International Journal of Latest Trends in Engineering and Technology
Vol.(16)Issue(1), pp.044-050
DOI: http://dx.doi.org/10.21172/1.163.06
e-ISSN:2278-621X

AROGYA ARAN - A CONTACT TRACING INITIATIVEUSING CURRENTBEST PRACTICES

Hari T.S. Narayanan[1]

Abstract- Bluetooth LE (BLE) is becoming a key component of *Contact Tracing* initiatives. There are multiple*contact tracing* solutions that are built with smartphone's BLE. The design choices of these solutions differ, and thus have different privacy, security, and performance issues to address. In this paper we propose a contact tracing design, *Arogya Aran*, that combines the best practices of prevailing solutions with some additional features to create a better overall design.

Key Words: Contact Tracing, Best Practices, Bluetooth LE, Exposure Notification, Encounter Message, Proximity, Inter-working

I. INTRODUCTION

There are many *Contact Tracing*initiatives [1-5], most of them havingthe common operational viewas shown in Figure 1. The term *proximity* mentioned here includes two components in general [6] – the distance between the two phones and *continuouscontact duration* within the mutual BLE range. We have not evolved to deal with interworking Apps from different initiatives yet. Thus, the scope of this paper is restricted to Apps supported by a single *contact tracing*system.

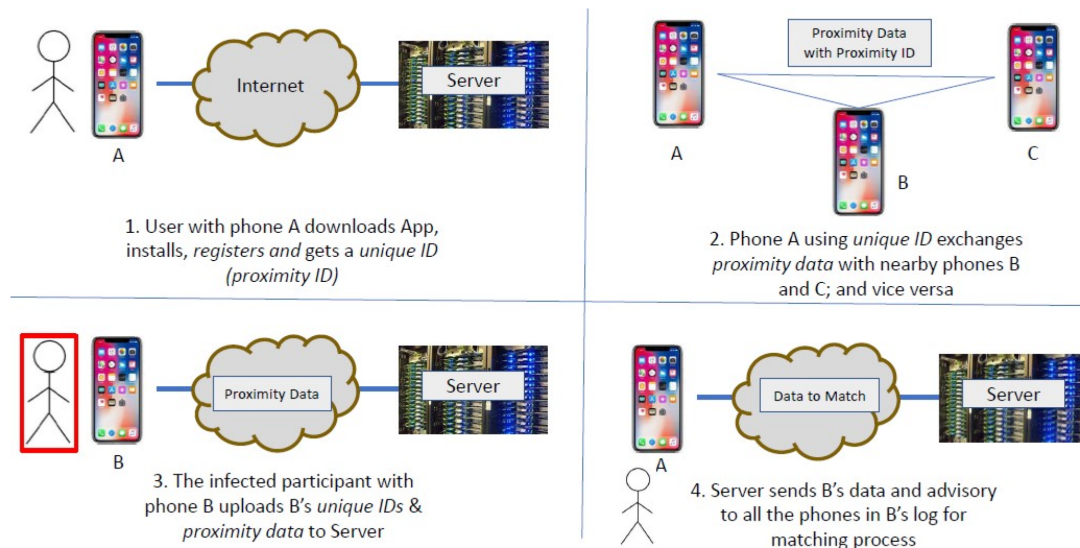

Figure 1. Operational View of Contact Tracing

1. User A downloads the App from the server suggested by the *Healthcare Institution*; installs the App on his/her smartphone and starts the App. App requests permissions for smartphoneresources from the user.

[1] *NetToolsConsulting, Chennai, TN, INDIA*

Once the permissions are granted, then optionally, the App collects the data to build the contact tracing profile of the user. This completes the *Registration* of the user with the contact tracing system.

2. The App using a *special ID,* that is unique to the user/smartphone starts exchanging the *proximity*datawith the smartphones that are presentin its BLE range. This ID is referred to as *Proximity ID*henceforth in the sequel. The *Proximity ID* identifies the user of the App*anonymously* and *indirectly*. A recycled, rotating value is preferred for*Proximity ID*. The exchange of proximity data is enabled by the *Bluetooth LE adapters*in smartphones. This exchange is expected to happen throughout the active period of the day. There are Apps that collect proximity data using *GPS* and *Cellular technologies*, that are outside the scope of this paper
3. If one of the subjects (B in Figure 1) participating in contact tracing is infected, then the proximity data in this subject's phone is uploaded to a server with anonymity. This anonymized data and similar data collected from other infected subjects are hosted by the server. This data is periodically used to find the potential subjects to be tested for infection.
4. Once a day, the data hosted on the server is matched against the data collected bythe smartphones of the non-infected subjects in the participating population. This process checks to see if the data of non-infected population have *proximity* data of any of the infected subjects. The proximity data weighs in on two factors in general – *contact distance* and *contact duration*. A decision to isolate a subject for testing is made based on these two factors.

Based on the results of the matching process, a decision to test or not to test is taken. Almost all the contact tracing applications seem to support this operational view. The lack of literature available in the public domain makes it difficult to confirm this as the only operational view. The design practices or design optionsused in these initiatives are not identical. Some have chosen options to ease the implementation and thus compromising user privacy [5] and some have taken options that offerstronger user privacy [7]. Section 2 below describes the design options. Section 3 compares the merits of design options, and Section 4 combines the best available practices to create our proposed design, *Arogya Aran*.

II. CONTACT TRACING DESIGN OPTIONS OR PRACTICES

Using the operational description that we did in the previous section we can identify the following design choices. We are not claiming that this list is exhaustive; we are simply stating that these choicesare seen among the initiatives in current use.

1. One of the simple design choices is related to the characteristics of the*Proximity ID*. The ID could be a fixed value; or it could be a rolling value. If it is a fixed ID, then it is bound to the App when the App is downloaded for installation. If it is not fixed, then it leads toourseconddesign choice.
2. If the *Proximity ID* is not fixed, then its generation and rotation needto be managed either locally in the smartphone or managed by a server over the Internet. These Identifies are generally stored wherever they are generated.
3. The third option is to exchange the *proximity data* with other smartphones with a minimal change to *BLE*, without requiring changes to current API [11] and protocol[8]. We see the following choices among current designs:
   A. Using the BLE as it is, with no change.
   B. Changing the *Payload* of an existing BLE message with no new messages.
   C. New *Service profile* exchange using an existing handshake and messages.

   None of these options needs changes to protocol or API, but is restricted to change in payload mostly. The third option requires a phone to setup a connection to another phoneto exchange the proximity data. The first two options are *non-intrusive*; that is no connection is needed. The data is exchanged using the existing advertisement message ADV_IND (*Connectable undirected advertising*). Thus, we can classify these design options to be *intrusive* or *non-intrusive*. The design is intrusive if BLE connection is used in the exchange of proximity data, otherwise *non-intrusive*.The design choice made here has got a bearing to the next choice to be mentioned below. The current advertisement messagesare not authenticated unlike the data exchanged with the third option. The authentication referred to here is scoped to *messageintegrity* alone and not the *source integrity*.
4. Any of the three exchanges that we listedso far as above could be authenticated or not authenticated, whichgives rise to our next design option. That is, proximity message can be exchanged with encryption,

[1] *NetToolsConsulting, Chennai, TN, INDIA*

message authentication, andsource authentication or with a subset of these features. The choice made here is sometimes restricted by the choice made in option 3 above.

5. The final design option is whether matching the proximity data is done locally in smart phone or at a cloud server hosted by the health authority. This design choice is constrained by the decision made in option 2 above. The matching process needs to happen either in the server or at the smartphone, depending on wherethe identifiers are generated, either the former or latter.
6. There are a number of optimization options that are available across all the designs. For example, the proximity activity can be shut down while the user is sleeping at his/her residence. These options are beyond the scope of this paper and reserved for its sequel paper.

In this section we simply enumerated the design options. The next section compares these options for their merits and demerits in the overall design.

III. MERITS OF DESIGN OPTIONS

Fixed or Rolling Proximity Identifiers – The fixed proximity identifier used in [5] leads to a simple design but it is vulnerable for abuse. A smartphone with fixed ID could be wireless tracked. Rolling ID [1,4] provides anonymity to the user at the cost of system complexity. In almost all the countries, the privacy is considered a key requirement thus making *rolling ID* mandatory.

The presence or absence of centralized server in generating Proximity ID: The presence of centralized server to generate and distribute *proximity ID*[1]includes some merits. This server maintains the association between*user* and *proximity ID* securely. The App periodically gets a new ID either by *Pull or Push* mechanism, and with or without acknowledgement. The use of Apps could be monitored, and analytics could be developed. The centralized server can also be used for other heavy-dutytasks like proximity matching. However,there are some issues with the centralized server option viz., the server requires a mechanism to send *ProximityIDs* in encrypted form with authentication. The Apps can exchange this *encrypted-authenticated Proxy ID*s as they are received. This requires the server to manage potentially millions of keys.Using a single key is likelyto simplify the problem but at a cost. Either way, the server becomes the single source of all the data. If a hacker breaks into this server, the entire data could leak. The other issues are specific to servers – *DOS* and *man-in-the-middle* attack, etc. Generating the *Proximity ID* in a smartphone could help us to create a decentralized solution [2-4]. The current smartphones have enough resources to manage the generation and rotation of their own Identifiers. This design choice eliminates the issues seen with the centralized serverviz., every smartphone manages its own *key*and *identifiers*.When a hacker breaks into a smartphone, only the corresponding data of the phone is compromised. There is no need for network transport for *Proximity ID*, thus nullifying the attacks listed earlier.

Non-intrusive or Intrusive interaction among BLEs[8-10]: A *non-intrusive*option makes use of Bluetooth advertisement (ADV_IND) to share proximity information. This is done with a new payload type, or with the existing payload type. A single advertisement message can reach any number of other phones in the broadcast range. This is different from the *intrusive* solution where a phone establishes Bluetooth connection to all its peer devices to exchange proximity data [1]. This latter solution does not scale. The connection can be abused to get other data. For example,abusing of the adapter's address for wireless tracking is one. In a crowded malls and events, the App needs to establish connection and exchange characteristics (aka *encounter data*) with many mobile phones. This could become a scalability issue. Critical data collection could be missed out in such scenarios.

The *encounter data*[1] is stored for every smartphone that appeared in BLE adapter's range.Thisdata is enough to compute the *contact distance*. Once this data is available for a phone, then it is added to the list of *blocked phones* to avoid reconnection [1]. Due to this there is no data to compute the *contact duration* if the blocked list is updated after the first encounter. If it is not, then this opens up*blocked list* management problem and multiple connections to the same smartphone problem (scalability issue). The number of *encounter messages* to manage in a phone is a linear function of the number of phones encountered. There are other problems like *replay-attack* that are common to all the three options.

Messages with or without Authentication: The proximity data is exchanged with message authentication in [1]. This makes it difficult for *rogue App* and Applications to mimic this exchange. There is no support for message authentication in two non-intrusive solutions [2-5].The receiving Apps will not know if the proximity data contained in BLE ADV_IND [8-10]is legitimate or not, thus making them vulnerable.This could completely ground the

phones with large volume of tailored proximity data. One can find the rogue App or Application in controlled environment, but this cannot be guaranteed always.

Distributed or centralized Proximity Computation – Proximity data is used to find if there is a match that requires further action. For instance, if subject had contact with a new patient then the subject need to go for testing. This matching computation can be done either in a centralized server [1] or in individual smartphones [2-4]. The distributed solutions empower the subject and a centralized solutions leave the results with competent authority. Apart from this there are technical issues with both. There is no dearth of computing resources with server option. The smartphones have relatively less resources (CPU, storage, and memory) at their disposal. Here is a simplified resource requirement calculation for smartphone option

- Each proximity data received is about 40-bytes. Let's assume that there are 10 other phones on the average in the scanning distance for entire day. Each phone sends 5 proximity data record per second, or in other words, 15x60x5 records in 15 minutes and 4x24x15x60x5 data records per day. Storage space needed per day is 10x40x4x24x15x60x5 bytes. This is approximately 170 Million bytes/day and 1.7 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.

Let's assume that there are 1000 new victims per day. Each victim's 20 days data includes – the day and the rolling ID for the day. The combined size of the rolling ID and the day is 30-bytes. This combination of Rolling ID-day is identified as*diagnostic key*henceforth. Each victim's *diagnostic keys* for the last 20 days are sent to smartphones for matching [2-4].

- Each phone 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 Rolling Proximity Identifier (RPI) 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 against an RPI is10x60x20x5. This is assuming every phone sends 5 proximity data per second. The number of RPIs are 1000x20x24x6. The total number of matches to be done assuming 10 users in every interval is 10x60x20x5x6x24x20x1000. This value is in 10^{11} range! Some of these matches are complicated. They need to weigh distance and duration combination before choosing a match.

Fortunately, there are ways to reduce the computation and storage requirements. This will be discussed in an independent future work with solution specific optimization. The result of match is finally notified to the user.The user is empowered to take an action if there are matches,otherwise, there is nothing to notify to the user.

These design choices offer certain advantage and certain challenges. For instance, *non-intrusive* interaction offers a scalable solution, however, it is inherently weak on authentication. A solution that is decentralized and non-intrusive offers number of advantages. Enhancing that with rolling identifiers that are authenticated to mitigate replay and other attacks is a recipe for a good solution. That is what is proposed in this paper. The main issue with the current non-intrusive solution is that it lacks authentication. We propose a simple authentication feature to enhance this design for a better solution. This authentication is simple and requires a common service to manage and distribute the key. It discourages the development of rogue App and Applications. The contact tracing Apps can be enhanced to quickly find these rogue applications with this authentication in place.

IV. CONTACT TRACING WITH THE BEST PRACTICES

In this section, we first describe the current design option, *Exposure Notification* (EN), from *Apple-Google*. Then we suggest an enhancement to it.

The BLE is Personal Area Network (PAN) technology that operates mostly in *Master-Slave* mode. A slave device with BLE starts advertising its presence when turned on. This advertisement message, ADV_IND (Figure 2),can be used to find and compute proximity (distance)and *contact duration*in *non-intrusive* manner. That is, without establishing a connection to the device. This offers a simple, scalable, *safer*(in some sense) solution for proximity problems. The solution requires 3 to 4 advertisements per second and it could be reduced to a smaller number ofadvertisements under certain conditions. There is a change that is made to BLE to support this – a new payload type is added to the existing ADV_INDshown in Figure 2 [2-4]. This new advertisement message is

[1] *NetToolsConsulting, Chennai, TN, INDIA*

referred to as *Exposure Notification* from here on. This requires no change to BLE API or protocol. EN includes two key items in *Service Data Part* (24-bytes) identified by its unique payload ID (0xFD6F0): 16-bytes of *Rolling Proximity Identifier* (RPI) and 4-bytes of *Associated Encrypted Metadata* (*AEM*) [2-4].

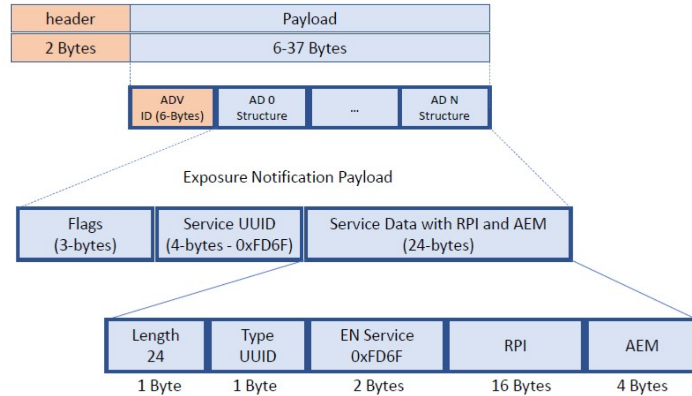

Figure 2. Payload Modified BLE ADV-IND Advertisement

The *Rolling Proximity Identifier* (RPI) is used in finding the proximity match between two users or subjects. The *Rolling Proximity Identifier* (*RPI*) is derived from *Temporary Exposure Key* (TEK). The TEK is a random number, generated within the phone and recycled every 24 hours. The RPI is recycled on the *average* every 15 minutes. The *AEM* includes an encrypted piece of BLE adapter'svendor 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 ADV_IND messages. The same ID (*0xFD6F*) [2-4, 10] is used for Tagging the *Service Data* of RPI and AEM. The MAC(ADV) address in the header part of *Exposure Notification* is configured to be random and RPI is recycled whenever the MAC address is recycled.

The *proximity* is tracked with RPI and the *contact distance* is computed with better accuracy using the *vendor ID*. There are no other fields in the payload besides these two. This design leads to an issue– any App or Application can easily mimic *EN* with the tailored RPI and vendor ID. In the current design there is no way to identify the legitimacy of an advertisement. A rogue application or an App can flood the system with these tailored advertisements and make the Micro SD on the phone to run out of capacity. This is a serious problem that we address in the rest of the Section.

We propose a simple*authentication* featureto address this problem. We need two bytes for *messageauthentication* and optionally two more bytes for tracking the life of *EN*. There are no bytes left to add these features in the current encoding. There are some options that can be explored based on the requirement. One option is to take two bytes from RPI and use it for message digest or take four bytes from RPI and use two of those bytes for digest and two bytes for tracking the lifetime. Decreasing the size of *RPI* increases the *false positive* probability. There is another option, in this option we treat the entire payload of service *UUID* as a single blob of data from protocol perspective (like it is done with *TempID* in [1]). The structure is visible only to App and related processing. This does not require Type and Length encoding. It is implied by the Service UUID chosen. The field sizes and their respective positions are fixed within the structure. This means there is no need for TLV after the Service UUID of 0Xabcd. This option gives 16-bytes for RPI, 4-bytes for AEM, 2-bytes for lifetime, and 2-bytes for digest. If we choose one of those options two accommodate both *digest* and *lifetime*, here is how we build EN with them. We prefer the blob option rather than stealing bytes from RIP.

The Advertising BLE populates the digest using a key, and rest of the data. The scanning BLE validates it for authentication with the same key, and rest of the data. If the *computed digest* matches the received digest, then *EN* is accepted, otherwise rejected.There is a single *authentication key* that all the App instances share.An App can request for this key securely from a network server. A server in the cloud serves this key and rotates it periodically at discrete Unix Time[12]; perhaps at every multiple of 60minutes. An App uses its last *authentication key*and its timestamp (TS) to get the latest key and its timestamp securely. The App gets its first authentication key (and TS) during its installation. The server where App is hosted could serve this key. An App that is inactive for a few days can still use its old key to get the current one. This *authentication mechanism* is simple and discourages rogue Apps

and Application from sending tailored *Exposure Notifications*. Even if they send, it will not incur the same magnitudeof resource usage.The App can have an integrated feature to identify rogue Apps and Applications.

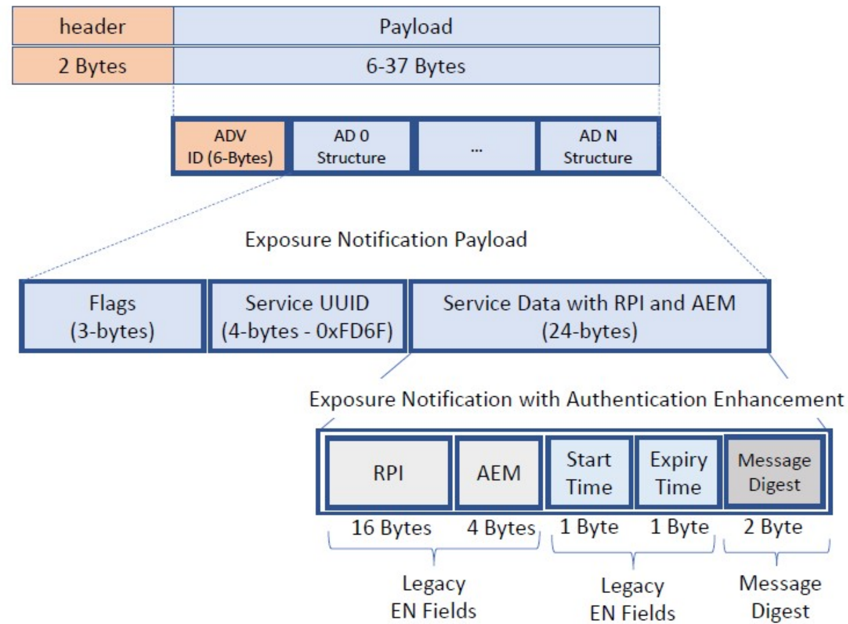

Figure 3. Exposure Notification with Message Authentication and Message Expiry Time

The 2-bytes lifetime code *start* and *expiry* time as Discrete Unix Time offset. The *lifetime*and*message digest*are used to reduce the replay attacks. The Figure 3 describes the new payload with all its fields. Both *start* and *expiry* times are specified as 1-byte *discrete Unix time offset* value. The *message digest* covers all the fields including EN *lifetime*.

The following example illustrates the use the *lifetime* and *message digest* in minimizing the replay attacks. The following values are assumed for this illustration: The *message digest key* changes on every 2nd Unix hour at the server. Every two-hour periodis divided into eight 15 minutes lifetime slots: 0 to 15, 15 to 30, and so on. There is an overlap of time at the interval junctions. The App processing logic includes some tolerance to handle this overlap. The *lifetime* of *Exposure Notification*isencoded into this field.

Here are the possible scenarios when a rogue BLE captures an *EN* with the lifetime encoded as 0 to 15 minutes:

1. The BLE can replay the captured *EN* within the same neighborhood where *EN* originated:
   - *EN* replayed in aperiod that matches 0 to 15: This could possibly result in an*elongated proximity contact time*.
   - *EN* replayed during the 15 to 30 minutes slot:Replay attack will fail because the receiver is looking for *ENs*encoded with 15 to 30 minutes.
   - Replaying during the next 0 to 15 minutes slot: This will fail due to wrong *message digest*.
2. The BLE can replay the captured *EN* within a different neighborhood:
   - *EN* replayed in a period that matches 0 to 15: Possibly a *False Positive Case*.
   - *EN* replayed during the 15 to 30 minutes slot: Replay attack will fail because the receiver is looking for *ENs* encoded with 15 to 30 minutes.
   - Replaying during the next 0 to 15 minutes slot: This will fail due to wrong *message digest*.

Thetwo changes suggested herecreate a new *Contact Tracing* initiative, *Arogya Aran*, which uses non-intrusive BLE exchange with distributed processing for matching *Proximity* data. The*proximity*data is enhanced to include *authentication* and *lifetime*data. It does make use of a centralized server for shared *authentication* key. This service could be hosted on the same cloud server that is hosting the *digest keys or the App*. The major issues seen in *non-intrusive* and decentralized design are addressed with authentication. More constants are added to thedesign, 2-hour

[1] *NetToolsConsulting, Chennai, TN, INDIA*

authenticationkey rotation time, and *lifetime* slot size of 15 minutes. A new constant need to be registered with BLE SIG group for the modified *EN* payload. This solution can form the seed for inter-working Apps once the constants are standardized. Vendors can still differentiate their offerings by innovative optimization of computing resources and added features.

V. CONCLUSION

In this paper we presented the common operational model of BLE based contact tracing initiatives. We identified some of the best practices among these initiatives using this model. A new design is created by combining these best practices with some enhancements of our own. Using our design choices as template different designs could be composed for different requirements. For instance, a centralized scheme can use smaller ID with BLE ADV_IND to scale better and conserve BLE power. Contact tracing is an evolving new area of study. The major task is to create the standard framework that supports interworking Apps. There is plenty of scope for optimization and analytical studies - reducing the number of exposure notifications using human activity model, quantifying the occurrence of *False Positive* events, etc. Apart from these,there are initiative specific enhancements that requires addressing. Since the key size is reduced to 8-bytes, there are higher possibilities for False Positive, we will be reviewing the implications of this in a sequel paper for different population size.

Note: - The phrase "*Arogya Ar*an" is a *Tamil* phrase; stands for *Health Fortress* – word by word translation!

REFERENCES

[1] BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf

[2] Exposure Notification Specification for Contact Tracing – a joint initiative of Apple & Google, May 2020: https://www.App le.com/covid19/contacttracing/

[3] Exposure Notification Bluetooth Specification Preliminary, April 2020 v1.2

[4] Exposure Notification Cryptography Specification Preliminary, April 2020 v1.2

[5] Arogya Setu Official site: https://www.mygov.in/aarogya-setu-app/

[6] Bradley Mitchell, Overview of a Personal Area Network (PAN) PANs and WPANs consist of personal, nearby devices, Lifewire, April 04, 2020, https://www.lifewire.com/definition-of-pan-817889

[7] Hari T.S. Narayanan, Contact Tracing with Bluetooth LE – A Status Review, May 2020 to International Journal of Latest Trends In Engineering And Technology.

[8] "Bluetooth Smart or Version 4.0+ of the Bluetooth specification". *bluetooth.com*. Archived from the original on 10, March 2017.

[9] Bluetooth 5: Go Faster, Go Further, https://www.bluetooth.com/bluetooth-resources/bluetooth-5-go-faster-go-further/

[10] Generic Access Profile of Bluetooth: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/

[11] Android Bluetooth API, Bluetooth advertisements scanning parameters – Comprehensive list: https://developer.android.com/reference/android/bluetooth/le/ScanSettings

[12] Unix Time: https://www.unixtimestamp.com/