International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 05, May-2020

A Study of *False Positive* Incidences in Current *Contact Tracing* Initiatives and Mitigating Solutions

Hari T.S. Narayanan
Net Tools Consulting,

Abstract **– There are several *Contact tracing* initiatives since the outbreak of COVID-19. The data collected during the *contact tracing* process may falsely identify someone as test subject even though the subject never was in the proximity of an infected person. This type of *False Positive* (FP) incidences could drain the resources of an otherwise well-designed system. In this short paper we are investigating the *False Positive* incidences of current *contact tracing* initiatives. We also suggest design alternatives to address related issues, specifically, the FP incidences that occur due to lack of *message authentication*.**

Keywords – Proximity Data, False Positive, contact tracing, Exposure Notification, Arogya Setu, Encounter Message, HMAC, Digest, Truncated Digest, TraceTogether

1. INTRODUCTION

The *proximity data* [1, 2] collected in a *contact tracing* application, generally includes enough details to compute the physical proximity of the contact and the duration of the contact. In all current initiatives [3, 4, 5], the Apps on smartphones using *Bluetooth Low Energy* (BLE) [6] adapter exchange *proximity messages* that enable the computation of *contact distance* and *contact duration*. There are challenges in the collection of this data [7,8], but that is not in the scope of this paper. Due to various reasons the data may incorrectly identify a subject for testing. An understanding of *False Positive* incidence in a large system like *contact tracing* is both socially and fiscally relevant. A good estimate of *False Positive* cases will allow us to protect the system from resource drain by addressing issues ahead of time and save unnecessary trauma to the subjects. This paper analyzes the *False Positive* incidences in some of the recent *contact tracing* initiatives.

The *False Positive* cases are caused by choice of design parameters that are not large enough or lack of authentication to proximity messages or due to *Replay Attack*s executed with genuine proximity messages. We notice that there are two different types of *False Positive* incidences in *contact tracing*. The *weak* one [8] that simply stretches the magnitude of contact duration and, the *strong* one with completely falsified proximity data. We analyze three initiatives here for their *False Positive* incidences. All the three initiatives are based on Bluetooth LE (BLE) and *Unix Time* [9]. We describe just enough BLE here to understand the message structure that is relevant for this paper. The next three sections, Section 2 to Section 4 provide short overviews of Bluetooth LE (BLE), Hashed Message Authentication Code (HMAC) [10], and *Unix Time*, respectively. These overviews are used in the Sections 5 to 7 to understand the *False Positive* incidences in three current contact tracing initiatives. Design alternatives to these initiatives are also suggested to address *False Positive* issues. Section 8 addresses *Replay Attack*, that is common to all the three initiatives.

2. BLE FOR CONTACT TRACING

The BLE is the best suited technology at this juncture for exchanging *proximity data*. Its prevalence and characteristics are best suited for *contact tracing*. The BLE is a *Personal Area Network* (PAN) technology that operates mostly in *Master-Slave* mode. Every BLE device (both slave and master) advertises its presence and scans for the presence of other BLE adapters in the range. A device with BLE starts advertising its presence when turned on. One of the advertisement messages (Figure 1), *ADV_IND* [6], is used in two of the three initiatives to exchange proximity data in a *non-intrusive* manner. That is, exchange takes place without establishing a connection to other BLE devices. The contact list can be computed using the scanned *proximity data*. This offers a simple, scalable, *safer* solution for *proximity data* exchange. There is a change that is made to BLE to support this in Apple & Google initiative [11,12] – a new payload type is added to the existing advertisement message. There are no changes required to BLE API [13] or *BLE Protocol* [6] to accommodate this change.

BLE ADV_IND Advertisement Message

Header	Payload
2-bytes	6-37-bytes

ADV-ID	AD_0 Structure	...	And Structure
6-bytes	31-bytes		

Figure 1. ADV_IND – BLE Advertisement Message Structure

A similar payload level change is used by *Arogya Setu* initiative [4] – an existing payload type (*device name*) is used for proximity data in *ADV_IND*. This modification is an interim solution, and acceptable in smartphones, but not with other BLE devices. The *TraceTogether* initiative [3], however, handles *proximity data* exchange quite differently. A *BLE handshake* is used for exchanging proximity data. Still there are no changes required to *BLE API* or *Protocol*. The *TraceTogether* could not make use of the *ADV_IND* because the *TraceTogether* message size is 84-bytes; this value is much bigger than the permitted payload size (31-bytes) of *ADV_IND*. There are several implications to *TraceTogether* performance due to this design choice – some are positive, and some are not. In this paper we will restrict our discussion to *False Positive* Implications of this design decision.

The Bluetooth 5.0 [14] offers an advertisement message that can accommodate a large (255-byte) payload. Eventually, this BLE 5.0 feature can be used for proximity data with enhanced features. However, at this point, the number of smartphones that are in use, supporting BLE 5.0 is not enough to make use of this feature.

3. HMAC – MESSAGE AUTHENTICATION CODE

As suggested earlier, one of the reasons for *False Positive* incidence is lack of message authentication. An authenticated message could reduce *False Positive* incidences and protect contact tracing applications from malicious Apps and Applications. In this section we will present a standard message authentication method – *Hash Message Authentication Code* (HMAC) [15-17]. In later sections, we will investigate its applicability for the contact tracing initiatives.

The HMAC assumes the presence of a shared key, *K*, between the *sender* and the *receiver* of a message *m*. This key is often rotated. The *sender* using a known hashing function, with key *K* and message *m* as inputs generates a *hash* output (Figure 2). This output is *Message Authentication Code*, *MAC* in short. The MAC is also referred to as *Message Digest*. Both the message (*m*) and the generated *MAC* are sent to the receiver. The receiver using the same hashing function generates a *hash* with shared key (*K*) and the received message (*m'*). If the received message *m'* = *m*, then the generated *hash* is same as the received *MAC*. If the received message or the *MAC* is tampered with, then the *hash* value computed at the receiving side will not match the *MAC* sent.

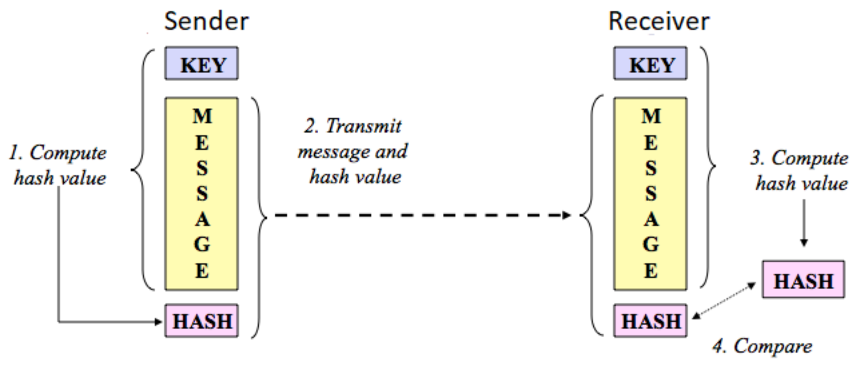

Figure 2. HMAC Operation [13]

The HMAC supports various combinations of key and digest sizes [16,17]. The ones that we are likely interested in this paper are short ones - MD5 HMAC that uses a 16-byte Key and produces 16-byte digest and SHA1 HMAC uses a 20-byte key and produces 20-byte digest. The messages of contact tracing initiatives are disconnected and short lived, thus a short MAC is good enough. We are also interested in exploring *truncated digest* [16] due to lack of space in some cases. Instead of sending the full digest of 16-bytes, we will use truncated digest of 8-bytes, assuming both *sender* and *receiver* have an agreement on this truncation.

4. UNIX EPOCH TIME

The *Unix Time* [10] is used in two of the three 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) [11] 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*.

5. FALSE POSITIVE ANALYSIS OF AROGYA SETU [4]

The *Arogya Setu* is initiative uses a fixed 4-byte value for *Proximity data*, which happens to be the *Proximity ID*. The source code is open. Every App is assigned a unique identifier. This identifier is static; that is, not rotated over time. This makes *Arogya Setu* vulnerable for *Wireless tracking [18]*. This design is simple and free of *False Positive* from the *Proximity ID* perspective. However, malicious Apps and Applications can still cause enough damage due to lack of authentication in *Proximity data*. We suggest the following design alterations to make *Arogya Setu* more robust with respect to the attacks that we just mentioned.

Since, there are no other data besides 4-bytes of Proximity ID, the proximity data can be enhanced to include two additional data items: 1. Digest to validate the authenticity of the data exchanged and 2. Expiry time for proximity message. This change requires an additional server and support for server interaction with App. This is not a complex suggestion, considering its benefits. It will discourage attacks from malicious Apps and Applications.

The App (*Arogya Setu*) can also be added with a feature to detect these malicious programs. The *Replay Attacks* are still possible, even with this enhancement. A malicious App can still capture legitimate Proximity data and use it for *Replay Attacks*. The *Replay Attack* and its implications are common to all the three initiatives assuming all of them support *authenticated messages*. We will discuss the implications of *Replay Attack* in a separate section (Section 8) in this paper.

The diagram Figure 3 shows the encoding of authenticated *Arogya Setu* Proximity message with expiry time. The *BLE ADV_IND* message includes a header of 2-bytes, followed by 6-bytes of *advertiser ID*, the *MAC address* of BLE adapter, optionally randomizable. The rest of the 31-bytes are user payload. The payload type of 0xFF is used to transport vendor specific payload. So, we can use this payload type to exchange proximity data with peer Apps instead of the default data type used for *device name*. The first byte after that is used to identify *Arogya Setu*. It is likely, other Apps may also use 0xFF payload type for proximity. The value in this 1-byte could be a registered constant that identifies *Arogya Setu* uniquely. The ID field of 4-bytes are coded after the App ID field. The *Stop* and *Expiry* time fields are used to encode the life of this message in *Unix Time seconds*. The last field of 16-bytes is for *HMAC-MD5 authentication*. The message digest is computed using all the preceding fields including the header field of 2-bytes. There is an alternative design using *SHA1 HMAC* with 20-byte *Message digest*. This reduces the bytes available for lifetime to 2 bytes – one byte for *start time* and one byte for *expiry time*. Instead of encoding the Unix Time (seconds), now the 2-bytes of lifetime encode *start* and *expiry* time as *Discrete Unix Time offset* (example: minute offset). An example illustrating the use of these fields appear in Section 8 where *Replay Attack* is discussed. The 29-bytes that appear after the *Type* field is type less for protocol level processing, but meaningful to Application. There may be other applications that are likely sending and scanning for vendor specific data with the type field set to 0xFF. If *Arogya Setu* gets one such payload, it will with a high probability reject that payload due to incorrect digest, length, App ID, and time fields. If the other application gets *Arogya Set* payload then it is likely an issue that needs to be worked out.

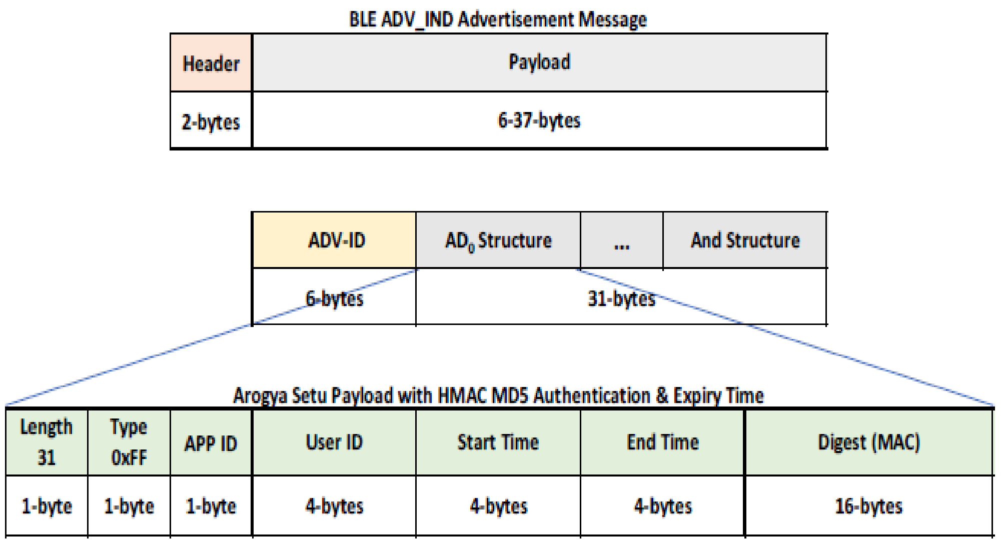

Figure 3. Authenticated *Arogya Setu* Proximity Payload in ADV_IND

We suggest the following key distribution system to complete our message digest design enhancement. 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 the payload 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; perhaps at every multiple of 60 minutes. An App uses its last *authentication key* and its timestamp (TS) to securely get the latest key and the related timestamp (either using PULL or PUSH mechanism). 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 malicious Apps and Application from sending tailored *proximity messages*. Even if they send, it will not incur the same magnitude of resource usage. The App can have an integrated feature to identify rogue Apps and Applications. This design change reduces all the *False Positive* incidences perpetrated by malicious Apps and Applications.

6. FALSE POSITIVE ANALYSIS OF TRACETOGETHER

The *TraceTogether* has well designed protection against *False Positive* incidences. It uses authenticated and *encrypted UserIDs*. This *UserID* is unique, and it is assigned to the user or smartphone during *registration*. The *UserID* is 21-bytes long (Figure 4), it is encrypted and authenticated with 16-byte *digest*, and 16-byte *initialization vector (IV)*.

This encrypted *UserID*, known as *TempID*, is rotated periodically, and sent to registered App. The lifetime of *TempID* is specified by the *start* and *expiry* time. The *start* and *expiry* time are specified using *Unix Time*. Every registered App gets its new *TempID* periodically. The size of *TempID* is too long to use BLE *ADV_IND* message. A *TraceTogether* App instead uses a *BLE handshake* to exchange *TempIDs* with its peers. This poses some problems in the areas of security, scalability, and power that are outside the scope of this paper. Despite all these, *False Positive* incidences could arise due to *Replay Attack*s. In fact, that is the only way the *False Positive* incidence could precipitate *in TraceTogether*. The TraceTogether is an initiative of Singapore Government. The *BlueTrace* is the protocol aspect of this initiative.

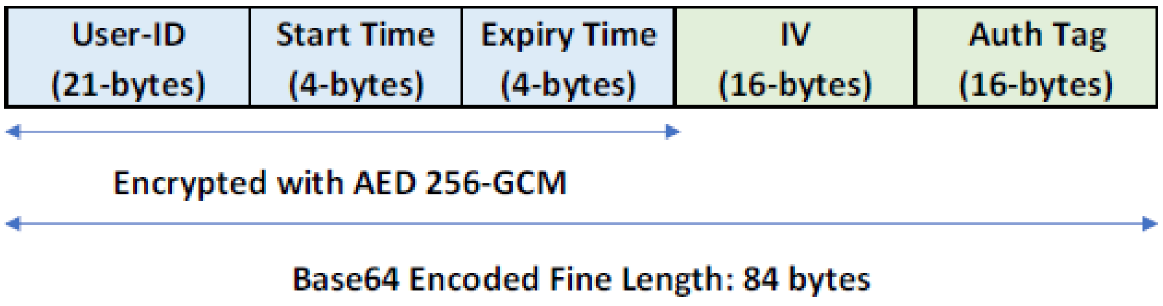

Figure 4. *TraceTogether* Authenticated UserID exchanged in Proximity Data

7. FALSE POSITIVE ANALYSIS OF EXPOSURE NOTIFICATION [5,9,10]

This initiative from *Apple and Google* makes use of the *ADV_IND* payload for proximity data exchange like *Arogya Setu* does. The new payload defined in this initiative, *Exposure Notification (EN)* [11,12], includes only two data items (Figure 5): 16-byte *Rolling Proximity ID* (RPI) and a 4-byte encrypted vendor ID of BLE adapter. The RPI provides one of the best anonymous identifiers in comparison to other two initiatives. The *proximity* is tracked with RPI and the *contact distance* is computed with better accuracy using the BLE vendor data. There are no other fields in the payload besides these two. There is no *message authentication* because there is not enough space in the payload to support it. This means, any App or Application can easily mimic these advertisement messages with the tailored *RPI* and vendor data creating *False Positive* data. There is no provision to identify the legitimacy of such advertisements when they are received by a smartphone. A malicious application or an App can flood the neighborhood with these advertisements and make Micro SD on a smartphone to run out of its capacity. This is a serious problem. When a case is identified as *False Positive*, it is already too late, the damage is done. Finding them at source without the authentication is a challenging task, and almost impossible. This type of attack is a superset of *Replay Attack*. The *Replay Attack* is discussed in Section 8 of the paper.

The 16-byte RPI [11,12] is generated from a 16-byte random number, *Temporary Exposure Key* (TEK). The TEK is recycled once in 24 hours. The RPI is derived as a function of TEK for the 24-hour period and *Discretized Unix Time* that ticks every 10 minutes. This process is completely distributed, in the sense that, every smartphone with the App generates its TEK every 24 hours, using which a new RPI is generated every 10 minutes. This RPI is the identifier exchanged in Proximity message. The *False Positive* is a possibility if two phones generate the same TEK at the same time. This situation can lead to *False Positive* only if one of the two subjects get infected. Otherwise, there is no issue with duplicate TEKs. Using a simple analytical model, we can compute the probability for *False Positive* to convince ourselves that it is indeed a low value for a TEK of 16-bytes.

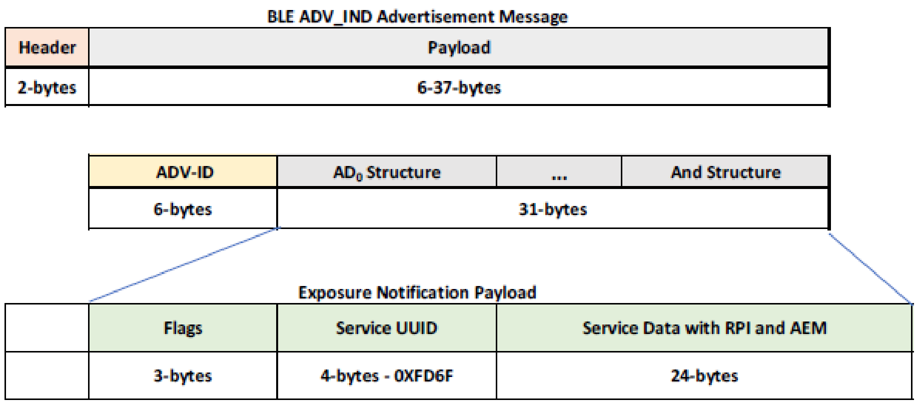

Figure 5. Exposure Notification Payload in ADV_IND

The range of values supported by TEK is 2^{128}. This means, a TEK is drawn randomly from this large population. The number of smartphones in US at this point of time is approximately 256 million. That is, equal to 2^{28}. We are interested in computing the probability of at least two smartphones generating the same TEK during the same 24-hour period. This is a type of *Birthday problem* [19,20], where number of unique days are $n = 2^{128}$ and the number of people $d = 2^{28}$. Since d is much smaller than n, we can apply *Taylor series approximation* to compute this probability as (where "*e*" is natural mathematical constant ≈ 2.7)

$$p(n,d) \approx 1 - e^{-n(n-1)/(2d)}$$

$$\approx 1 - e^{-n^2/(2d)}.$$

The value of this probability is extremely low. There are 26 zeros following the decimal point before a non-zero digit appears. Thus, we can safely conclude the *False Positive* incidences for the chosen value and US population or even with world population is extremely low. The chosen parameter values are large and safe. We can even explore if we can borrow a few bytes from the RPI for a truncated message digest (*HMAC MD5*) to fortify from malicious Apps and Applications.

Unlike *Arogya Setu*, there is not much room here to work on this. The accuracy of *contact distance* cannot be compromised because for large distances (6 feet or more) *contact distance* is error prone without the support of 4-byte vendor data. That leaves us with the only other data field, *Proximity ID* (RPI/TEK) with 16-bytes. The standard digest cannot be smaller than 8-bytes, and even that is a truncated one [16]. Let us try if we can borrow 8-bytes from TEK (RPI). The crypto system needs to be modified for 8-byte TEK and 8-byte RPI. That is a software change, and we need to find the implication of this design change over *False Positive* incidences.

Reducing the size of TEK increases the probability of identical TEKs generated during the same time interval. The duplicate TEKs generated at the same interval could lead to *False Positive* incidences only if one of the TEKs is that of a patient and the other one is not. We can do an estimate of the *False Positive* incidence assuming 8-byte TEK.

There is no change in smartphone population, it is still $n \approx$ 256 Million (Smartphones in US) $\approx 2^{28}$

The size of TEK = 8-bytes (64-bits); possible values for TEK is $d = 2^{64}$

The probability that at least two of the smartphones generating the same TEK on the same day can be computed using *Taylor Series Approximation of Birthday Problem*. Using *Taylor's series approximation* is justified because still (going from 16- bytes to 10-bytes) d is much bigger than n. This probability of 0.002; this is an indication of unacceptable level of *False Positive* incidences for US population size as expected.

$$p(n,d) \approx 1 - e^{-n(n-1)/(2d)}$$

$$\approx 1 - e^{-n^2/(2d)}.$$

This value is close 0.002
That is, 2 duplicates in every 1000
Close to 512000 duplicates every day for the population size of 256 million!!

The one option that is still open for exploration is the encoding of the payload using the vendor specific type 0xFF that we used with *Arogya Setu* in the earlier section. This can be done as follows: 1-byte length field, 1 byte type field that is set to 0xFF, 1-byte unique App ID of EN, 4-bytes of vendor data, 14-bytes of RPI, 2-bytes of message life time (start and expire), and finally 8-bytes of truncated *HMAC-MD5* digest. The truncation can be justified based on the message lifetime and disconnected messages. Rest of the design is like what we described for *Arogya Setu*. The time fields here carry *Discrete Unix Time offset* (in minutes) instead of Unix Time in seconds.

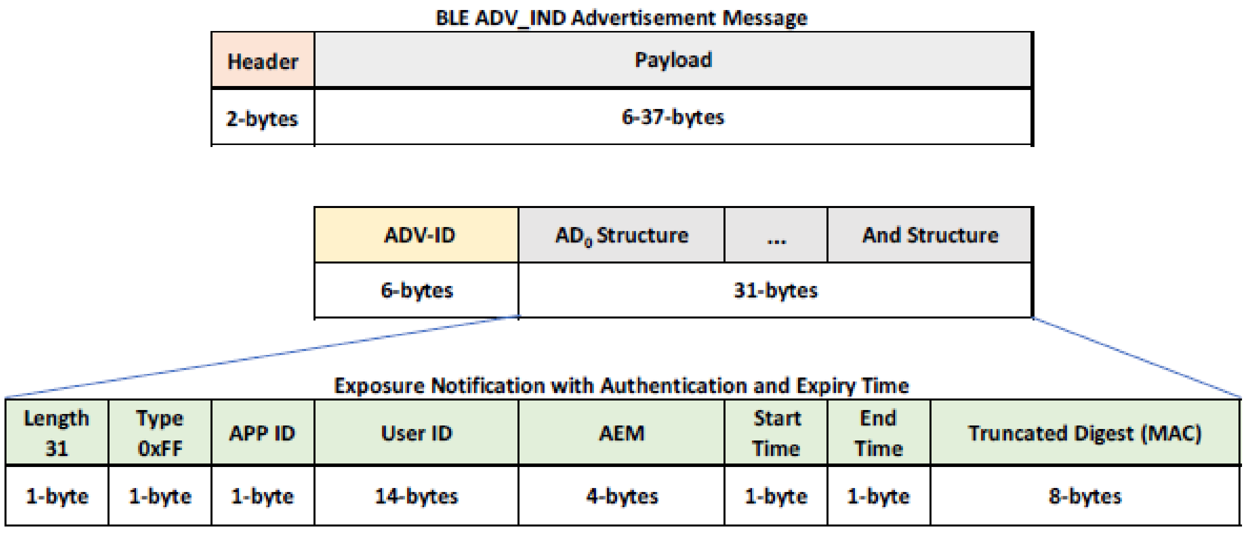

Figure 6. Exposure Notification* with Authentication and Expiry Time

8. FALSE POSITIVE WITH REPLAY ATTACKS

The Replay Attacks are one of the difficult problems to deal with in Information Security. It persists even with Encryption and authentications. In this analysis, we are assuming that all the three initiatives are supported with sufficient authentication and message *expiry time* feature. The following example illustrates how the two features are used in minimizing the *Replay Attack*s. The *digest Key* values are rotated at the message expiry time frequency or vice versa [8].

Here are the possible scenarios when a malicious BLE captures a proximity message and decided to replay this message. Let us assume the lifetime fields are coded with T1 (start) and T2(expire) time respectively. These time offsets are based on a reference time that is synchronized to *digest key* rotation.

1. The BLE can replay the captured *proximity message* within the same neighborhood where *the original Proximity message* originated:
   - *Proximity message* replayed in a period that matches T1 to T2: This could possibly result in an *elongated proximity contact time*. A possible case for *weaker False Positive* incidence.
   - *Proximity message* replayed during any other interval: *Replay Attack* will fail because the receiver is looking for *Proximity messages* within the interval T1 to T2.
   - Replaying during the next cycle of T1 to T2: This will fail due to wrong *message digest*.
2. The BLE can replay the captured *EN* within a different neighborhood:
   - *Proximity message* replayed in a period that matches T1 to T2: Possible *False Positive* incidence
   - *Proximity message* replayed during the any other interval: *Replay Attack* will fail because the receiver is looking for *Proximity messages* encoded with T1 to T2.
   - Replaying during the next cycle of T1 to T2: This will fail due to wrong *message digest*.

9. CONCLUSION

In this paper we investigated the *False Positive* incidences of contact tracing initiatives that are built with BLE. Design modifications are suggested to address *False Positive* issues. These modifications primarily address those *False Positive* incidences that occur due to lack of authentication in proximity messages. The current class of solutions are interim and local in nature. However, many such solutions that are thought of as interim, stayed around longer and are found to be sufficiently good. Feature rich interworking solutions are expected with BLE 5.0 and other technological evolutions.

10. REFERENCES

1. 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
2. Rachel M. Burke, et, al, Active Monitoring of Persons Exposed to Patients with Confirmed COVID-19 — United States, January–February 2020, *Weekly*, March 6, 2020, 69(9);245–246
3. *TraceTogether*: A privacy-preserving protocol for community-driven contact tracing across borders:https://*TraceTogether*.io/static/*TraceTogether*_whitepaper-938063656596c104632def383eb33b3c.pdf
4. *Arogya Setu* Official site: https://www.mygov.in/aarogya-setu-app/
5. Apple & Google, Exposure Notification Specification for Contact Tracing – a joint initiative of Apple & Google, May 2020: https://www.App le.com/covid19/contacttracing/
6. "Bluetooth Smart or Version 4.0+ of the Bluetooth specification". *bluetooth.com.* Archived from the original on 10, March 2017.
7. Hari T.S. Narayanan, Contact Tracing with Bluetooth LE - A Status Review, International Journal of Latest Trends in Engineering and Technology, Volume 15, Issue 5, May 2020.
8. Hari T.S. Narayanan, *Arogya Aran* - A Contact Tracing Initiative using Current Best Practices, International Journal of Latest Trends in Engineering and Technology, Volume 15, Issue 5, May 2020.
9. Unix Time: https://www.unixtimestamp.com/
10. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms, RFC 6151, March 2011.
11. Apple & Google Contact Tracing Initiative, Exposure Notification Bluetooth Specification Preliminary, April 2020 v1.2
12. Apple & Google Contact Tracing Initiative, Exposure Notification Cryptography Specification Preliminary, April 2020 v1.2
13. Android Bluetooth API, Bluetooth advertisements scanning parameters – Comprehensive list: https://developer.android.com/reference/android/bluetooth/le/ScanSettings
14. Bluetooth 5: Go Faster, Go Further, https://www.bluetooth.com/bluetooth-resources/bluetooth-5-go-faster-go-further/
15. *Gary C. Kessler*, An Overview of Cryptography, 26 April 2020, https://www.garykessler.net/library/crypto.html
16. Quynh Dang, NIST Special Publication 800-107 Revision 1 Recommendation for Applications Using Approved Hash Algorithms, Computer Security Division Information Technology Laboratory, Aug 2012. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf
17. The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, July 2008.
18. Johannes K Becker, David Li, and David Starobinski, Tracking Anonymized Bluetooth Devices, Proceedings on Privacy Enhancing Technologies; 2019 (3):50–65
19. Mathis, Frank H. (June 1991). "A Generalized Birthday Problem". *SIAM Review*. **33** (2): 265–270
20. Birthday Problem: https://en.wikipedia.org/wiki/Birthday_problem