises UserID, its creation time, and expiry time. It is encrypted symmetrically with AES-256-GCM and then Base64 encoded. The health authority holds the secret key to encrypt and decrypt TempIDs. The TempID also includes two encryption parameters: The Initialization Vector (IV) and an Authentication Tag (for integrity checks). The TempID’s lifetime is 15 minutes, and the expiry time in it is used to prevent replay attacks. Smartphones periodically pull TempID from the backend server. They can request multiple TempIDs in each pull to have enough TempIDs during Internet outages.

Two smartphones when they come in the proximity of each other can discover, connect, and exchange their TempIDs as shown in Figure 10. One of the smartphones takes the role of Master, reads the TempID from the slave and writes its TempID to the slave after connecting. This exchange takes place with Encounter Message. Duplicate connection is avoided by keeping a cache of blocked devices with which connection is completed. Each Encounter Message received from the other phone is stored in the local phone along with the RSSI, Timestamp, and the BLE device model of the other phone.

4.2 Contact Tracing

When a subject is infected, the health authorities with the consent of the subject upload the Encounter Message log from the subject’s phone to the backend server.
To protect users and the system from fraudulent uploads, an authorization code is provided by the health authority and entered through the App to obtain a valid token to transmit the logs. The encrypted TempID can be deciphered using the Single key that the health authorities hold. Once TempIDs and validity periods of TempIDs are known, then the corresponding user ID & phone number can be made use of, if needed.

4.3 Privacy

- The plainUserID is never sent out from the Server.
- The TempID includes UserID, which is encrypted by the server with an authentication and expiry period. If maliciously created TempID ever reaches the server, it can be sorted out for illegal UserID with the key and the authenticator.
- TempID is recycled every 15 minutes, thus eliminating wireless tracking.
- The Encounter Handshake time and the TS in TempID are used to reduce the size of Replay attack window

4.4 Privacy Issues

- The same server holds all related key items: UserID, Phone Number, and the single Key used in creating TempID. If a hacker breaks into this server, the entire system is exposed.
- A sustained DOS attack on the server can create an opportunity window to perpetrate other attacks. For example, the Man-in-the-Middle attack is possible. The batch delivery option of TempID magnifies this issue and makes wireless tracking a possibility.
- Creating Bluetooth connection to other phones, opens out the possibility for exchanging other undesirable information like location information. We can trust the official App. However, a malicious App can impersonate the official App by thus making the latter vulnerable. Hardwired MAC address of the BLE of other phones is known to the App and other phones in the proximity with Connectable directed advertising. This could be exploited for wireless tracking.
- Malicious Apps that get legitimate copies of TempID could create False Positive conditions. There is nothing to validate whether the handshake is happening with the legitimate App or malicious App.

4.5 Other Issues

The App is available on both Android and iOS. The iOS App can use proprietary mode to advertise its presence when it is in the background. This is an issue for non-iOS Apps to locate and connect to this App when it is in background.

4.6 Performance

The backend server sends 80+ bytes of TempID to every user every 15 minutes. This is manageable load for a server even with a population of 5 million. If this is implemented with a UDP Request-Response, the number of bytes exchanged is around 250 bytes per App. Nearly 1 Gb data over a period of 15 minutes. Current servers and broadband networks can handle this comfortably.

In crowded malls and events, the App needs to establish connection and exchange characteristics with a large number of mobile phones. This could become a scalability issue. Vital data collection could be missed in such a scenario. Analytical and simulation models are required to prove the robustness of the design. This needs to be proven before the App is made ready for deployment for normal (non-circuit-breaker) days.

One encounter data is stored per smartphone with enough data to compute the proximity value. There is no room to compute the contact duration if the blocked list is updated after the first encounter. If it is not, then this opens out blocked list management problems and multiple connections to the same smartphone problem (scalability & BLE power issue). The number of encounter messages to manage in a phone is a linear function of the number of people
encountered. This is a manageable number assuming the BLE connections with all those phones are successful. Assuming 1000 subjects are infected per day where each infected person has encountered 100 subjects then the number of people to contact is Approximately 1000x100. This is a manageable load for the server.

V. AROGYA SETU

There are several Apps currently in use all over the world[15]. Arogya Setu is an Indian APP used in contact tracing. Arogya Setu is like the Singapore initiative – a centralized server distributes ID used in proximity computation. However, unlike Singapore initiative, this 8-digit hexadecimal ID is static with no encryption and no authentication. The Bluetooth advertisement is used to scan for proximity information. This design aspect is like Apple-Google EN initiative. It is the weakest in terms of proximity ID due to its size and static nature.

VI. CONCLUSION

This paper is topical. We primarily reviewed two recent contact tracing initiatives. These initiatives use Bluetooth LE present in smartphone providing user privacy. The operational procedures of these initiatives are described and then they are reviewed for their privacy and performance. The Apple-Google EN initiative is decentralized as opposed to TraceTogether initiative. The Apple-Google EN initiative defines a new payload for an existing Bluetooth advertisement message for proximity data gathering. The proximity data is gathered from these advertisements non-intrusively. The number of advertisements gathered creates scalability problem for storage and processing. The Singapore Ministry of Health initiative defines new Bluetooth handshake mechanism to exchange proximity information among phones. This mechanism collects and stores a limited number of proximity records. However, it requires every phone to establish a connection to every other phone in the proximity to complete the handshake. There are other areas of concern with both initiatives based on the information on the public domain. We will be pursuing these concerns in our research and development efforts. The cellular initiatives are not discussed due to their privacy limitation, GPS initiatives are not discussed due to their proximity inaccuracy. This analysis helped us build a design based on the best practices in contact tracing[16].

REFERENCES