Real-world Exploitation and Vulnerability Mitigation of Google/Apple Exposure Notification Contact Tracing

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree
Master of Science in the Graduate School of
The Ohio State University

By

Christopher J. Ellis

Graduate Program in Computer Science and Engineering

The Ohio State University

2021

Thesis Committee:

Anish Arora, Advisor

Zhiqiang Lin

Abstract

Digital contact tracing offers significant promise to help reduce the spread of SARS-CoV-2 and other viruses. Google and Apple joined together to create the Google/Apple Exposure Notification (GAEN) framework to determine encounters with anonymous users later diagnosed COVID-19 positive. However, as GAEN lacks geospatial awareness, it is susceptible to geographically distributed replay attacks. While the replay attack is generally known, we contribute a new proof-of-concept for an easily deployed, anonymous, low-cost, crowd-sourced replay attack network by malicious actors (or far away nation-state attackers) who utilize malicious (or innocent) users' smartphones to capture and replay GAEN advertisements that drastically increase false-positive rates even in areas that otherwise exhibit low positivity rates. In response to this powerful and feasible replay attack, we introduce GAEN$^+$, a solution that enhances GAEN with geospatial awareness while maintaining user privacy, and demonstrate its ability to effectively prevent distributed replay attacks with negligible overhead compared with the original GAEN framework.

Dedicated to anyone with big dreams and the desire to pursue them.

Acknowledgments

There are many individuals to whom I am thankful for their continued inspiration, motivation, and support while pursuing my Master's.

First and foremost, I would like to thank my advisor, Professor Anish Arora, for his mentorship and coaching throughout my research. It is thanks to his interest and encouragement that we were able to begin and complete this journey exploring the nuances of digital contact tracing protocols.

I would also like to thank Professor Zhiqiang Lin for his insightful and valuable feedback. Additionally, thanks to all of my undergraduate professors for sharing their knowledge, allowing me to build a strong base to which I could use to reach higher.

Finally, I'd like to thank my parents, who thought I'd be too wrapped up in the arts to ever want to return to college, much less for engineering, and much less for a graduate degree. Thanks Mom and Dad for never once bounding my ambitions and dreams, always sharing in my excitement, and your never-ending love and support.

Vita

2018 ..... B.S. Computer Science and Engineering, Magna Cum Laude
The Ohio State University

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

Page

List of Tables

List of Figures

Chapter 1: Introduction

1.1 COVID-19 & Digital Contact Tracing

The COVID-19 pandemic continues to evolve and spread nearly two years since its outbreak, claiming over 4.7 million lives globally as of September 2021 [1]. Contact tracing remains as one of the early identified countermeasures to reduce the spread and overall impact of viruses such as SARS-CoV-2. To reduce time and cost intensive resources required by contact tracing solely performed by humans, numerous digital contact tracing (DCT) protocols and smartphone apps have been developed for complementary use [2]. These protocols commonly utilize native smartphone features, such as Bluetooth, WiFi, or GPS, to provide the underlying mechanisms for smartphone apps to determine encounters with other individuals and ultimately notify of potential exposure to positive diagnosed users.

Compared to other wireless technologies, Bluetooth Low Energy (BLE) predominately enables numerous DCT protocols, such as BlueTrace [3], Temporary Contact

Numbers (TCN) Protocol [4], Pan-European Privacy Preserving Proximity Tracing (PEPP-PT) [5], SpreadMeNot [6], PPContactTracing [7], Decentralized Privacy-Preserving Proximity Tracing (DP-3T) [8], and more. BLE is an attractive enabling technology partly because of its relatively low power consumption and its ability to facilitate proximity awareness without using additional sensors or location data. DCT protocols relying solely on BLE are characterized as more decentralized and privacy preserving because they detect proximity between devices to determine encounters as opposed to uploading precise location coordinates to a central server. However, findings from Wen et al [9] reveal numerous protocols leak information, as privacy preservation is simply not inherent to BLE. Thus, the design and implementation is a key factor for privacy preserving, decentralized protocols.

Google and Apple joined forces to create the Google/Apple Exposure Notification (GAEN) framework [10, 11] based heavily on DP-3T, and provided SDKs for approved public health authorities to develop and publish smartphone contact tracing apps. As a result, GAEN-powered apps are widely available, operate more efficiently in the background, and offer a unified protocol to enable communication between the two typically competing, widely adopted platforms with a combined near 99% of the mobile OS global market share [12].

At a high level, GAEN securely generates pseudo-random, ephemeral identifiers and broadcasts them over BLE for nearby participating smartphones to capture. Later, these identifiers are re-derived from a key shared anonymously by a positively

diagnosed user and matched with previously captured identifiers. These mechanisms for encounter determination and significance occurring on users' devices, with a central server only to relay pseudo-random keys, make GAEN a largely decentralized protocol.

While GAEN attempts to balance the tradespace between data-utility and data-privacy to preserve user privacy and increase adoption rates, this approach also introduces weaknesses. Particularly, analysis of GAEN shows it is susceptible to wide-spread replay attacks due to the wireless broadcasting nature of BLE advertisements and the lack of geospatial awareness in the protocol itself. Therefore, a malicious actor can capture a legitimate GAEN advertisement from an honest user and replay it anywhere in the world. If another honest user captures a replayed advertisement originating from a user who is later diagnosed positive for COVID-19, they may receive a false-positive exposure notification even though the two were not in close proximity. Consequently, geographically distributed replay attacks would substantially increase false-positive exposure notifications and bring negative consequences to individuals' and collective society's daily life.

Previous works have specifically addressed the replay vulnerability in GAEN. For instance, SpreadMeNot [6] offers a public-private key distribution solution that diverges significantly from GAEN. Others offer modifications that arguably fit within the GAEN framework.

In particular, Raskar et al present a recommendation for adding global location to GAEN [13], suggesting GPS context to be included as an encrypted payload in broadcasted GAEN advertisements. While this recommendation appears to be a strong candidate against replay attacks, among other issues that we discuss later, it risks decreased adoption from public perception around GPS locations being continuously broadcasted to others, even if encrypted. Another recent work, *ACTGuard* [14] recommends a third-party app and server to ultimately verify two users' identifiers are broadcasted at the same time and location through relaying one-way hashes upon encounter and calculation of encounter to a separate server. However, this solution seems to add unnecessary complexity to an otherwise simple protocol.

1.2 Thesis Contribution

The main contribution of this thesis is twofold. First, we raise public awareness of the impact of replay attacks against GAEN, by demonstrating concretely such an attack in a low-cost, easily deployed, scalable, and practical way (§3). While replay attacks are known, we introduce a geographically distributed, crowd-sourced replay attack network Proof-of-Concept (PoC) that offers citizen malicious actors a low-barrier to participate and crowd-source data through a simple smartphone app installation. More critically, the essence of our PoC can be extended by far away nation-state actors with advanced capabilities for deploying botnets [15] through compromised smartphones and BLE IoT devices. If adapted as malware, innocent

users across the entire world may unknowingly contribute or fall victim to a larger attack aimed to destroy public trust and confidence in DCT protocols.

More importantly, the contribution's second aspect includes our proposal of GAEN$^+$ (§4), an elegant variant of the GAEN framework that increases its resiliency against a geographically distributed replay attack by introducing geospatial awareness. After recognizing the weaknesses of the existing GAEN protocol and the proposed solutions from Raskar et al [13] and *ACTGuard* [14] (detailed in §8), we propose a solution that does not modify, add any sensitive location data to the transmitted protocol fields, or introduce additional infrastructure. Instead, GAEN$^+$ slightly modifies the existing key derivation to include location context provided by a hierarchical geospatial index while still preserving bi-directional anonymity.

Chapter 2: Background

In this chapter, we discuss the systematic characterization (§2.1) and introductory summary of three BLE enabled DCT protocols. Then we focus on the GAEN framework in particular (§2.2.1), describe its protocol specification (§2.2.2) and then discuss various attacks and countermeasures (§2.2.3).

2.1 Digital Contract Tracing Protocols

The following section provides descriptions for systematic characterization of DCT protocols which provides necessary background to then discuss three BLE-enabled DCT protocols: DP-3T, BlueTrace, and GAEN.

2.1.1 Characterization

Digital contact tracing protocols can be understood and compared by three defining operational characteristics:

1. Detection of encounters
2. Determination of epidemiological significance
3. Communication of encounters and exposure

Each characteristic lies on the spectrum of centralized and decentralized approaches. Generally, one can think of centralized approaches having access to more granular data such as user location, age, gender, personally identifiable information (PII), etc, and therefore able to process large, multi-variate data sets on high performance computing environments. However, this increased data utility often comes at the cost of user privacy. Decentralized approaches on the other hand push computation to the edge, often using pseudo-random tokens to avoid potential for identification.

When determining an encounter, a centralized approach may capture a single device's location and time and upload to an aggregating, remote server which ingests space-time points to determine proximity. However, a decentralized approach may transmit beacons over a nearby wireless communication protocol between devices using a broadcast protocol, storing captured beacons in local storage.

When determining if an encounter has epidemiological significance, a centralized approach may easily run processor-intensive algorithms that yield more accurate results, save data in large quantities to provide insight overtime to epidemic, disease, and other related medical researchers, and provide public health insight to mitigate and predict patterns for spread. Whereas a decentralized approach may run analysis completely on the edge device and therefore must consider processing and power constraints, potentially using more simple algorithms that yield less insightful results.

Finally, communication of encounters and exposure in a centralized approach may employ human contact tracers or other public health workers to contact users directly over phone, email, or text, to gather more personal information and recommend quarantine. This contrasts to a decentralized approach where positive cases may be self-identified on a user's device and exchanged over a beacon to other nearby devices.

Exploring these extremes reveals a value calculation that protocol developers must consider in their designs. This calculation navigates the tradespace between data-utility and data-privacy to create a protocol that is both effective and widely adopted. A fully centralized approach may appear to be more capable and effective at determining, gathering information about, notifying users, and providing valuable data to health authorities about potential exposure, but will likely be less adopted if individuals and societies are skeptical about the use of this data. A fully decentralized approach may appear to better safe-guard personal data, but inherently suffers from less accurate determination algorithms, environment context, exposure notifications, and withholds valuable data from epidemiologists.

An optimal solution is likely found between these extremes. The following protocols explored in this thesis each attempt to strike a balance between centralized and decentralized characteristics in order to provide an effective protocol while preserving the greatest amount of privacy.

Protocol	Encounter	Significance	Communication
DP-3T	Decentralized	Decentralized	Decentralized
BlueTrace	Decentralized	Centralized	Centralized
GAEN	Decentralized	Decentralized	Decentralized

Table 2.1: Surveyed protocols characteristics described as centralized vs. decentralized

2.1.2 Enabling Protocols

Numerous existing network protocols are leveraged to facilitate digital contact tracing protocols' core requirement to transmit and receive data. The application of various wireless protocols has been explored as a mechanism to exchange data between DCT applications and determine proximity, to include WiFi, GPS, Zigbee, and Bluetooth. Trivedi et al present a solution with *WiFiTrace* which utilizes WiFi network logs to passively determine user proximity and trajectory [16]. *Safe Paths* from researchers at MIT Media Lab propose utilizing fuzzy, time-stamped GPS trails to determine proximity with infected users. Bluetooth Low Energy emerges as the dominate wireless technology utilized to determine encounters with other individuals.

2.1.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is one of two Bluetooth wireless communication protocols developed for lower power consumption devices [17]. BLE is capable of providing point-to-point, broadcast, and mesh device communication network topologies

in addition to its device proximity, direction, and distance positioning capabilities. Operating at the 2.4GHz unlicensed ISM frequency band, BLE features over 40 channels with 2 MHz separation providing 3 advertising and 37 data channels.

The protocols explored in this thesis all utilize BLE to determine encounters with other participating devices. As described above, this makes these protocols' approach to encounter determination decentralized. Further, they utilize a smartphone's BLE interface Receiver Signal Strength Indicator (RSSI) to determine distance as part of the significance calculation. However, research from Q. Zhao et al [18] observed 20 BLE DCT apps to conclude several factors can make the accuracy of this calculation challenging, including broadcasting frequency, TxPower, and lack of specific tuning.

2.1.4 DP-3T

The first protocol we explore is DP-3T [8], developed by an international group of technologists, legal experts, engineering and epidemiologists, primarily from universities. It features decentralized proximity tracing through BLE beacons, mostly decentralized communication of data through non-attributable data uploads to servers and subsequent broadcasting to other users, and an on-device, decentralized encounter significance computation. The latest revision of DP-3T features three designs: low-cost, unlinkable and hybrid. The specifications are open-source and provided with demo implementations [19].

Once a DP-3T powered smartphone app is downloaded, a user is not required to provide any personally identifiable information. Upon first-time initialization, the

app generates a random secret key, SK_0. Daily, the app would use the previous day's secret key as an argument to a hash function to generate the current day's secret key, SK_T:

$$SK_T \leftarrow H(SK_{(T-1)}) \tag{2.1}$$

Using the newly generated key for some day T, a set of n ephemeral IDs (eIDs) are generated using a pseudo-random function (PRF), combined with a shared, predetermined, constant string, and used as the argument to a pseudo-random generator (PRG), such that:

$$eID_1||eID_2||\ldots||eID_n \leftarrow PRG(PRF(SK_T, \text{“} < constant\ string > \text{”})) \tag{2.2}$$

With each passing epoch, recommended to be approximately 15 minutes, the application will randomly choose a new eID to broadcast as its payload in a BLE advertisement. There is no additional data in the BLE packet that potentially identifies the user and epochs randomly vary in time to reduce temporal pattern matching. While broadcasting an eID, the app is also capturing BLE advertisements of similar characteristics, storing them on the device with a timestamp and RSSI.

Upon a user's positive diagnosis, a health authority provides an anonymous, random code to the user to enter into the app. The user now decides still whether to upload their secret keys over a chosen date range. In effect, the user uploads a set of SK and T pairs:

$$UploadSet \leftarrow \{(SK_i,\ T_i) \mid T_d\ \geq i\ \geq T_e\} \tag{2.3}$$

Where each i is in the range from the day of diagnosis T_d, to a chosen day no earlier than a date that is determined to have epidemiological relevance, T_e. In the case of COVID-19, the range would include 14 days. Therefore, a user wanting to fully share their max amount of daily generated *SK* would upload 14 total pairs.

Other participating devices download large sets of new *(SK, T)* pairs daily. For each day T, a device will use the SK_T to compute sets of eIDs to search for matches in its local database of previously captured eIDs. Upon finding a match, a distance calculation and potential duration calculation is performed to determine the encounter's significance. If the predetermined significant-encounter threshold is surpassed, the device notifies the user. As days pass, captured eIDs and generated *SK* that are older than the epidemiological relevant time period (i.e., 14 days for COVID-19) are deleted from the device's local storage.

2.1.5 BlueTrace

BlueTrace was developed by Singapore's Government Technology Agency and Ministry of Health [3]. BlueTrace features decentralized proximity tracing through BLE handshakes, centralized communication of data through encrypted but attributable by phone number and PIN to health authority servers, and a decentralized encounter significance computation on individuals devices. Utilizing the BlueTrace privacy-preserving protocol, TraceTogether is the first nationally deployed Bluetooth-based application and contact tracing system in the world. Subsequent to over 50

governments expressing interest, Singapore's Government Technology Agency released OpenTrace, the open-source implementation [20].

Once a BlueTrace-powered application is installed, the user provides a phone number and is returned a randomized *UserID*. This phone number is stated to enable a human-in-the-loop component, allowing a contact tracer to call an individual who has notified the server of a positive diagnosis for any clarifying questions. BlueTrace apps transmit *TempIDs* that are AES-256-GCM encrypted payloads consisting of the *UserID* and start and end times for lifetime and validity checking. This payload is combined with an initialization vector (IV) and *Auth Tag*, then Base64 encoded, totaling 84 bytes. Only the health authority has access to the encryption key to encrypt and decrypt the encrypted payload and therefore logically must generate and send *TempIDs* to each user. Batches of *TempIDs* are sent to account for potential network instability or a device being offline for extended periods.

As opposed to DP-3T that simply broadcasts and captures BLE advertisements, BlueTrace performs a typical BLE handshake that relies on standard BLE protocols, such as Generic Attribute Protocol (GATT), to facilitate device discovery and Encounter Message (EM) sharing through characteristic exchanges. The transmitted EM is in JSON format and consists of a user's *TempID* for a specific time window, device model, health authority organization code, and protocol version. The RSSI, as seen from the Central device, and device models are shared so each device can better estimate distance during processing.

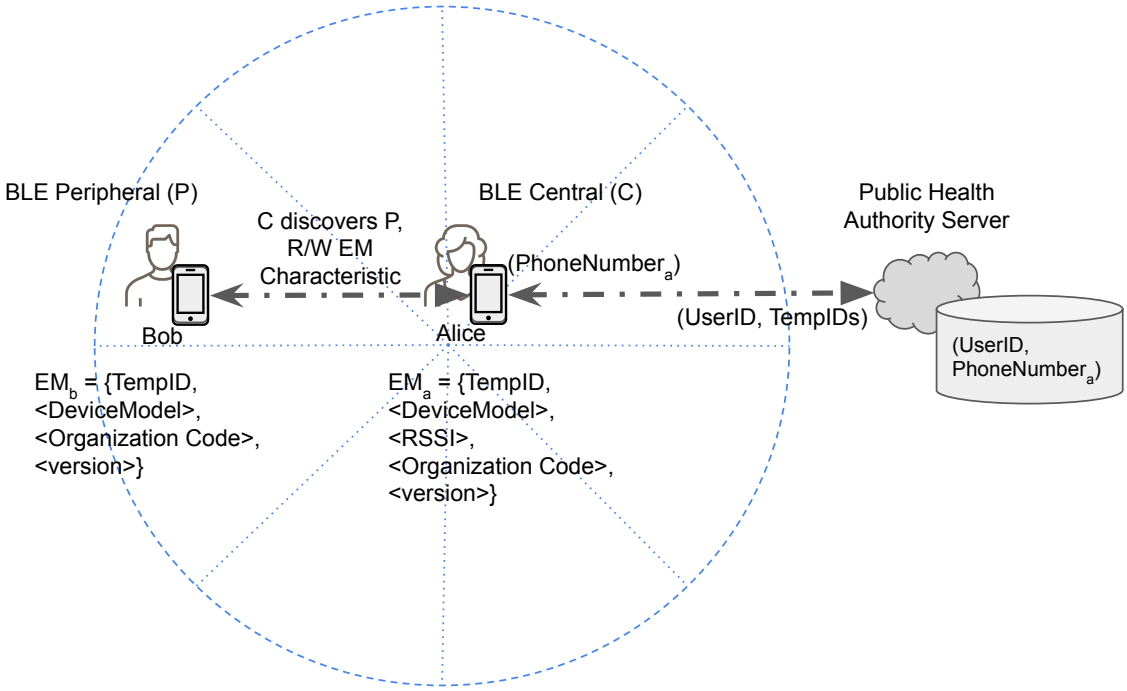


Figure 2.1: BlueTrace registration and BLE exchange of Exposure Messages from Central and Peripheral devices

Upon positive diagnosis, a user receives an authorization code from the health authority and decides whether to upload captured EMs. The health authority decrypts the transmitted data and determines the epidemiological relevance of each encounter using time, RSSI, and device model as factors. Since their database holds at least a *UserID* and phone number pair for each participant, manual contact tracers can now reach out to other users that may have encountered the positively diagnosed individual. As provided by the OpenTrace implementation, devices delete captured EMs after a 21 day period.

2.2 Google/Apple Exposure Notification (GAEN) Framework

The following provides an introduction GAEN and a closer look at its cryptography and BLE specifications. We then further explore attacks against GAEN, particularly focusing on geospatial and temporal characteristics that enable replay attacks.

2.2.1 Introduction to GAEN

GAEN is heavily based on the decentralized, privacy-preserving proximity tracing system, namely, DP-3T [8]. The primary enabling technology is the use of smartphones' BLE chipsets to continually broadcast and capture pseudorandom ephemeral tokens or identifiers. Captured identifiers are later compared against anonymous identifiers derived from keys provided by others who are diagnosed positive for COVID-19. The occurrence of an encounter is determined on a user's smartphone, anonymous keys are shared with other users through a central server that does not store personally identifiable information, and the significance of an encounter is again determined on each individual's device. These qualities make GAEN largely a decentralized protocol.

While many other DCT protocols are implemented entirely in a standalone smartphone app, GAEN is provided as an SDK framework for iOS and Android app developers at approved public health authorities to utilize, making the core cryptographic

and BLE functionality implemented at the operating system level. Providing a framework also enables the capability for iOS applications to continually broadcast BLE beacons, overriding the typical behavior which shuts down critical BLE functionality when an application is in the background or the smartphone's screen is off.

Privacy & Adoption. With a combined approximate 99% mobile OS global market share [12], a framework provided by Google and Apple intuitively has the highest potential for wide-spread adoption. This potential underscores the responsibility of their researchers to release a framework which users can understand and feel their privacy is preserved while providing the most utility to combat the spread of highly contagious diseases. To this end, both companies have made freely available online the specifications, documentation, terms of use for application developers and health authorities, source code and reference implementations, and explanations of the privacy preserving characteristics of GAEN [10, 11]. These privacy preserving characteristics and policies include [10]:

1. Users opt-in to share their anonymous tokens upon positive diagnosis with the health authority.
2. No location collection or tracking, only determination of proximity to other anonymous devices.
3. Identity and diagnosis status is not shared with other users, Apple, or Google.
4. Applications are limited to approved public health authorities who must adhere to privacy, security, and data use restrictions [21].

These characteristics trade data-utility for data-privacy. While these likely benefit adoption, they also limit the potential to ultimately reduce the spread of disease through more accurate exposure notifications. GAEN is very specific to only include BLE backed technology for proximity detection without location services, despite many useful and commonly used apps, such as navigation, utilizing GPS.

2.2.2 GAEN Specification

In the following, we summarize the GAEN framework cryptography and BLE specifications [22, 23] to provide sufficient context for the attack analysis and countermeasure recommendations.

Pseudo-random Key Derivations. Once a user has downloaded and confirmed installation of a GAEN-powered app from Google Play Store or Apple App Store, a Temporary Exposure Key (TEK) is generated daily using a cryptographic random number generator function (CRNG):

$$TEK_i \leftarrow CRNG(16) \tag{2.4}$$

and stored securely, along with its creation time interval i, for 14 days on the user's device. The current day's TEK is then combined with a null salt value and the static string "`EN-RPIK`" to generate a 16-byte Rolling Proximity Identifier Key (RPIK) through an HMAC Key Derivation Function (HKDF):

$$RPIK_i \leftarrow HKDF(TEK_i,\ NULL,\ \text{“}\texttt{EN}-\texttt{RPIK}\text{”},\ 16) \tag{2.5}$$

The current RPIK is combined with 16-bytes of padded data, consisting of the static string "`EN-RPI`" and a discrete time interval value, as input to a symmetric encryption algorithm, $AES128$. A 32-bit Exposure Notification Interval Number (ENIN):

$$ENIN(UnixTimestamp) \leftarrow \frac{UnixTimestamp}{60 \times 10} \tag{2.6}$$

is derived from 10 minute windows starting from the Unix Epoch and therefore allow all participants to use the same values for key derivation, including the j-th interval:

$$ENIN_j \leftarrow ENIN(j) \tag{2.7}$$

for TEK derivation. With the additional 6-NULL bytes added in the padding:

$$PadData_j \leftarrow (\text{“}\texttt{EN}-\texttt{RPI}\text{”},\ NULL,\ ENIN_j) \tag{2.8}$$

the framework eventually generates the ephemeral Rolling Proximity Identifier (RPI) for a given time interval, resulting in 144 derived per day:

$$RPI_{i,j} \leftarrow AES128(RPIK_i,\ PadData_j) \tag{2.9}$$

Ephemeral Payloads. A new 16-byte RPI is generated every 10-15 minutes, coinciding with the operating system's rotation of the broadcasting BLE address to reduce the potential for identification and long-term tracking. A 4-byte Associated Encrypted Metadata (AEM) field is generated containing version information and transmission power to assist in distance calculation, as well as reserved bytes. An RPI and AEM is combined to form the service data portion of a BLE payload. The

BLE advertisement includes a *0xFD6F* service UUID that allows applications and chipset interfaces to apply filters. The complete BLE advertising payload is broadcasted several times per second.

Scanning. Every 2-5 minutes, the GAEN-powered app opportunistically enters a scanning mode for approximately 10 seconds. Each captured RPI is paired with its BLE RSSI value and timestamp before being securely stored in a database on the user's smartphone for 14 days. These captured RPIs never leave the smartphone.

Exposure Notification. Upon positive diagnosis, a user is provided a submission key from the app's governing public health authority and is given the option to upload Diagnosis Keys (DKs) to a centralized server managed by a public health authority. The set of DKs consists of a range of TEK and respective creation ENIN_i pairs, $(TEK_i,\ i)$, stored on the device for up to 14 days. If a user is not positively diagnosed, the pairs do not leave their device. DKs from all positively diagnosed users who choose to share are aggregated on the server and sent periodically to other app users.

The app now uses the aggregated TEK and ENIN_i pairs to reproduce the RPIKs and subsequent RPIs. The RPIs derived from the anonymous TEKs are then compared to the RPIs captured and stored in the user device's database. Upon a successful match, the proximity and significance of an encounter is determined through an RSSI calculation. If an encounter is considered significant, the user receives an exposure notification from the app with healthcare guidance.

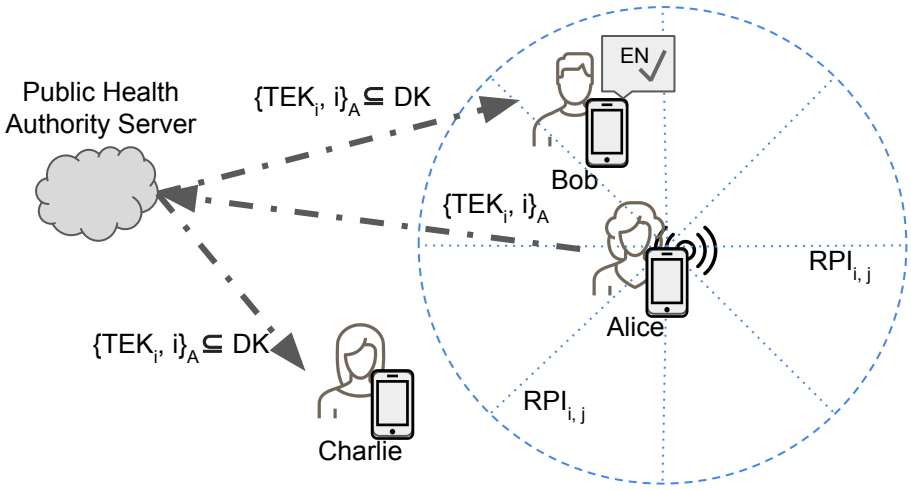


Figure 2.2: Current GAEN framework operations among honest users

Scenario. Figure 2.2 depicts a typical scenario between honest users Alice, Bob, and Charlie, where Alice and Bob exchange RPIs as they are within BLE range of each other. Upon Alice's positive diagnosis, she anonymously uploads $(TEK_i,\ i)$ pairs to the public health authority which combines them with other pairs to periodically distribute to Bob and Charlie to calculate potential encounters. Since Bob determines he was in close proximity anonymously with Alice, he receives an exposure notification.

While some GAEN-based apps may ask users to optionally provide their phone number during initial signup to assist human contract tracing efforts, no other personally identifiable information (PII) is required by or uploaded to the server. Therefore, the identity of the individual(s) for whom a user was exposed remains private.

2.2.3 Attacks Against GAEN

DCT protocols such as GAEN are subject to various attacks [24, 25], including (I) Denial-of-Service (DoS), (II) sniffing, (III) tracking, (IV) identification, and (V) replay particularly due to their reliance on BLE for connection-less, wireless data transmission broadcasting. In the following, we review these attacks and discuss countermeasures incorporated thus far into GAEN.

(I) Denial-of-Service. A DoS attack aims to degrade or limit access to a service. In the case of BLE, this may present itself as packet flooding such that the chipset is unable to keep up with incoming transmissions and is forced to drop, potentially valid, packets. Another type of DoS occurs after successful capture and during data processing. If a protocol does not limit the type or amount of packets it will process, finite system resources such as storage can become exhausted which may result in degraded application or overall system performance. Tech-savvy users can modify the minimum and maximum broadcast interval parameters of an advertisement of a device's BLE interface to increase transmission rate of packets. With inexpensive USB BLE dongles, the number of broadcasted advertisements in a given time span can be easily scaled.

Protocols like GAEN that transmit connection-less, pseudo-random payloads are easy targets for DoS attacks. As such, GAEN mitigates the impact from storage DoS attacks by only scanning for advertisements opportunistically once every 2-5 minutes. This results in approximately 415 scans per day on average. Each GAEN advertising

payload is 20 bytes, which includes the 16-byte RPI and 4-byte AEM, and is stored for 14 days. If we assume a constant attack with a target smartphone capturing an average of 100 unique GAEN advertisements per scan window, this totals to 581,000 after 14 days and requires approximately 11.62MB of storage. Given typical modern smartphone storage capacities ranging from 32-256GB and more, this DoS attack's impact on storage is minimal.

(II) Sniffing. As a wireless communication protocol, BLE is inherently susceptible to sniffing attacks. BLE utilizes adaptive frequency hopping to reduce collisions [26] for connection-oriented data transmission, which makes following stateful transmissions between two devices more difficult. However, the connection-less nature of advertisements in GAEN removes this barrier, requiring simply a BLE enabled device that follows the protocol specification to receive BLE advertisement payloads.

Therefore, there is not a mitigation GAEN can employ to address sniffing BLE advertisements. Doing so would logically contradict the very purpose of advertising presence or services. Consequently, the intended ease of BLE advertisement capture paves the way for other attacks, such as tracking and relay/replay.

(III) Tracking. Mobile apps that advertise an identifier are prone to tracking attacks that aim to observe data trends and record an individual's location over time. Essentially, if an advertised attribute (e.g., the BLE MAC address or a payload value) is observed in one location at one time and again at another time, one can

infer the device's trajectory, average speed, and location. This attack becomes more accurate the longer low entropy, identifiable attributes are broadcast.

GAEN attempts to counter tracking attacks by deriving a new RPI with each change in BLE Media Access Control (MAC) address already in place by the Android and iOS operating systems. Specifically in BLE parlance, the advertiser's BLE MAC address is set as Random, Private, and Non-resolvable [27]. This occurs approximately every 10-15 minutes to mitigate timing attacks that attempt to observe a strict change frequency to further support tracking. In effect, this type of ephemeral address hides the true, static, private MAC address of the BLE interface. This reduces the potential for tracking a repeatedly observed BLE MAC address or RPI. However, despite this mitigation, Corona-Sniffer [28] has demonstrated the feasibility for tracking a smartphone using a GAEN-powered app through a deployed network of geographically dispersed BLE receivers.

(IV) Identification Attacks. Malicious tech-savvy actors can develop and deploy additional sensors alongside BLE receivers to enable identification attacks. For example, one can setup a low-cost video camera feed to pair with sniffed GAEN advertisements. With root access to a smartphone with a GAEN-powered app installed, the malicious actor may be able to identify a positively diagnosed user after RPIs are derived from received DKs. They may utilize familiar knowledge, such as recognizing an employee or student, or through advanced facial recognition capabilities.

Attacks that utilize complementary data are difficult to mitigate, particularly when sources include media such as video or sound that are potentially easily captured without notice. For example, to block identification from a video feed, one may require a physical barrier or to disguise their person. Therefore, it is again by the nature of BLE advertisements and the technical skills of a malicious actor that enable this attack that effectively fall out of the scope for GAEN to reasonably address.

(V) Replay Attacks. Generally, a replay attack involves the capture and repeat transmission of data by a third-party. This is typically carried out by a malicious actor who intends to exploit a protocol's weakness to gain access to a system, poison a data set, or cause undesirable system effects.

Replay attacks may be mitigated at different levels of the network communication stack. For example, by using session IDs that cryptographically confirm the originating source, packet sequence numbers, or two-factor authentication. Effectively, these mitigations aim to provide connection state, timing, or other contexts to verify the validity of the transmission. However, mitigating replay attacks with broadcasted, connection-less protocols, such as BLE advertisements, becomes more difficult.

As such, since GAEN transmits its RPIs within BLE advertisements, it is a prime target for replay attacks. While the developers of GAEN are clearly aware of replay attacks, the ephemeral nature of RPIs only provides a weak temporal defense, leaving the geospatial vector open to attack.

- **Temporal Context.** GAEN limits the replay window of a captured RPI to approximately 2 hours. While RPIs are both originally created and later derived using established Unix timestamp intervals, GAEN adds a +/- 2 hour buffer to increase validity assumingly to account for the fuzzyness of RPI rotation and minor time disparities among devices. RPIs are stored by interval when captured by a GAEN-powered app, rendering RPIs replayed outside the interval +/- 2 hours invalid.

  However, an RPI and timestamp interval pair is 20 bytes total, and therefore can be quickly transmitted across the Internet with typical smartphone data rates, even at high volume. For example, with a modest 5 Megabit/second download rate, a smartphone can theoretically download 200 kilobytes (KB), or 10,000 captured $(RPI,\ interval)$ pairs, in just 320 milliseconds. Consequently, distributed RPIs can be replayed by a malicious actor and captured by honest users within this wide window.

- **Geospatial Context.** GAEN relies on the limited range of BLE and the RSSI values at time of capture to calculate proximity to nearby smartphones. However, since the payload lacks geospatial awareness, an RPI is valid when broadcasted in any location. For example, an RPI generated and broadcasted in Brooklyn, New York is equally valid in San Diego, California.

  This lack of geospatial context makes GAEN vulnerable to geographically distributed replay attacks.

Chapter 3: Low Cost, Crowd-Sourced Replay Attacks

While it is well-known in principle that GAEN is subject to replay attacks [24, 25], we are unaware of concrete realizations of a crowd-sourced, geographically distributed replay attack network targeting GAEN, or consideration of the ease of its implementation, deployment, cost, scalability, and the potential severity of their impact. We address these issues in this chapter, in terms of the attack scenario (§3.1), a concrete implementation, (§3.2), and its impact (§3.3).

3.1 Attack Scenario

We consider a tech-savvy, malicious actor who aims to disrupt intended functionality of GAEN. At a high level, this malicious actor has the ability to design and develop smartphone applications using standard operating system SDKs, deploy infrastructure necessary to facilitate an attack, observe and dissect wireless traffic, and modify device components and drivers to deviate from protocol specifications.

To demonstrate exactly how the attack works, we provide a walk-through example shown in Figure 3.1.

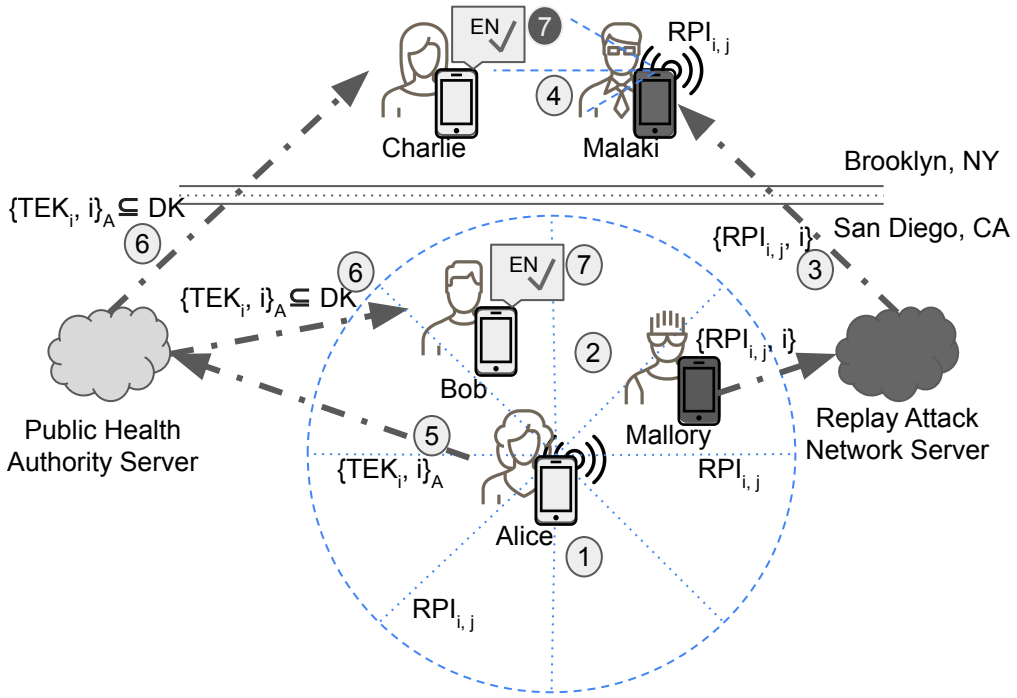


Figure 3.1: Step-by-step attack scenario demonstrating GAEN's vulnerability to crowd-sourced, geographically distributed replay attacks

Assume there are three honest users Alice, Bob, and Charlie who have a GAEN-powered app installed on their smartphone, and two malicious users Mallory and Malaki who do not know each other but have individually subscribed to the replay attack network and have the replay app installed on their smartphone. The replay attack works as follows:

1. Alice generates a TEK and derives an RPIK daily. Then, she creates a new RPI every 10 to 15 minutes and broadcasts it over BLE.

2. Bob and Mallory are within Alice's BLE advertising range, capture Alice's GAEN advertisement, and extract the RPI. Bob stores the RPI for 14 days. Mallory calculates the current ENIntervalNumber i and sends the captured (RPI, i) pair to the replay attack network from his location in San Diego, CA.

3. Malaki, in Brooklyn, NY, receives (RPI, i) pairs, including Alice's, and continuously replays the broadcast of received RPIs in the interval window i.

4. Charlie receives Alice's RPI, replayed by Malaki.

5. A few days later, Alice is positively diagnosed for COVID-19. She elects to submit her (TEK, i) pairs to the public health authority's server.

6. Bob and Charlie receive the set of anonymous DKs from the public health authority server and derive the RPIKs and subsequent RPIs. Bob and Charlie individually find a match between Alice's RPI and an RPI derived from a TEK.

7. Bob and Charlie are both notified of exposure (due to proximity to Alice, though this is anonymized). Bob's notification is valid whereas Charlie's is a false-positive.

3.2 Our Implementation

Simple Network Topology. An anonymous, crowd-sourced, geographically distributed replay attack network easily scales through smartphone apps and cloud services, without requiring specialized hardware. Like-minded malicious actors download the replay app on their BLE enabled and Internet connected smartphones to serve as nodes that query a central server. After a simple, one-time setup, they simply carry their smartphone on their person throughout their normal daily routine capturing, transmitting, and replaying RPIs. The replay attack network central server includes a simple application that receives and distributes captured RPIs from subscribing nodes.

Capture. Each node will continuously scan for BLE advertisements and filter for the `0xFD6F` Exposure Notification Service UUID, as outlined in the GAEN Bluetooth specification [23]. Upon successful capture of a GAEN advertisement, the 16-byte RPI value is extracted from the payload and paired with the current calculated Unix timestamp interval. This interval is calculated exactly how GAEN calculates an ENIntervalNumber for key derivation. To avoid unnecessary traffic from sending duplicates, only unique RPI and timestamp interval pairs are published to the replay attack network central server's message queue. The capture algorithm is described as pseudo-code in algorithm 1.

Distribution. The central server distributes received pairs to all other participating nodes through standard HTTP requests. Received pairs are checked for recent receipt

Algorithm 1: Replay Attack Network: Node - Capture

```
1  hciDevice = HCIConnection(ble_device_id);
2  hciDevice.setServiceUUIDFilter(0xFD6F);
3  server = httpConnection(server_endpoint);
4  adverts = new Dictionary;
5  // Capture Thread
6  while true do
7  |   gaenAdvert = hciDevice.blockUntilMatch();
8  |   rpi = gaenAdvert.extractRPI();
9  |   curTimeInterval = ENIntervalNumber(now);
10 |   if adverts does not contain (rpi, curTimeInterval) then
11 |   |   adverts.add((rpi, curTimeInterval));
12 |   |   server.post((rpi, curTimeInterval));
13 |   end
14 end
```

and uniqueness to remove duplicates that may arrive from two nearby malicious nodes. The server effectively acts as a many-to-many, bent pipe transport mechanism for unique data. By grouping received RPIs in the valid time interval window, the server does not distribute expired RPIs.

Our PoC does not implement additional meta processing, but can be easily augmented to extend its utility with additional data from the nodes, such as location. This enhancement could provide tracing capabilities as demonstrated in previous works [28] but outside the scope of our focus and contribution on demonstrating a geographically distributed replay attack.

Replay. In a separate thread, nodes query the central server every 60 seconds to receive valid RPIs for the current interval, captured by other participating nodes.

Meanwhile, the node is continuously iterating through its local queue of received RPIs. For each pair, the node modifies its BLE address, confirms timestamp interval validity, and constructs and broadcasts the RPI in a valid GAEN BLE advertisement format. The replay of RPIs is described in pseudo-code in algorithm 2.

Algorithm 2: Replay Attack Network: Node - Replay

```
1  server = httpConnection(server_endpoint);
2  rpis = new FIFOQueue;
3  // Server Query Thread
4  while true do
5  |   // Every 60 Seconds
6  |   newPairs = server.queryForPairs();
7  |   rpis.append(newPairs);
8  end
9  // Broadcast Thread
10 while true do
11 |   curTimeInterval = ENIntervalNumber(now);
12 |   curPair = rpis.dequeue();
13 |   if curPair.timeInterval == curTimeInterval then
14 |   |   SetRandomBLEAddress();
15 |   |   BroadcastGAENPayload(curPair.rpi);
16 |   |   rpis.queue(curPair);
17 |   end
18 end
```

App Installation. An app intended to cause disorder and undermine a social good is likely to be removed from the app stores. However, malicious users interested in participating in a replay attack network may understand other methods for app installation, such as "jailbreaking", "rooting" [29], or direct installation via IDEs

or pre-compiled binaries. Meanwhile, motivated nation-state actors may integrate the concepts into malware and a worm that develops a botnet, compromising smartphones and IoT devices connected to the Internet without user awareness - a feasible effort given the numerous 0-click attacks against innocent users' smartphones from nation-state attackers (e.g. [30, 31, 32]). While a 0-click exploit costs millions of USD, it is still a low cost attack for a nation-state.

We have implemented the replay smartphone app PoC using standard Android SDKs for BLE and HTTP requests on a Samsung Galaxy S8. This PoC does not utilize the provided GAEN framework and therefore does not require approval as a public health authority. The central server application is created using Python 3.8, a micro web framework Bottle, and served with Gunicorn WSGI.

3.3 Attack Impact

Our concrete, crowd-sourced, geographically distributed replay attack network can cause severe consequences, albeit we have never deployed it in practice. Below are a few hypothetical consequences.

First, depending on government, public health authority, employer, or academic institution guidance and policies, an exposure notification may require someone to self-quarantine and lead to wide-spread cascading effects to others. Since rapid, accurate in-home tests are not currently common place, teachers or students may miss classes requiring a decreasing supply of substitutes to cover, resulting in continual disruption of education. Food suppliers may become excessively delayed in shipping

causing grocery store stocks to dwindle. Replay attacks continuously and successfully executed in high numbers would become widely publicized, likely resulting in a loss of user confidence and a severe negative impact on GAEN adoption rates. Daily personal activities or long-planned vacations may be cancelled or postponed, exacerbating economic and mental health effects. The scenarios that introduce chaos to typically predictable activities that have health, political, and economic impacts, among others, are countless.

Chapter 4: Countermeasures & Research Goals

The absence of geospatial awareness makes GAEN exposed to geographically distributed replay attacks. If a replay attack network, as introduced by our PoC (§3.2), is deployed and adopted by malicious actors, false-positive rates in otherwise low positivity rate areas would substantially increase. This potentially decreases public trust and adoption rates of GAEN and DCT applications in general. Therefore, GAEN must maintain public trust in order to increase adoption rates and provide its maximum utility for notifying users of exposure to positively diagnosed individuals.

To this end, we establish the following research goals to enhance the current GAEN framework:

- G_1**:** Current location data retrieval without compromising user privacy (§4.1).
- G_2**:** Enhancement of GAEN with geospatial awareness to reduce the effect of replay attacks (§4.2).
- G_3**:** Incorporation of modifications without requiring significant rework of the protocol or significantly changing its performance or user experience (§4.3).

In the following, we describe how we achieve each of these goals in greater detail.

4.1 G_1: Retrieving Current Location

When a concept requires location context, GPS is typically the first solution considered. Fortunately, modern smartphones commonly feature GPS receivers that utilize satellite constellations for its current latitude and longitude. Researchers demonstrate accuracy of GPS data from smartphones varies wildly depending on conditions, however, determine an average horizontal position accuracy of an iPhone 6 to be within 7-13m and frequently under 2m, consistent with other autonomous GPS receivers [33]. While an approximate 23-42 feet falls outside of the Center for Disease Control and Prevention's recommendation of maintaining a distance of 6 feet between others [34], it falls well within BLE range of approximately 300 feet. However, this level of GPS accuracy both may appear too pervasive to user privacy and plainly violates GAEN's Additional Terms that states apps can not collect precise location [21]. Further, traditional GPS solutions typically consume significant amounts of power when active, resulting in noticeable battery drain and degraded user experience.

There are numerous alternative sources for location, including WiFi, Bluetooth, cellular networks, and the use of infrared and ultrasound frequencies [35]. Fortunately, both Android and iOS offer WiFi and cellular network positioning through coarse location services [36], which benefits our use case from the perspectives of adoption and adhering to GAEN's terms for developers. Additionally, these alternatives consume less power and are generally faster to retrieve results since the data is

received from ground systems versus satellite constellations with GPS. Given these characteristics, WiFi and cellular positioning appear to be viable solutions for a smartphone to retrieve its current, approximate location. However, the challenge still remains how to use these coordinates in a meaningful way to provide geospatial awareness.

4.2 G_2: Geospatial Awareness

Now with an appropriate solution to retrieve location, the next goal involves applying this context to the current GAEN framework to reduce the impact of replay attacks. The most obvious way is to include GPS coordinates in the AEM field as part of the advertised payload, as recommended by Raskar et al [13]. However, broadcasting precise or coarse GPS coordinates, even if encrypted, may leave GAEN susceptible to more public scrutiny. Further, GPS coordinates are point values and do not intrinsically provide any surrounding bounds to reduce the space to which an RPI is considered valid and still requires a calculation on the receiving smartphone. This inspires the use of a predetermined partition of area, restricting RPI validity from the entire Earth to something more reasonable.

Using Political Boundaries. Naturally, we first considered established and adopted conventions for geographic partitions, such as political boundaries, to include states, counties, and zip codes. In the United States, development of GAEN-powered applications are left up to state-level health authorities and governments. If each state

uses their own backing server to upload and exchange keys, this would provide up to 50 distinct geographically distinct zones.

However, the Association of Public Health Laboratories hosts a national key server that U.S. states can utilize to increase effectiveness during travel across state lines. As of September 2021, nearly half of the U.S. states are participating [37]. While a national server aims to ensure users in all states receive DKs despite their originating state, it also increases the space for which a geographically, wide-reaching replay attack is valid. However, this seems like a reasonable trade-off that can simply be addressed through a creative solution in the protocol itself. Further, given their non-uniformity of area and representation of population densities, state boundaries make for seemingly arbitrary divisions from protocol and statistical perspectives. Finally, using states as a geographic separation would simply prevent a malicious subscriber from receiving RPIs from other states with the simple counter of focusing malicious adoption on more concentrated areas of cases within state lines. These issues do not support using state boundaries to as a geospatial component to mitigate replay attacks.

Counties and ZIP codes are the next intuitive alternatives that provide more granular geographic bounds compared to states. However, these still suffer from the same population density and size issues discussed above. Further analysis reveals an issue around encounters near borders where two users are only feet apart but technically in different zones. The reasonable solution is to include neighboring zones during

RPI derivation from received TEKs. But accurate calculation becomes intuitively very difficult when considering the number of edges a county or ZIP may have. If we simply include neighboring counties, non-uniform size and population densities again make the results of this calculation vary wildly. A relatively small county may border a significantly larger neighbor which may drastically and inaccurately increase the geographic search space.

Using a Hierarchical Geospatial Index. The need to address zone size, population density, and cross-border calculation issues leads us to consider more uniform, hierarchical geographic data structures. The underlying recursive algorithms take latitude and longitude coordinates and output index values at various resolutions that are contained within each other, providing their hierarchical characteristic. We next systematically examine a number of open-source geospatial index implementations, which include Geohash [38], Google's S2 Geometry library [39], and Uber's H3 [40]. The key differences between these solutions largely involve the shape of the geofenced area represented by the output value from provided coordinates, as illustrated in Figure 4.1, providing other defining characteristics.

1. **GeoHash.** GeoHash is a hierarchical, geospatial system that encodes latitude and longitude coordinates into alphanumeric strings. Using Z-curves [41], it effectively subdivides maximum resolutions of longitudinal coordinates from -180 to +180 degrees and latitudinal coordinates from -90 to +90 degrees for each successive bit pair. The higher or lower interval is chosen with bits 1 and

0, respectively. For example, a latitude value starting with 1 will make the current latitude interval 0 to +90, producing an effective error of ±45. This produces a sub-divisible collection of non-overlapping rectangles as precise as the number of bit-pairs.

While GeoHash is relatively straight forward, its simplicity introduces undesirable flaws. Generally, shared proximity between two points is easily determined by matching common prefixes. However, this is not true for points on either side of the 180 meridian due to its Mercator projection [42], where they would have different prefixes on each side. While this is an edge case, it represents an unnecessary limitation that is addressed by other available solutions. Further, GeoHash's geometrical representation is rectangular with varying areas, depending on the latitude input, due to its two-dimensional projection on the Earth's spherical shape. This results in varied error rates for proximity detection that again introduce an unnecessary limitation given alternatives.

2. **Google's S2 Geometry.** Google's S2 Geometry library overcomes GeoHash's geometric limitations by providing data overlaid to a three-dimensional sphere that better represents the Earth. Latitude and longitude are converted to indices representing geodesic squares of consistent area depending on the resolution. The spherical geometry approach also removes the discontinuity limitation around the 180 meridian. While S2 is an improvement over GeoHash, it still provides a less optimal solution compared to others when considering

neighboring areas. This is due to its geodesic square zone representation, requiring consideration of both edge-to-edge and diagonal neighbors, eight in total. Further, distance calculation from a zone's center to its neighbors is non-uniform, as edge-to-edge neighbors have smaller distances than diagonal neighbors.

3. **Uber's H3.** Developed by Uber, the H3 hierarchical geospatial indexing library overcomes S2's geometric limitation from using a geodesic square representation by utilizing hexagonal areas instead. Hexagons provide the useful characteristic that all neighboring cells' centers are equidistant, removing S2's non-uniform distance calculations. Additionally, the number of nearest-neighbor cells to process is reduced from 8 to 6. Hexagons also impact the hierarchical overlap; rather than one resolution perfectly containing its child rectangles from a higher resolution, hexagons will have small overlaps which may provide interesting optimizations for neighbor calculation. H3 provides 16 levels of resolution that range from providing 122 unique indexes at approximately 4.2×10^6 km^2 at resolution 0 to 1.1×10^{13} unique indexes at approximately 4.3×10^{-5} km^2.

Our analysis leads us to choose H3 as the more optimal geospatial index solution. While GeoHash and S2 perform well in typical cases, H3 is able to overcome more edge cases and provides a more uniform grid overlay of the Earth, thereby reducing the false-positive and false-negative rates from neighboring zone calculations.

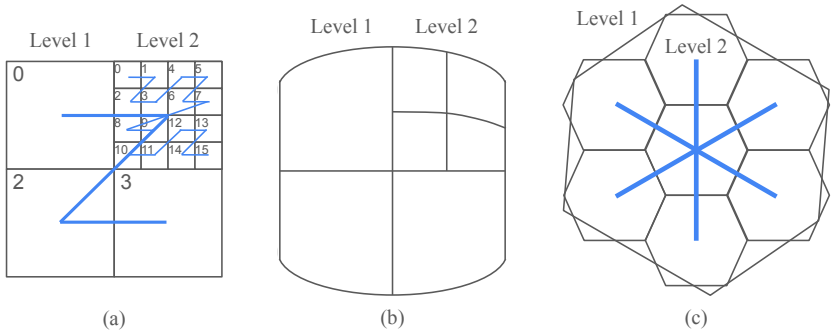


Figure 4.1: Hierarchical geospatial indexes with (a) GeoHash Z-Curve encoding (b) S2 geodesic squares (c) H3 hexagons

4.3 G_3: Minimal GAEN Modifications

The final goal is to incorporate the geospatial awareness in a manner that would not require drastic changes to the existing GAEN protocol, reduce privacy, or require additional third-party services. Once again, including the fuzzy, bounded area in the AEM field presents an obvious first consideration. However, the property that two different smartphones are able to separately determine a shared, bounded area index prompts us to consider GAEN's pseudo-random key derivation algorithms as discussed in §2.2.2.

As shown in the specification background, GAEN creates three keys during its derivation scheme. TEKs are sent to the public health authority by users who opt-in upon positive diagnosis and derived RPIs are broadcasted for others to capture. This makes the RPIK particularly interesting because its placement between the

TEK and RPI make it the only key that truly remains on the smartphone and is not exposed to other parties. *The RPIK is thus an ideal candidate to include a geospatial component to achieve the desired property.*

In effect, *using the smartphone's calculated geospatial index as the salt value — instead of just using the NULL bytes during RPIK derivation*— adds geospatial awareness through a bi-layer indirection. No location context is directly included in the RPI payload nor in the TEKs uploaded to the central server. As a result, this key differentiator from other solutions maintains GAEN's current level of user privacy.

By using the geospatial index as the salt, the static string "$EN-RPIK$" is left intact and therefore still adhering to the GAEN specification and its ability to create an Associated Encrypted Metadata Key (AEMK). Therefore, our enhanced protocol GAEN$^+$ introduces a Location-based Rolling Proximity Identifier Key (LRPIK) that includes the geospatial index, Loc_x, as its salt argument instead of the NULL byte as in the RPIK derivation in the original GAEN:

$$LRPIK_x \leftarrow HKDF(TEK_i,\ Loc_x,\ \text{“}EN-RPIK\text{”},\ 16) \tag{4.1}$$

GAEN$^+$ generates a single daily TEK as before. Now, the smartphone will retrieve its coarse location coordinates to query a geospatial index periodically to detect movement into a new zone and with each new RPI computation. A new LRPIK will be derived whenever the geospatial index query returns a new value, implying the user is in a new zone. When the returned geospatial index matches the previous

query return value, the LRPIK does not need to be derived again, implying the user is still in the same zone, reducing unnecessary calculations. Finally, the RPI is then derived from the current $LRPIK_x$ and broadcasted as before with GAEN.

We now consider new scanning and RPI derivation behavior. As before, smartphones within BLE range will scan and receive nearby GAEN advertisements containing an RPI. Now, after a scan which results in at least one successful GAEN advertisement capture, the current geospatial index value, Loc_x, is stored on the smartphone's local, secure storage L. The smartphone will periodically download the DKs from the central server as before. Similar to advertising, each Loc_x in L will be used to derive a new $LRPIK_x$ for each TEK. For each derived $LRPIK_x$, the set of RPIs is derived and compared with previously captured RPIs for a given ENIntervalNumber as before.

Effectively, our enhancement defines the geospatial bounds that a derived RPI is valid. If the RPI is broadcasted and received in the same bounds as it was derived, its derivation from a TEK provided by a positively diagnosed user will result in a match. Otherwise, a match will not occur. As a result, GAEN$^+$ can drastically the number of false-positives introduced by a replay attack network demonstrated by our PoC in §3.

Chapter 5: GAEN$^+$ Implementation

We now discuss the implementation details and algorithms of GAEN$^+$. Our proposed enhancements to the GAEN framework align with our research goals (§4): (I) enabling location services permissions for coarse latitude and longitude coordinates (II) including Uber's H3 hierarchical geospatial index, and (III) modifications of the algorithms to include LRPIK derivation without requiring drastic rework of the original GAEN protocol.

5.1 Enabling Coarse Location Services

We arrived at using coarse location services for its numerous benefits for our use case over standard GPS. Compared to precise locations offered by standard GPS, coarse location services adheres to GAEN's Additional Terms, helps preserve privacy, and consumes less power by using location data from WiFi access points and cellular towers. Android provides resolution approximate to a city block [43] or 311 feet [44]. For iOS, the *Significant-change Location Service* provides updates approximately

every 1600 feet. While this represents a 5x difference, we discuss future options for iOS later in §7.

The first modification to the framework is to require users to grant app permissions for coarse location services. On Android, this includes adding the *ACCESS_COARSE_LOCATION* and *ACCESS_BACKGROUND_LOCATION* permissions to allow access to data while the app is in the background [43]. On iOS, this includes the *Significant-change Location Service* and *Always* authorization [36]. If a user disables GPS, a notification should appear to the user that the contact tracing application is unable to operate, similar to the notification that currently appears when a user disables Bluetooth with a GAEN-powered app installed.

5.2 Calculating Hierarchical Geospatial Index with H3

GAEN$^+$ features H3 as the hierarchical geospatial index to provide geospatial awareness. H3 is written in C and compiled as a shared library with bindings available in Java and other languages [45], making it easily portable to both Android and iOS. The library size is approximately 200 kilobytes and therefore, when compared to the average size of smartphone apps in the tens of megabytes [46], the additional size is negligible.

The coarse latitude and longitude coordinates provided by the location services will be fed into an H3 function to calculate an index, given a resolution:

$$long\ geoToH3(double\ lat, double\ lng, int\ res);$$

H3 overlays a hexagon grid to provide coverage for the entire Earth with 16 resolutions [47]. We recommend resolution 10 which provides an average hexagon edge length of approximately 216 feet and provides over 33 billion hexagons with approximately 1.5×10^{-2} km^2 of area. We discuss future work for dynamic resolutions in §7.

5.3 Modification of GAEN Algorithms

Our proposed solution requires minimal changes to the GAEN cryptography and BLE specifications. Specifically, (1) RPIK derivation must include the geospatial index as a salt value. This includes deriving the RPIK to both create RPIs to broadcast and match for exposure notification. (2) Storing the geospatial index whenever an RPI is received to later use for matching. This index value is stored for 14 days and not paired with any additional information; it is therefore temporally agnostic within the 14 day period. Algorithm 3 shows the addition of a loop on line 5 that uses the previous 14 days geospatial indexes as salt values to derive LRPIKs for matching against RPIs from a DK.

Next, the framework must periodically retrieve the smartphone's coarse GPS coordinates to compute the geospatial index. If the index value changes from the last query, the framework must generate a new LRPIK and RPI, change the BLE advertising address, and continue broadcasting the new GAEN advertisement. This feature is outlined in Algorithm 4.

Algorithm 3: GAEN$^+$ RPI Matching

```
locs = getLocFromLocalSecureStorage();
rpis = getRPIFromLocalSecureStorage();
dks = getDKPublicHealthAuthorityServer();
for dk in dks do
    for loc in locs do
        lrpik = HKDF(dk.TEK, loc, "EN − RPIK", 16);
        curInterval = ENIntervalNumber(dk.i);
        for interval in [0...144] do
            curInterval += interval;
            rpi = AES128(lrpik, curInterval);
            if rpi in rpis[curInterval] then
                // Encounter significance calculation per GAEN
            end
        end
    end
end
```

Algorithm 4: GAEN$^+$ Broadcasts

```
while true do
    latestLatLng = GetCoarseLocation();
    latestGeoIndex = GetGeospatialIndex(latestLatLng);
    if curGeoIndex != latestGeoIndex then
        curGeoIndex = latestGeoIndex;
        SetRandomBLEAddress();
        rpik = HKDF(TEK_i, curGeoIndex, "EN − RPIK", 16);
    end
    if 10-15 minutes has elapsed then
        SetRandomBLEAddress();
        enin = ENIntervalNumber(now);
        rpi = AES128(rpik, ENIN);
    end
    BroadcastGAENPayload(rpi);
end
```

5.4 Mitigated Replay Attack Scenario by GAEN$^+$

This replay attack network scenario is provided for the curious reader and demonstrates the mitigation by GAEN$^+$. Once again, it involves honest users Alice, Bob, and Charlie who have an app installed on their smartphone that now uses our proposed GAEN$^+$. As illustrated in Figure 5.1, Mallory and Malaki are malicious actors who have subscribed to the replay attack network and have the replay app installed on their smartphone. To demonstrate the defense and mitigation of our proposed variant, Mallory and Malaki operate as before.

1. Alice generates a TEK daily. She queries the geospatial index and uses it to derive the current LRPIK for her location in San Diego, CA. Then, she creates a new RPI every 10-15 minutes and broadcasts it over BLE. She periodically checks if her location has changed to warrant a new RPIK and RPI derivation.

2. Bob and Mallory are within Alice's BLE advertising range, capture Alice's GAEN advertisement, and extract the RPI. Bob stores his current geospatial index and the RPI for 14 days. Mallory calculates the current ENIntervalNumber i and sends the captured $(RPI,\ i)$ pair to the replay attack network from his location in San Diego, CA.

3. Malaki, in Brooklyn, NY, periodically receives $(RPI,\ i)$ pairs, including Alice's. He determines the current interval is i, and continuously replays the broadcast of all matching received RPIs.

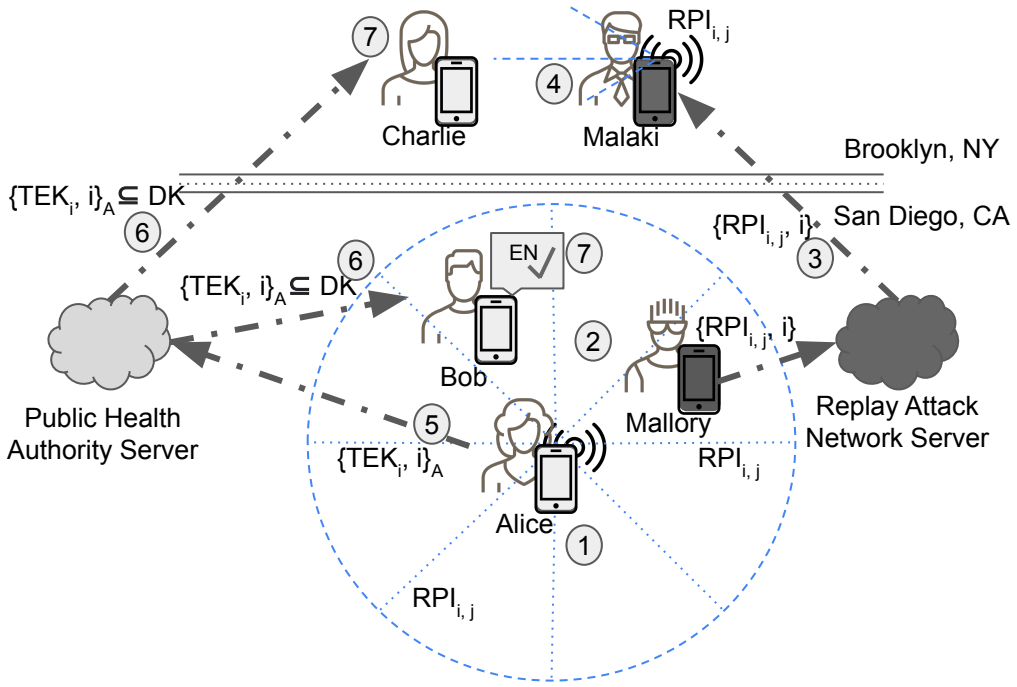


Figure 5.1: Our proposed GAEN$^+$ adds a geospatial component to decrease potential for broad, cross-region replay attacks

4. Charlie is nearby Malaki and receives Alice's replayed RPI. He stores his current geospatial index and Alice's RPI for 14 days.

5. A few days later, Alice receives a positive diagnosis for COVID-19. She elects to submit her (TEK, i) pairs to the public health authority's server.

6. Bob and Charlie receive the set of DKs from the public health authority server. Using each stored geospatial index, they derive the LRPIKs and subsequent RPIs for each TEK. Bob finds a match between Alice's RPI and a derived RPI

from an anonymous TEK. Charlie does not because he was not in San Diego, CA during the day and time Alice generated her RPI.

7. Bob is correctly notified of exposure (due to proximity to Alice, though this is anonymized). Charlie is not notified of exposure due to capturing Alice's RPI (however, he may from another individual's valid RPI).

Chapter 6: Evaluation

In this chapter, we evaluate the cost of our replay attack network PoC as well as the overhead and performance of GAEN$^+$ compared to the current GAEN framework.

6.1 Replay Attack Network Cost

Deploying and participating in the replay attack network PoC has relatively low-cost and a low-technical barrier from the perspective of both the malicious network's owner and participants.

Infrastructure. The central server application is implemented using Python 3.8, served with Gunicorn, and runs on a typical Ubuntu 20.04 installation. Since this application mostly serves as a bent-pipe for RPI distribution, processing, memory, and non-volatile storage requirements outside of request handling are minimal.

We deployed the central server on a range of default virtual private server (VPS) configurations from DigitalOcean to evaluate the number of subscribers downloading RPIs each offering can accommodate. We use Apache Bench to simulate 25, 50, and

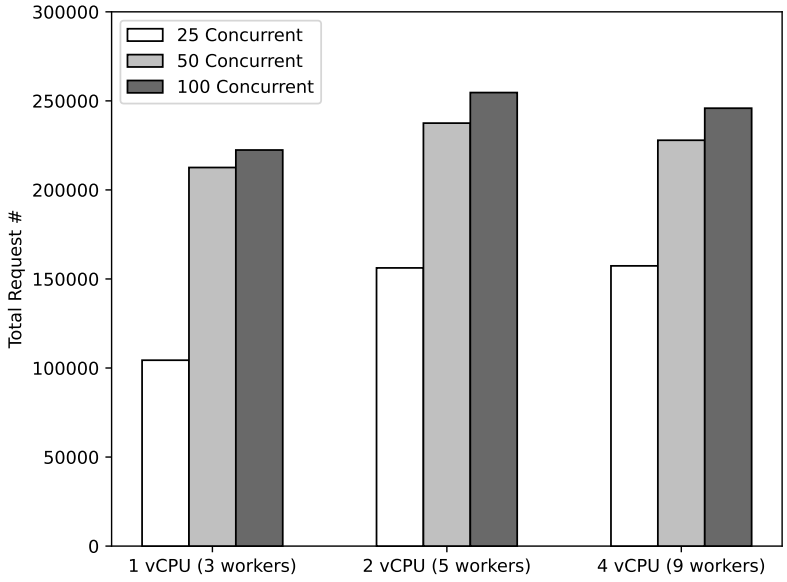


Figure 6.1: Comparing stable virtual CPU performance with total requests served over 10 minutes with 500 RPIs/request

100 concurrent requests over a 10 minute interval window. Each response contains 500 RPIs, totaling 8kB per request.

The results indicate that with default, "1-click" deployments and minimal additional configuration, one can easily accommodate over 100,000 requests per an interval window and scale the replay attack network further. With default configurations, a $5 USD/month VPS with a single virtual CPU is able to accommodate approximately 100 concurrent connections to serve 222,000 requests for 500 RPIs within a 10 minute interval window [48]. This totals to approximately 7,686 GB/month, which would require an additional $80 USD/month in bandwidth charges over 30

days. This cost can be reduced through server optimizations and the use of Content Delivery Networks. In summary, a standard cloud service for $85 USD/month can serve approximately 480 trillion RPIs to malicious subscribers.

The replay attack network can be augmented with additional features, such as persistently storing payloads and associated metadata for further analysis. To scale with subscribers, one could implement the application entirely in a more performant language, such as C, and incorporate load balancers to handle thousands of concurrent connections.

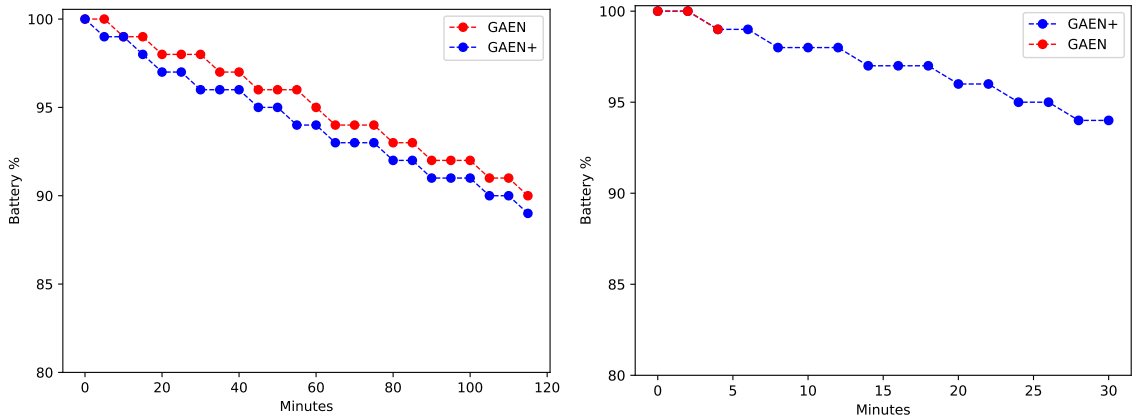
Nodes. To participate in the replay network, a malicious actor simply downloads and installs an app on their smartphone with BLE capabilities and Internet connectivity. Given smartphones are commonplace in the U.S. and typically on an individual's person throughout the day, this low-barrier to participate makes it reasonable to recruit other malicious actors interested in disrupting and creating distrust in GAEN and DCT in general.

Network usage is negligible relative to typical smartphone data packages providing unlimited or multiple gigabytes of data per month. For example, 500 RPI downloads every 10 minute interval results in approximately 1.2MB of data per day. Since replayed RPIs are discarded after no more than 10 minutes, there is no persistent storage requirement outside of the app installation. Power usage is ironically similar to that of a legitimate GAEN application due to continuously broadcasting BLE beacons with the intent to be captured by other users. In effect, the minimal impacts

to malicious actors' smartphones and requirements to join a replay attack network represent a low-barrier to participation.

6.2 GAEN$^+$ Overhead

Power Consumption. We evaluate GAEN$^+$ for power requirements compared to the original GAEN. Since the framework's SDK will only compile if provided a license granted to an approved public health authority [10], we use the reference code provided for Android [49] and implement critical operations for deriving, broadcasting, and matching RPIs.



(a) RPI Derivation and Broadcasting

(b) RPI Derivation and Matching

Figure 6.2: Battery performance of GAEN$^+$ and GAEN over time in different tasks.

Figure 6.2a depicts RPI derivation and advertisement broadcasting power consumption over two hours and reveals GAEN$^+$ is within 1% on average of GAEN. The

remaining battery percentage decreases linearly, as expected, given the consistent operations. While the precision of battery percentage is limited to integer values, this level of granularity still shows the minor difference between GAEN$^+$ and GAEN. If more precise introspection was available, this difference might actually prove to be less than 1%.

Figure 6.2b measures RPI derivation and matching power consumption for $1,400$ TEKs and shows GAEN$^+$ has exactly the same battery consumption with GAEN for the first 5 minutes. However, GAEN$^+$ spends more time during RPI matching due to the additional runtime complexity from the additional LRPIK derivations for each location a user has entered, versus a single RPIK derivation per day with GAEN.

This effect can be mitigated in future work through algorithm optimizations to reduce the computational complexity. Additionally, to minimize any impact to user experience, GAEN$^+$ can defer matching to a time when the smartphone is not actively in use, as recommended by SpreadMeNot to address similar effects and concerns [6].

Network & Storage. GAEN$^+$ does not modify the original GAEN upload or download process of DKs and therefore network traffic remains unchanged and without additional burden. GAEN$^+$ requires storing geospatial index values for which an RPI was successfully captured over a 14-day period. For example, Brooklyn, New York and its surrounding areas has 823,543 cells at resolution 10. At 8 bytes per cell,

this results in only 6.5MB of storage if every cell was visited and captured an RPI within a 14-day period. In practice, the number of visited cells will be substantially lower, resulting in minimal storage requirements.

6.3 GAEN$^+$ Performance

Effectiveness Against Replay Attacks. To evaluate GAEN$^+$'s capability of preventing replay attacks, we simulate a set of experiments and compare the results with the original GAEN framework. The experiment setup is presented in Table 6.1. Specifically, we generate $1,400$ TEKs for 100 GAEN$^+$ (or GAEN) users, which are then used to derive $201,600$ RPIs from the 30 locations spread across the globe at different time intervals. This simulates an attack scenario where a malicious subscriber replays RPIs captured from other locations.

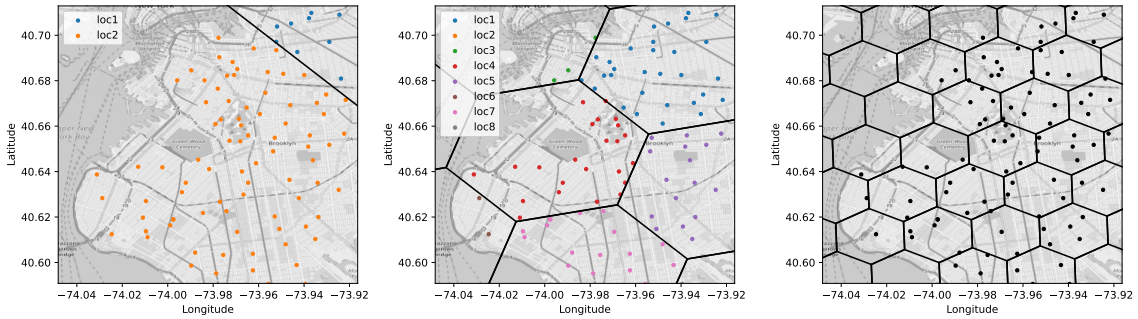
Generate				**Receive**			
Random Seed	**#TEK**	**#RPI Gen.**	**#Locations**	**#RPI Recv.**	**#Location Visited**	**#Matched RPI (GAEN$^+$)**	**#Matched RPI (GAEN)**
255	$1,400$	$201,600$	30	$6,048$	5	$1,046$	$6,048$
254	$1,400$	$201,600$	30	$6,048$	5	995	$6,048$
253	$1,400$	$201,600$	30	$6,048$	5	965	$6,048$
252	$1,400$	$201,600$	30	$6,048$	5	926	$6,048$
251	$1,400$	$201,600$	30	$6,048$	5	$1,044$	$6,048$

Table 6.1: Effective evaluation of GAEN$^+$ using H3 Resolution 10 compared with original GAEN.

Next, we assume $6,048$ (3%) of these $201,600$ generated RPIs are received by an honest user (i.e., a single replay attack victim), further restricting the user to have only visited 5 of the 30 locations. The user's smartphone then validates the captured RPIs using the $1,400$ TEKs and 5 locations to derive and check for matches between the received and computed RPIs. The H3 library is used to calculate the geospatial index of the locations with 10 as the resolution parameter. As shown in Table 6.1, we use 5 different seed values for RPI generation and location selection to provide randomness and enable evaluation of multiple, distinct simulations.

The results of RPI matching are shown in the last two columns of Table 6.1, showing GAEN$^+$ accurately confirming approximately $1,000$ ($1/6^{\text{th}}$ of the generated RPI set) in the cells for which the user recorded being present. However, GAEN inaccurately confirmed all $6,048$ RPIs, including those from locations which the user did not visit. Our results demonstrate the ability for GAEN$^+$ to accurately determine valid captured RPIs, further indicating its general effectiveness against geographically distributed replay attacks.

Impact of the Resolution Parameter. To clearly show how resolution affects geospatial indexing, we randomly selected 100 latitude and longitude points in Brooklyn, NY, and show their distribution and placement within H3 cells at resolution 4, 6, and 7, respectively in Figure 6.3a, Figure 6.3b, and Figure 6.3c. As shown, under resolution 4, 6, and 7, the 100 locations are respectively mapped to 2, 8, and 26 H3 hexagon-shaped cells.



(a) Resolution 4 (b) Resolution 6 (c) Resolution 7

Figure 6.3: Distribution of 100 H3-indexed latitude and longitude coordinates under different resolutions

To further evaluate the impact of resolution on GAEN$^+$, we conduct different sets of experiments with similar setup, shown in Table 6.1. And additionally, the impact decreasing resolution has on accuracy, from resolution 10 down through 4, as presented in Table 6.2.

For each experiment, we count the number of the RPIs that are correctly matched (i.e., the matched RPI originated from a location the receiver actually visited) and the RPIs mismatched (vice versa). As shown in the last two columns of Table 6.2, there are a great number of mismatched RPIs under relatively low resolutions 4, 5, and 6, due to the high frequency of H3 index collisions of two different locations. For instance, under resolution 4 (Figure 6.3a), location points that are relatively long distances between each other will still be mapped to the same H3 index, as the area of the hexagon is significantly larger. However, at higher resolutions, the chance of collision between these same points is significantly reduced and thus the number of

Res.	#Loc	#H3 Index	RPIs		
			#Total	#Matched	#Mismatched
4	100	2	6,048	76	5,101
5	100	3	6,048	76	3,755
6	100	8	6,048	145	3,316
7	100	26	6,048	207	898
8	100	86	6,048	274	134
9	100	97	6,048	340	0
10	100	99	6,048	340	0

Table 6.2: Impact of resolution parameter on GAEN$^+$.

mismatches becomes lower, converging to 0 at resolution 9. In summary, a higher resolution parameter in GAEN$^+$ offers finer-grain protection against replay attack, therefore capable of eliminating more false positives.

Time Complexity. Enhancements from GAEN$^+$ impact the run-time complexity for RPI derivation for both advertising and receiving. In GAEN, the complexity to derive an RPI for an advertiser is:

$$O(1)$$

In GAEN$^+$, a single smartphone's number of locations is upper-bounded by the number of discrete cells in the chosen H3 resolution. In practice, this is significantly smaller given a person's average daily schedule has them in similar locations for work, school, etc, and not travelling the whole Earth. Therefore producing:

$$O(|ResolutionCells|)$$

When deriving RPIs as a receiver of DKs, the time complexity for GAEN is similarly straight forward:

$$O(|ReceivedDKs| \times |CapturedRPIs|)$$

However, GAEN^{+} introduces another loop to derive LRPIKs for each geospatial index it stored in L. Again as an upper bound, every cell in a resolution must be considered but is not practical, resulting in complexity:

$$O(|ReceivedDKs| \times |ResolutionCells| \times |CapturedRPI|)$$

Yet again, in practice the set L is significantly smaller than *ResolutionCells*, so run-time is more realistically:

$$O(|ReceivedDKs| \times |L| \times |CapturedRPI|)$$

Chapter 7: Discussion

We now discuss the effectiveness of GAEN$^+$ against replay attacks, privacy implications of GAEN$^+$ and related works, and future work around accuracy of coarse location and optimizing the use of H3.

7.1 Effectiveness Against Replay Attack

GAEN$^+$ does not fully remove the ability for replay attacks to rebroadcast within the same cell a legitimate RPI was captured. For example, a static node could capture RPIs in an area of high traffic and immediately replay them for the duration of the timestamp interval. This effectively mimics a scenario where every individual broadcasting a valid RPI stays in place at the point of collection until the valid time window has elapsed, creating an opportunity for other users later passing by to capture a replayed RPI that was derived from the same geospatial index. Alternatively, the malicious replay network could reference the same geospatial index database and nodes could subscribe to only download RPIs from their current location's index. The countermeasures to this type of static and confined variant of a replay attack

include more dynamic use of higher H3 resolutions, thereby reducing the geospatial validity area, and more precise smartphone locations.

7.2 Privacy Preserving Assertion

Despite adding geospatial awareness, GAEN$^+$ does not diminish the privacy preserving aspects and requirements of GAEN. Introducing a geospatial component during RPIK derivation, GAEN$^+$ maintains a bi-layer of indirection because the location context is effectively hidden from those nearby receiving RPIs and the central server and users receiving DKs. Note that in order to derive a location from a TEK, one must also have an RPI derived from it, and vice versa.

From the central server's perspective, we do not include additional data in uploads, such as location or RPIs, since we do not modify this behavior from GAEN (a user still only generates one TEK per day and optionally uploads a limited selection upon diagnosis). Since the server does not have RPIs by regular means, there is no way to determine the location a TEK was generated.

There are two recipients to consider for captured RPIs: the honest user and the malicious actor. It is important to note that, by virtue of being able to capture an RPI from a nearby user, one can simply query their own location which is inherently shared by the nearby user. The honest user does not attempt to derive location and therefore is not a threat. Suppose the malicious actor has captured RPIs, receives DKs, and attempts to bruteforce the RPI derivation to determine the originating smartphone's location. For each TEK, this would require computing LRPIKs for

each geospatial index found in a resolution, then comparing each of the 144 derived RPIs to the set of captured RPIs. Considering resolution 10 from H3, this would require approximately 2^{42} values to compare per random TEK. Despite this being a reasonable safeguard, as noted, bruteforcing is a superfluous activity - the malicious actor can already query their own location, thus the nearby user's. Therefore, adding a location component to LRPIK derivation does not expose location context that a malicious actor couldn't already attain with easier, direct methods.

Geospatial index values are stored on a user's device in a secure way similar to captured RPIs, using the operating systems' underlying mechanisms for secure storage. A severe flaw in an operating system would need to be exploited in order for a malicious actor to retrieve these values, therefore out of scope. Further, these index values are intended to remain on device and not shared with any other user, application, or enabling server for the protocol.

GAEN, like many other protocols, suffers from an inherent ability for tech-savvy bad actors to complement RPIs with other captured information or indicators. For example, an individual can setup a video camera to record a crosswalk while capturing RPIs with a BLE receiver to pair with timestamps. If a user passing by is diagnosed COVID-19 positive and decides to upload a set of DKs that include the time of video capture, the bad actor could potentially calculate all the RPIs from the received DK pairs and match with complemented video data and identify with advanced image processing techniques. While our proposed modification does not

remedy this issue, it also does not further reveal any additional location data despite adding a location component as discussed above.

Responsible Disclosure. We reported our "low-cost" replay attacks against GAEN to Apple via `product-security@apple.com` and Google via `https://issuetracker.google.com/issues/new` on 10/14/2021. On 11/17/2021, Google stated they are investigating current documentation and considering publicly releasing details of our research, which will receive an Honorable Mention at `https://bughunters.google.com/leaderboard/honorable-mentions`.

7.3 Future Work & Recommendations

While GAEN$^+$ demonstrates ability to reduce the impact of geographically distributed replay attacks while preserving user privacy, there is still room for future improvement and areas for future research.

1. **Location Precision.** GAEN$^+$ utilizes coarse locations to adhere to GAEN's Additional Terms that forbid the use of precise location data. However, the two platforms offer various levels of precision that may vary substantially in practice (§5). In keeping with the approach of GAEN, it is recommended that Google and Apple collaborate on defining a consistent radius coarse location service specifically for contact tracing, which allows for optimized use with H3.

2. **H3 Resolutions.** Given the range of precision provided by the 16 resolutions of H3, future work can explore creative applications of dynamic resolutions that

adapt to population density and events. For example, if a smartphone detects it is inside a stadium, the resolution increases to provide more resilience against replay attacks.

3. **Nearest Neighbors.** While we chose H3 for its uniform consistency in distance to nearest neighbors, we do not evaluate its full potential in this paper, including calculating near-border encounters. This can be currently implemented using the H3 library, but would introduce increased computation requirements, with a worst-case 6x increase. However, additional research into optimized calculations and algorithms would further strengthen GAEN^{+}'s ability to efficiently and accurately account for this edge-case.

Chapter 8: Related Work

Having recognized the absence of geolocation context in GAEN, *ACTGuard* [14] recommends thwarting replay attacks by installing a third-party app that creates a hash of captured RPIs with respective time, location, and its own RPI. This hash is then uploaded to a third-party central server to later distribute to other users. However, compared to GAEN apps, the installation of an additional app for *ACTGuard* introduces an unnecessary adoption barrier and complexity. For each captured RPI at a new time and location, the user's smartphone must compute, store, and upload a one-way hash. Additionally, download a very large set of hashes and compare during matching against positive users' DKs. Without specific guidance on limiting the geospatial context, besides using precise GPS coordinates, this concept potentially creates massive amounts of hashes and network transfer burden.

Raskar et al [13] propose recommendations for adding global context to GAEN in an effort to address a broader set of inherent limitations of its decentralized approach, including reducing false-positives and—in contrast to the GAEN principle—to allow

Protocol	Location Context	No Modification to the Payload	Location Indirection	Location Absent in Payload	Replay Attack Defense	No Third-party Infrastructure	No Third-party App	No Additional Uploads
GAEN [10, 11]	✗	-	✗	✓	✗	✓	✓	✓
ACTGuard [14]	✓	✓	✗	✓	✓	✗	✗	✗
Raskar et al [13]	✓	✗	✗	✗	✓	✓	✓	✓
GAEN$^+$	✓	✓	✓	✓	✓	✓	✓	✓

Table 8.1: Comparison of our proposed GAEN$^+$ with other closely related works.

a user to recall the environment in which an exposure occurred. They propose inserting GPS/location data into the AEM field, which is subsequently encrypted with the AEMK, and broadcast as part of the advertisement payload. Upon a successful RPI match, the TEK used to derive the RPI will subsequently be used to also derive the AEMK. Since the encryption of the AEM field uses the AEMK as a symmetric key, the app can now decrypt the AEM field and retrieve the originating user's location.

While their recommendation begins to approach similar resiliency against geographically distributed replay attack as our GAEN$^+$, it does so by including location context directly in the broadcasted payload. This direct inclusion thus risks alarming users who may not understand cryptography and only focus on the idea their GPS coordinates are broadcasted for anyone to capture. This approach may further inhibit adoption, whereas GAEN$^+$ is completely location context-free in the protocol payload.

While both solutions recommend GPS, neither concretely provides a solution for representing geospatial location relative to other users. Given the limitations of their approaches, as illustrated in Table 8.1, we motivated the development of GAEN^{+}, whose payload location-free protocol is completely compatible with GAEN.

Chapter 9: Conclusion

Digital contact tracing offers significant potential to help reduce the spread of SARS-CoV-2 and other viruses. Low-cost, crowd-sourced, geographically distributed replay attack networks threaten the GAEN framework if deployed by malicious citizen or nation-state actors, replaying GAEN advertisements throughout the world. Such a scaled attack would greatly increase false-positive notifications and introduce cascading social effects, undermining the intent of GAEN as a public good. By adding geospatial awareness during an intermediate key derivation, enabled by coarse location coordinates and a hierarchical geospatial index such as H3, our GAEN$^+$ is a solution that demonstrates its ability to mitigate cross-region replay attacks while preserving user privacy.

We implemented a PoC of the distributed replay attack and a prototype of GAEN$^+$ on Android and evaluated it with a set of experiments. Our results show that GAEN$^+$ is able to effectively prevent distributed replay attacks with negligible additional overhead compared with the original GAEN framework.

Bibliography

[1] COVID-19 map - Johns Hopkins Coronavirus Resource Center. `https://coronavirus.jhu.edu/map.html`. (Accessed on 09/26/2021).

[2] Tianshi Li, Camille Cobb, Jackie (Junrui) Yang, Sagar Baviskar, Yuvraj Agarwal, Beibei Li, Lujo Bauer, and Jason I. Hong. What makes people install a covid-19 contact-tracing app? understanding the influence of app design and individual difference on contact-tracing app adoption intention. *Pervasive and Mobile Computing*, 75:101439, 2021.

[3] BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders. `https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf`. (Accessed on 09/26/2021).

[4] Sourabh Niyogi, James Petrie, Scott Leibrand, Jack Gallagher, Manu Eder Hamish, Zsombor Szabo, George Danezis, Ian Miers, Henry de Valence, and Daniel Reusche. TCNcoalition/TCN: Specification and reference implementation of the TCN protocol for decentralized, privacy-preserving contact tracing. `https://github.com/TCNCoalition/TCN`, April 2020. (Accessed on 04/23/2020).

[5] Documentation for pan-european privacy-preserving proximity tracing (PEPP-PT). `https://github.com/pepp-pt/pepp-pt-documentation`. (Accessed on 10/01/2021).

[6] Pietro Tedeschi, Spiridon Bakiras, and Roberto Di Pietro. SpreadMeNot: A provably secure and privacy-preserving contact tracing protocol. *arXiv preprint arXiv:2011.07306*, 2020.

[7] Priyanka Singh, Abhishek Singh, Gabriel Cojocaru, Praneeth Vepakomma, and Ramesh Raskar. PPContactTracing: A privacy-preserving contact tracing protocol for covid-19 pandemic. *arXiv preprint arXiv:2008.06648*, 2020.

[8] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

[9] Haohuang Wen, Qingchuan Zhao, Zhiqiang Lin, Dong Xuan, and Ness Shroff. A study of the privacy of covid-19 contact tracing apps. In *International Conference on Security and Privacy in Communication Networks*, 2020.

[10] Exposure notifications: Helping fight COVID-19 - google. `https://www.google.com/covid19/exposurenotifications/`. (Accessed on 10/02/2021).

[11] Privacy-preserving contact tracing - Apple and Google. `https://covid19.apple.com/contacttracing`. (Accessed on 09/27/2021).

[12] Mobile OS market share 2021. `https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems/`. (Accessed on 09/27/2021).

[13] Ramesh Raskar, Abhishek Singh, Sam Zimmerman, and Shrikant Kanaparti. Adding location and global context to the google/apple exposure notification bluetooth api. *arXiv preprint arXiv:2007.02317*, 2020.

[14] Marco Casagrande, Mauro Conti, and Eleonora Losiouk. Contact tracing made unrelayable, 2020.

[15] Doj announces seizure of domain behind Russian-backed botnet. `https://www.cnn.com/2018/05/24/politics/doj-botnet-takedown-malware-russia-hackers/`. (Accessed on 10/12/2021).

[16] Wifitrace: Network-based contact tracing for infectious diseases using passive wifi sensing. `https://arxiv.org/pdf/2005.12045.pdf`. (Accessed on 11/21/2021).

[17] Bluetooth technology overview — bluetooth® technology website. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. (Accessed on 11/07/2021).

[18] Qingchuan Zhao, Haohuang Wen, Zhiqiang Lin, Dong Xuan, and Ness Shroff. On the accuracy of measured proximity of bluetooth-based contact tracing apps. In *International Conference on Security and Privacy in Communication Networks*, 2020.

[19] Dpˆ3t · github. https://github.com/DP-3T. (Accessed on 11/07/2021).

[20] Opentrace · github. https://github.com/opentrace-community. (Accessed on 11/06/2021).

[21] Exposure notifications service additional terms. https://blog.google/documents/72/Exposure_Notifications_Service_Additional_Terms.pdf/. (Accessed on 10/09/2021).

[22] Exposure notification - cryptography specification.pages. https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf/. (Accessed on 09/26/2021).

[23] Exposure notification - Bluetooth specification.pages. https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf/. (Accessed on 09/26/2021).

[24] Privacy and security attacks on digital proximity tracing systems. https://github.com/DP-3T/documents/blob/master/Security%20analysis/Privacy%20and%20Security%20Attacks%20on%20Digital%20Proximity%20Tracing%20Systems.pdf. (Accessed on 10/01/2021).

[25] Jaap-Henk Hoepman. A critique of the google apple exposure notification (GAEN) framework. *arXiv preprint arXiv:2012.05097*, 2020.

[26] How bluetooth technology uses adaptive frequency hopping to overcome packet interference — bluetooth® technology website. https://www.bluetooth.com/blog/how-bluetooth-technology-uses-adaptive-frequency-hopping-\to-overcome-packet-interference/. (Accessed on 10/12/2021).

[27] Bluetooth addresses & privacy in bluetooth low energy - novel bits. https://www.novelbits.io/bluetooth-address-privacy-ble/. (Accessed on 10/12/2021).

[28] Corona-sniffer: Contact tracing BLE sniffer PoC. https://github.com/oseiskar/corona-sniffer. (Accessed on 09/30/2021).

[29] How to install apps from outside your phone's app store. https://www.wired.com/story/install-apps-outside-app-store-sideload/. (Accessed on 10/12/2021).

[30] The pernicious invisibility of zero-click mobile attacks. https://www.forbes.com/sites/forbestechcouncil/2020/12/14/the-pernicious-invisibility-of-zero-click-mobile-attacks/?sh=5dfe48433079. (Accessed on 10/13/2021).

[31] A new NSO zero-click attack evades Apple's iPhone security protections, says citizen lab. https://techcrunch.com/2021/08/24/nso-pegasus-bahrain-iphone-security/. (Accessed on 10/13/2021).

[32] Sneaky zero-click attacks are a hidden menace. https://www.wired.com/story/sneaky-zero-click-attacks-hidden-menace/. (Accessed on 10/13/2021).

[33] Smartphone GPS accuracy study in an urban environment. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0219890. (Accessed on 10/10/2021).

[34] How to protect yourself & others. https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/prevention.html. (Accessed on 10/10/2021).

[35] Accuracy of iPhone locations: A comparison of assisted GPS, WiFi and cellular positioning. https://www.globe.gov/documents/2631933/6965868/42534991.pdf. (Accessed on 10/10/2021).

[36] Using the significant-change location service — Apple developer documentation. https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_significant-change_location_service. (Accessed on 10/10/2021).

[37] Exposure notifications. https://www.aphl.org/programs/preparedness/Crisis-Management/COVID-19-Response/Pages/exposure-notifications.aspx. (Accessed on 09/26/2021).

[38] Geohashes - IBM documentation. https://www.ibm.com/docs/en/streams/4.3.0?topic=334-geohashes. (Accessed on 10/03/2021).

[39] S2 geometry. https://s2geometry.io/. (Accessed on 10/02/2021).

[40] H3. https://h3geo.org/. (Accessed on 10/02/2021).

[41] Christian Böhm. Space-filling curves for high-performance data mining. *arXiv preprint arXiv:2008.01684*, 2020.

[42] Mercator projection — definition, uses, & limitations — britannica. https://www.britannica.com/science/Mercator-projection. (Accessed on 10/12/2021).

[43] Location data maps SDK for android google developers. https://developers.google.com/maps/documentation/android-sdk/location. (Accessed on 09/28/2021).

[44] What is the distance of one city block? https://www.reference.com/science/distance-one-city-block-532a6843f5039b79. (Accessed on 10/10/2021).

[45] Bindings — H3. https://h3geo.org/docs/community/bindings. (Accessed on 10/03/2021).

[46] Average app file size: Data for Android and iOS mobile apps. https://sweetpricing.com/blog/2017/02/average-app-file-size/. (Accessed on 10/03/2021).

[47] Table of cell areas for H3 resolutions. https://h3geo.org/docs/core-library/restable/. (Accessed on 09/26/2021).

[48] DigitalOcean pricing. https://www.digitalocean.com/pricing/. (Accessed on 10/03/2021).

[49] Exposure notifications API. https://github.com/google/exposure-notifications-internals. (Accessed on 10/12/2021).