Reconciling Security and Utility in Next-Generation Epidemic Risk Mitigation Systems

Paper # 3

ABSTRACT

We present Silmarillion, an inclusive, secure and privacy-preserving system that aids pandemic management by facilitating digital contact tracing and exposure risk notification as well as epidemiological analysis. Unlike today's contact tracing systems, which rely on peer-to-peer (P2P) encounters between smartphones, Silmarillion relies on peer-to-infrastructure (P2I) encounters between user devices and beacons installed in well-known locations – a conceptual idea that has been proposed before but never implemented or analyzed in detail so far. Unlike P2P systems, P2I systems can enrich encounter histories with geo-location and environment conditions while maintaining user privacy at a level comparable to manual contact tracing. This enriched information enables detailed scientific analysis of disease parameters and reduces false positives in exposure risk notification.

We describe the design of Silmarillion and its communication protocols that ensure user privacy and data security. We also evaluate a prototype of Silmarillion built using low-end IoT boards, showing that the power consumption and user latencies are adequately low for a practical deployment. Finally, we briefly report on a small-scale deployment within a university building as a proof-of-concept.

1. INTRODUCTION

Containing infectious diseases such as the recent COVID-19 pandemic requires contact tracing, testing, and isolation as well as epidemiological analysis. Epidemiological analysis can help understand conditions of infection propagation, while contact tracing enables reaching out to at-risk individuals, so that they can test and isolate.

Digital systems have been developed to scale contact tracing and provide personalized risk notifications. Currently deployed systems, generically called SPECTS [19] (Smartphone-based Pairwise Encounter-based Contact Tracing Systems), rely on pairwise bluetooth encounters between users' smartphones to record contact and a central backend to mediate risk dissemination. Current SPECTS are broadly grouped into two types: centralized and decentralized. In *centralized* SPECTS [3, 4, 9, 21], the backend learns about encounters between *all pairs* of users, thus raising privacy concerns for users. *Decentralized* SPECTS [8, 11, 22, 27], by contrast, enable risk estimation locally on users' devices. Thus, the backend only learns of encounters between individuals who have tested sick, which is no worse for user privacy than manual contact tracing. Both centralized and decentralized SPECTS have been deployed with varying adoption and success rates during COVID-19 [38].

To minimize user-privacy concerns, both centralized and decentralized SPECTS deliberately avoid collecting physical locations and environmental conditions alongside encounters. However, this information is sought by epidemiologists, as it can improve understanding of physical factors (temperature, humidity) that promote disease transmission [34] and justify policy decisions. It can also support more accurate personalized risk score estimation, reduce false positive rates in contact tracing, and enable better use of limited test resources [18].

Partly motivated by these shortcomings of SPECTS, prior work like PanCast [19] and Lighthouses [37] proposed an alternative design based on P2I (person-to-infrastructure) encounters in place of the person-to-person encounters of SPECTS. In these systems, beacons are installed in strategic places where people tend to congregate (e.g., classrooms, markets, and theatres). Each beacon broadcasts (on short-range BLE radio) identifiers that are unique to the beacon and the current time (time is roughly quantized). Personal devices of nearby users record these identifiers – in particular, two users in the vicinity of same beacon at similar times will record the same identifiers. Contact tracing then works as in decentralized SPECTS. When a user tests sick, the identifiers on their device from their period of contagion are collected at a backend and disseminated to everyone. Other users match these disseminated identifiers to those stored on their own devices, and assess their infection risk locally.

Going beyond SPECTS, P2I systems can tag beacons' identifiers with location and environment information. Thus, sick individuals' devices provide a rich dataset for various epidemiological analyses in the backend. Extensive simulations on human mobility have also demonstrated other benefits of this enriched information, such

as improved risk scores for users and fewer false alarms, even with a partial P2I rollout where beacons are installed only in a few strategically selected locations [19].

Notably, *this collection of location and environment information does not raise any new privacy concerns* since the collected data only reveals (to the backend) the trajectories and mutual encounters of sick individuals during their period of contagion – information that is collected by health authorities even for manual tracing.

Despite their benefits, *P2I systems have only been proposed in principle and justified through simulations so far* – they have never been built, and the protocols needed for communication between beacons, user devices, and owners have never been explained in detail, nor examined for security, privacy and scalability.

Our work. Our paper fills this gap through two contributions. First, we present the detailed design of Silmarillion, a P2I system that implements contact tracing and provides location/environment-tagged encounters for epidemiological analysis without raising privacy concerns beyond those of manual tracing (§2-§5).

Second, we build (§6) and evaluate (§7) a prototype of Silmarillion with low-end IoT devices for beacons and users. In particular, user devices with a simple UI and low costs (e.g., keyfobs) suffice for participating in the Silmarillion system for most times; thus allowing economically or technologically challenged individuals in the system. We demonstrate that an inclusive solution can be deployed at low cost, while achieving scalability, security, and energy efficiency for the devices.

To the best of our knowledge, Silmarillion is the first epidemic risk mitigation system based on P2I encounters that has actually been implemented and evaluated. Like other P2I systems, Silmarillion can be deployed incrementally by placing beacons in locations of primary interest first (§8).

2. OVERVIEW

Fig. 1 shows an overview of Silmarillion's architecture and workflow. Silmarillion consists of two types of beacons (BLE-only and network beacons), personal devices (dongles)[1], terminals, and a backend platform that relays risk notifications and aggregates data for epidemiological analysis.

(1) The beacons, which are placed in strategic locations (e.g., shops, restaurants), continuously broadcast cryptographically-generated random strings called *ephemeral ids*. The dongles listen to these beacons passively (i.e., without transmitting anything) and store the beacons' ephemeral ids. When an individual tests positive for the infectious disease, they may be legally required to or may choose to disclose (a selected subset of) the list of ephemeral ids stored in their dongle to the backend. (2) The individual must explicitly authorize the transmission of data to the backend from their dongle via a trusted terminal. The backend periodically (e.g., daily) assimilates the information about which locations were contaminated at which times (which depends on users' visits and location features) into a risk database. (3) The risk information is then disseminated back to all users whose dongles estimate their exposure risk locally. Finally, dongles download the risk information via the network beacons (or trusted terminals), compare the ephemeral ids in their storage against those in the risk information disseminated, and notify their owners in case of non-zero matches. Silmarillion supports two mechanisms for risk dissemination: (4) a broadcast of the risk information of local neighborhoods via network beacons in the neighborhoods, and (5) an active querying mechanism whereby dongles query the backend for the risk information of specific regions via a network beacon (or terminal).

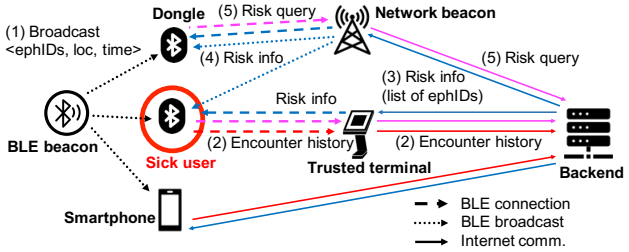


Figure 1: Silmarillion's architecture.

We now provide an overview of Silmarillion's components (§2.1), properties (§2.2), and threat model (§2.3).

2.1 Components

Beacons. There are two types of beacons. A majority of the beacons are commodity, battery-operated *BLE-only beacons*. A smaller number of beacons—called *network beacons*—also use BLE, but additionally require mains power and a network connection to be able to disseminate risk information to user dongles. Beacons may be installed by health authorities or individual organizations in classrooms, restaurants, or even mobile locations, e.g., a city bus or a train. Network beacons may be installed, e.g., next to a WiFi router.

All beacons have a coarse-grained timer and a small flash storage. The beacons are registered with the backend using an id, a secret key, a location id comprising their stationary coordinate or a route id, and a description about their location that may be epidemiologically relevant (e.g., humidity, temperature). The descriptor may be configured statically or updated using on-board

[1]In the rest of this document, we focus on personal devices in the form of dongles, unless stated otherwise. For smartphone users, Silmarillion also provides an app; the app and its data could be secured using encryption and isolation primitives, such as TEEs [23, 26].

sensors, and can be used by the backend for intelligent risk estimation and/or epidemiological analysis.

Dongles. Dongles are small, simple devices that users can attach to a keyring, or wear on the wrist or around the neck. They operate off of a coin battery and have a coarse-grained timer, a counter, a small amount of flash storage, a minimal user interface in the form of a LED that indicates risk status and battery condition, and a button to control the LED notification.

Similar to beacons, dongles are registered and authenticated with the backend and receive a secret key from the backend. In addition, the users configure a password in the dongle, which they use to authenticate themselves to the dongle. The password and the counter are also used to control upload of data from the dongle.

Backend. The backend, which may be managed by a health authority or an independent entity, maintains several databases. (i) $\texttt{DB}_{\texttt{Beacon}}$ contains each registered beacon's location/trajectory and the secret key used by the beacon to generate its unique sequence of ephemeral ids. (ii) $\texttt{DB}_{\texttt{User}}$ contains registered users, their dongles, and the cryptographic keys required to authenticate each dongle. (iii) $\texttt{DB}_{\texttt{Risk}}$ contains the uploaded encounter histories of tested individuals.

The $\texttt{DB}_{\texttt{Risk}}$ is available to health authorities for risk dissemination and for analytics, e.g., to identify hotspots and superspreading events, and to estimate epidemiological parameters. Additionally, the backend consists of two non-colluding servers that serve user queries, in an IT-PIR framework, for risk information of specific regions in a privacy-preserving manner.

Trusted terminals. Terminals are provided at locations that issue dongles and health care facilities that do testing. Terminals allow users to connect to their dongles over BLE to inspect what data is recorded on their dongles, upload data to the backend and, when allowed or required by law, decide what subset of the recorded information they wish to upload when they are diagnosed, and receive risk information from the backend. Users can also use personal computers or smartphones as terminals to perform these tasks, or use the smartphone of a care provider who visits their home.

2.2 Silmarillion properties

Utility. Silmarillion provides high utility as follows. First, it relies on low-cost infrastructure that can be easily deployed anywhere, and facilitates participation of technology-challenged, economically disadvantaged, or physically challenged individuals who cannot or do not use smartphones. Second, users' devices passively collect encounter data in the background, and require user interaction only to upload encounter history when sick and to download risk data from the backend at low frequency (e.g., once a day). Third, the beacons' ephemeral ids, which are repeated for a short duration, help identify non-contemporaneous transmissions. For example, a user entering an elevator shortly after an infected individual leaves the elevator can be exposed and thus could receive a notification with Silmarillion. Fourth, the location and other contextual information recorded in encounters facilitates accurate risk notification to users, as well as epidemic analytics. For instance, the fine-grained location and time information can help in identifying the peak time of superspreading events or an emerging infection hotspot, and rapidly contacting the affected crowds. It can help in determining ambient information (e.g., air quality, humidity) even without deployment of extra sensors. Furthermore, the location-time and contextual data from multiple hotspots can be compared to refine epidemiological models.

Efficiency. Silmarillion uses a hybrid protocol, combining Bluetooth broadcasts and PIR-based querying to the backend, to enable dissemination of only relevant risk information with modest latency, bandwidth, compute, and power costs (see details in §4.3).

Security. Silmarillion secures data and protects user privacy under the threat model described in §2.3. We further analyze Silmarillion's security in §5.

2.3 Threat model

Silmarillion seeks to provide user privacy at a level comparable to manual tracing. Admissible threats are from network observers and attacks (e.g., malware, exploits) that compromise subsets of user dongles and beacons.

Silmarillion assumes that the entity that deploys the system (typically, a local authority or a government) and, hence, the backend, are trusted. Indeed, if the deploying authority is malicious and wishes to covertly spy on people, it does not need Silmarillion; it can simply install cameras, Bluetooth sniffers and other recording devices, or coerce users to reveal their whereabouts. Nonetheless, as an added precaution against *partial compromise* of the backend, our active risk query protocol is based on private information retrieval (PIR); it divides trust between two backend servers, and hides user information from each of them individually. A real deployment of Silmarillion may further secure the backend against partial compromise, even of hardware, using standard techniques like secure multi-party computation and hardware enclaves (TEEs), as in CoVault [30].

We discuss specific assumptions about various components below.

Beacons and dongles. In the normal use of the system, ephemeral ids recorded by all individuals who test sick are broadcast. An adversary may record these broadcasts and combine them with ephemeral ids they have recorded on dongles they control, to infer places that sick people visited. We assume realistically that an attacker does not have the ability to make large-scale observations in the real world (indeed, an attacker with

this ability does not need Silmarillion to violate others' privacy). Silmarillion uses this assumption and differentially private noise in broadcasts to mitigate information-combining attacks.

Further, an adversary may actively compromise a subset of dongles. This endangers the privacy of the compromised users and their encounter peers only as the adversary can use the ephemeral ids stored on the compromised dongles to determine only pairwise encounters among them, and encounters between them and other uncompromised users who test sick (whose ephemeral ids end up being broadcast). In practice, encrypting the dongle storage can prevent these attacks.

An adversary may also compromise a subset of beacons and relay or replay ephemeral ids of the compromised beacons to generate false positives in the system. Silmarillion's backend can leverage the encounter data uploaded by sick users to detect inconsistent beacon broadcasts and initiate repair for the compromised beacons. We elaborate on this in a tech report [14].

Terminals/Network beacons. Terminals are trusted with a user's password and the encounter history from their dongle only when the user uploads the encounter history through the terminal. All communication between the dongles and the backend is otherwise encrypted and MAC'ed end-to-end, so terminals (and network beacons) are not trusted in any additional way.

Side channels. Physical side channels (e.g., EM, power), which could be exploited to steal devices' crypto keys, are out of scope. In practice, devices could implement constant-time crypto [15] to mitigate these attacks.

3. DATA COLLECTION AND PROCESSING

Digital contact tracing consists of two parts: collecting contact data and disseminating risk information based on the data. In this section, we explain how Silmarillion realizes data collection. In §4, we discuss the risk dissemination mechanism.

3.1 Initial configuration

Each beacon is configured with a unique beacon id b, the backend's public key, a location id loc_b and a descriptor $desc_b$ corresponding to the location where the beacon is supposed to be installed, a initial clock C_b synced to real time, and a secret key sk_b from the backend that is known only to the beacon and the backend.

Similar to beacons, each dongle is configured with a unique id d, the backend's public key, a initial clock C_d synced to real time, a secret key sk_d and a monotonic counter ctr from the backend, and a password from the user. The dongle's secret key and initial counter value are known only to the dongle and the backend and are used to mutually authenticate and establish a secure channel between the two, when required. The password is used to mutually authenticate the owner/terminal and the dongle whenever the owner interacts with the dongle (e.g., to initiate upload to the backend as in §3.3).

The dongle and beacon configurations are registered with the backend, which stores this data in a device database in the form: {*device type*, *device id*, *secret key*, *initial clock*, *clock offset*, *location id*, *desc*}, where the location id and desc are set only for beacons. The clock offset in the backend is used to track any divergence between the real time and the local timer of a device known to the backend. It is initialized to 0 during device registration. The clock offsets for a dongle d and a beacon b are denoted δ_d and δ_b, respectively[2].

3.2 Capturing beacon encounters

Each dongle and beacon has a coarse-grained timer of 1-minute resolution (t_d and t_b respectively), which is set to the initial clock value provided by the backend, and subsequently increments every minute. A device stores its timer value to local storage at intervals of fixed length L, called epochs. A variable tracks the epoch id or the number of epochs elapsed since the device's start (i_d and i_b for the dongle and beacon, respectively). In practice, we suggest epoch length $L = 15$ minutes.

A beacon generates a new ephemeral id every epoch. In the i^{th} epoch i_b, the beacon b generates an id $eph_{b,i} = hash(sk_b, loc_b, i_b)$. Here $i_b = \lfloor (t_b - C_b)/L \rfloor$ and $hash$ is a one-way hash function. The beacon broadcasts $\{eph_{b,i}, b, loc_b, desc_b, t_b\}$, which is captured by nearby user dongles. Beacons broadcast each ephemeral id several times within an epoch. Dongles persist a single entry for each unique ephemeral id along with the first beacon and dongle timestamps at which the id was received (t^b_{start}, t^d_{start}), the duration for which the id was observed (t^b_{int}, t^d_{int}), and the average of the RSSI values observed ($rssi$). Thus, a log entry $enctr$ in dongle d would be: $\{eph_{b,i}, b, loc_b, desc_b, t^b_{start}, t^b_{int}, t^d_{start}, t^d_{int}, rssi\}$.

Datastructure configurations. The byte size of each field in an $enctr$ is as follows: $eph_{b,i}$: 15, b: 4, loc_b: 4, $desc_b$: 4, t^b_{start}: 4, t^d_{start}: 4, t^b_{int}: 1, t^d_{int}: 1, $rssi$: 1. The $eph_{b,i}$ is generated by computing a SHA-256 hash of the inputs and taking the least significant 15 bytes of the result. Thus, each beacon broadcast is 31 bytes and fits in a single BLE advertisement. Each stored dongle encounter is 38 bytes. Assuming that users encounter on average no more than one unique beacon ephemeral id every 10 min in a day, dongles need to store data for 2016 encounters in a 14-day window (the infectious period for the COVID-19 disease as determined by health experts), which requires ∼79KB of persistent storage in a dongle. In reality, a dongle is unlikely to encounter a distinct beacon ephemeral id every 10 min continuously for 14 days, so this estimate is conservative.

The 4-byte location ids are organized to tile the earth's

[2]For ease of exposition, we assume that the initial clock is set to zero in all devices at the same time, but.

surface area hierarchically. Specifically, we consider a 3-level tiling: the lowest level tiles (L-tile) correspond to 1x1 sq.km. areas, the medium level tiles (M-tile) cover the L-tile within a 100x100 sq.km., and the highest level tiles (H-tile) cover the M-tile within 6° longitude x 9° latitude areas. The chosen tiling leaves sufficient unmapped IDs, which we use for labeling mobile beacons in each region. The hierarchical location ids enable efficient risk dissemination, as we elaborate in §4.3.

3.3 Encounter data upload

Diagnosed individuals may share their encounter data with health authorities to enable dissemination of risk information to people who might have been in their vicinity. Because dongles only have Bluetooth connectivity and a minimal user interface, users fundamentally need access to a separate device through which they can authorize their dongle to upload data to the backend in case of a positive test. This device, called a terminal, can be any device with a GUI, BLE, and Internet support. The terminal can be part of a kiosk installed in a test center, clinic, or doctor's office, or a personal device (e.g., a phone) owned by the user or a care provider.

Upload overview. Users typically visit a test center or a clinic for testing and receive their results offline via email or phone. If the result is positive, they may wish to or be required by law to upload their data. We present different mechanisms that enable users with different levels of technological access to upload their data. The mechanisms vary in the time when data is uploaded and when it is actually released to the backend. In all cases, patients identify themselves at the time a test is taken using the normal procedures in place for this purpose. Normally, their contact details are recorded along with the id of their dongle and the test kit used for them. Once the test results are available, the user is informed using their contact details. If the result is positive, the notification includes a certificate indicating that the patient has tested positive on the given date, which the user can forward to the backend. The certificate includes the ids of the patient's dongle and test kit only.

Delayed release. A user initiates the data upload after they receive a positive test result. They establish a secure connection to their dongle via the terminal. For this, the user enters their dongle's id and their password for the dongle into the terminal, which then runs an authenticated Diffie-Hellman (DH) key exchange protocol with the dongle and establishes a secure connection. Once the connection is established, the user may enable the upload and attach their test certificate. The dongle initiates a BLE connection to the terminal, which in turn connects to the backend over HTTP. The dongle then establishes another DH session with the backend, encrypts its data and test certificate with the session key, and uploads them to the backend via the terminal.

Early release. Alternatively, users may choose to upload their data into an escrow at the time of their testing, pending a positive result. The user establishes a secure connection to their dongle using a terminal at the testing site using the same DH protocol described above, and consents to contributing their data if the test result turns out positive. The user sends a random one-time password (OTP) string via the terminal to their dongle, which then encrypts the data with a key derived from this OTP and immediately uploads the data to the backend. When the result is positive, the user reveals the OTP key to the backend directly or via the testing site, which uploads it along with the certificate.

Selective upload. Where it is legally allowed, users may choose to select what parts of their history they wish to share with the backend. For this, they first transfer their dongle's history to a trusted terminal, select the entries they wish to upload at the terminal and then send a command to the dongle via the terminal to upload only the selected encounters.

Once the backend receives a user's encryption key, it decrypts their dongle's encounter entries and adds them to $\texttt{DB}_{\texttt{Risk}}$. We provide protocols for upload mechanisms in our extended tech report [14].

4. RISK DISSEMINATION

We start with an overview of the risk information structure and the risk notification mechanism in the dongles. The risk information consists of a list of ephemeral ids. The ephemeral id of a beacon b for epoch i is included in the list only if a diagnosed individual encountered b in epoch i. For accurate risk estimation, the risk information may contain additional encounter parameters, e.g., rssi, encounter duration, beacon's descriptor, and weights for the beacon descriptors.

If a dongle has previously recorded any of the ephemeral ids listed in the risk information, its owner may have been exposed to a diagnosed individual. The dongle computes a risk score based on the number of matched ephemeral ids and (optionally) other features of the matched encounters. A user presses a button on the dongle and, if the risk exceeds a certain threshold, the dongle notifies the user via a LED so they can self-isolate and get tested.

We now discuss how the risk information is disseminated from the backend to dongles with the help of network beacons (or terminals). We assume that most users check their risk status once a day on average. We start with the requirements that the risk dissemination protocol needs to satisfy and then describe how Silmarillion satisfies each of the requirements.

4.1 Requirements

The risk dissemination protocol needs to address four requirements, which we discuss in this section. **D1.** the

information disseminated must be correct, **D2.** the protocol must preserve the privacy of the diagnosed patients whose information is being disseminated (§4.2), **D3.** the protocol must maintain privacy of the users seeking the risk information (§4.3), and **D4.** the relevant information must reach potentially affected users in a timely manner and with low bandwidth, power, and computational costs for dongles (see §4.3). Note that all risk information is signed by the backend to allow detection of any tampering by intermediate network beacons, which addresses D1.

4.2 Noising the risk dissemination

We present two scenarios where the *number of entries* in the risk broadcast could potentially reveal an individual's movements or health status to an adversary in the locality of the individual. We then describe our solution to mitigate such leaks, addressing requirement D2.

(i) Movements of diagnosed individuals. Suppose Alice learns (from the local news) that there was only one case of infection in the past few days within some geographic region. Separately, she learns that Bob was diagnosed and that he agreed to upload his encounter history when he got diagnosed. Alice can infer if Bob was near any beacon in the region while he was contagious based on whether she receives risk information for the region or not. Thus, the length of the risk information (zero vs. non-zero) reveals to an adversary information about the movements of a diagnosed individual.

(ii) Health status of an individual. Suppose Alice lives in an area with few people, say n, and Alice is able to track the movements of $n-1$ of these people through outside channels. If Alice receives risk information with more ephemeral ids than can be accounted for by the movements of the $n-1$ people she is tracking, she knows that the nth person (whom she is not tracking) must be sick as well. Even though such an attack requires a significant amount of offline information and may be difficult in practice, it does raise privacy concerns.

Note that these leaks rely solely on the *number of ephemeral ids in a risk notification* and arise without the adversary having even encountered an individual. We mitigate these leaks by adding noise to the risk information to hide the actual number of ephemeral ids. We add junk ids that do not correspond to any real beacon and thus do not match the history of any user dongle. Given our threat model, no adversary can monitor the ephemeral ids from a significant fraction of beacons and thus distinguish the junk ids from legitimate ids. The number of junk ids satisfy differential privacy (DP):

We adapt a mechanism proposed in prior work [17]. Given a risk broadcast, we add N junk ids to it, where $N = t + \lfloor \tilde{X} \rfloor$ is always non-negative; t is a natural number, and $\tilde{X}$ is a random value sampled from a Laplacian distribution with mean 0 and parameter λ *truncated* to the interval $[-t, \infty)$. The values of t and λ depend on the privacy required. To get (ϵ, δ)-DP, we pick $\lambda = A/\epsilon$ and $t = \lceil \lambda \cdot \ln\left((e^{(A/\lambda)} - 1 + \delta)/2\delta\right) \rceil$. Here, A is the sensitivity of the risk broadcast function; it equals the maximum number of risk entries that could be contributed by a *single* diagnosed individual, which we conservatively set to 2016 (§3.2). For $\epsilon = 0.1$ and $\delta = 0.01$, the 99th %ile noise required is 115991. We prove that our mechanism is (ϵ, δ)-DP in an extended tech report [14].

4.3 Dissemination protocol

A key requirement for Silmarillion is to ensure that users can receive risk information without revealing their own encounter history. A naïve way to satisfy this requirement would be to broadcast risk information of the entire world to users and let the users' devices filter the data for relevant matches. However, the size of the risk information payload can be several hundreds of MBs or even GBs, particularly during times of rapid infection spread. Thus, a broadcast mechanism would generate a significant bandwidth pressure on both the infrastructure and the end users. Moreover, it would incur high latency, power, and computational costs for the dongles. Silmarillion balances between requirements D3-D4 by using two protocols: broadcast and querying.

4.3.1 *Broadcast protocol.*

By default, network beacons download risk information of their *neighborhood* from the backend and then broadcast it over BLE's periodic broadcast channel. The neighborhoods are defined statically, e.g., as L-tile tiles (see §3.2). Each day, the backend encodes the set of $\texttt{DB}_{\texttt{Risk}}$ entries within a L-tile into a cuckoo filter (CF) [32] and sends the filter to the network beacons in the corresponding neighborhood. The CF encoding ensures that users can only check for presence of specific ephemeral ids but not learn all the ids in the risk payload, thus slowing down ephemeral id harvesting by colluding users.

When a dongle is in the proximity of a network beacon, it syncs to the beacon's broadcast channel and downloads the broadcast information. Note, BLE transmitters do not learn about devices syncing to their broadcast channels. Thus, users can receive risk information about their local region without revealing their presence or history to network observers or the backend.

4.3.2 *Querying protocol.*

While the neighborhood broadcast may be sufficient for many users, traveling individuals may require risk information of other regions they have recently visited. We use an IT-PIR protocol that allows users to query the backend for risk information without revealing the regions they have visited or are interested in.

The backend groups the $\texttt{DB}_{\texttt{Risk}}$ entries into blocks of fixed size. Unlike the broadcast protocol, grouping by

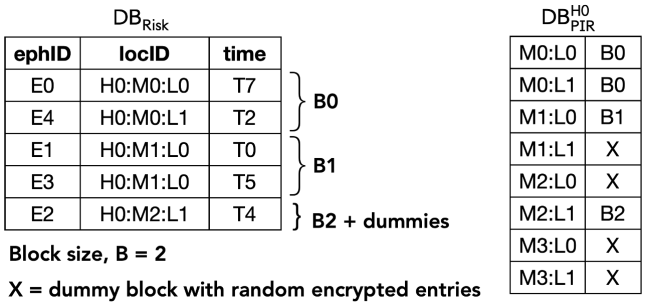


Figure 2: PIR data in the backend on a given day.

the geographical tiling alone may not be efficient, since the distribution of risk entries may vary between regions at different times. Therefore, the backend dynamically adjusts the grouping of $\texttt{DB}_{\texttt{Risk}}$ entries based on the prevailing infection rate distribution, while maintaining a uniform block size B.

A block corresponds to a tile at a certain level if the total number of risk entries for all tiles below it is $\leq B$. When the number of entries overflows B, the entries are split into blocks for all tiles at the next lower level. To ensure that a user does not learn the actual number of entries within a block, dummy entries are added to each block following the DP mechanism of §4.2. The entries are also encoded into a CF as in the broadcast protocol. Next, the backend maintains a database, $\texttt{DB}_{\texttt{PIR}}$, indexed by the L-tile id and pointing to the block containing the risk entries of L-tile. Fig. 2 shows the datastructures.

PIR protocol. We use a PIR protocol based on [29]. Our scheme relies on two servers, S_1 and S_2, each of which has a complete copy of the $\texttt{DB}_{\texttt{PIR}}$ D. We make the standard assumption that at least one server is non-malicious and non-colluding. Suppose D contains N blocks, each of which corresponds to a L-tile, and let the size of the largest block in D be B bits. We assume that all blocks are padded with dummy entries up to size B.

To query for the n-th block in D, a dongle generates two secrets shares Q_1, Q_2, which are random bit strings of length N with same bits except the n-th bit, which is flipped. The dongle sends Q_j to S_j; $j \in \{1, 2\}$.

Each server expands its query share from a vector of 1-bit entries to a vector of B-bits entries by replicating the bit at each index in the original query B times. Next, each server S_j returns a response $A_j = \bigoplus_{k=0}^{N}(Q_j[k] \cdot D[k])$ for $j \in \{1, 2\}$. The dongle XOR's the responses to retrieve $D[n]$.

We assume separate $\texttt{DB}_{\texttt{PIR}}$ instances for each region corresponding to a H-tile and dongles can query one L-tile at a time. Thus, if location IDs consist of H, M, L bits for identifying H-, M-, and L-tiles respectively, the dongle's complexity for generating a query-pair and uploading the queries, and each backend's computation complexity are $O(2^{(M+L)})$ each. The cost for receiving the response shares and retrieving a block is $O(B)$ each.

For querying, the dongle sets up an end-to-end secure session with each backend PIR server via a network beacon (or terminal), as is done for encounter history upload. The dongle then issues PIR queries for each unique L-tile covering the logged encounters.

User's privacy is protected from the servers, since each server receives only a share of the query and iterates over the entire DB regardless of the query, and only the user can recover the DB block from the response shares of the servers. All queries and results are encrypted end-to-end between the clients and servers to prevent leaks to eavesdroppers. Furthermore, the dongle generates fake queries using DP (similar to §4.2) to hide the actual number of queries issued in each round of risk notification (about once a day). The only remaining leak is through the timing and number of risk notification rounds.

Summary. Silmarillion ensures strong privacy for users receiving risk information. In the broadcast protocol, users receive information without revealing their presence or encounter history to network beacons, the backend, or other users. In the querying protocol, end-to-end encryption protects against network adversaries, while PIR hides from the backend which risk information a dongle is requesting, and thereby the user's recent whereabouts. Silmarillion also minimizes bandwidth overheads for users as they receive risk information only of their regions of interest. For diagnosed individuals, the DP mechanism protects the number of encounter entries uploaded by them, while the cuckoo filter encoding of risk information prevents others from easily learning the actual entries. Cuckoo filters also reduce bandwidth overheads, albeit at the cost of a small percentage of false positives. For instance, a cuckoo filter with entries of size 32 bits (as opposed to the 15-byte ephemeral ids) reduces risk payload sizes by ∼3.75x while incurring <0.01% of false positives for a 14-day period.

5. SECURITY ANALYSIS

We discuss Silmarillion's privacy properties in this section. Due to space constraints, we defer discussion about handling failures to an extended report [14].

Table 1 summarizes the information revealed to different principals in the system. Note that BLE beacons learn nothing about nearby users since they only transmit information unidirectionally. Similarly, no information is leaked during ephemeral id broadcast, since dongles only record information at this stage.

Leaks to the backend. As stated in our threat model, the backend is generally trusted. It can be further secured using standard hardware and cryptographic techniques, e.g., CoVault [30]. Even in case of a compromise, the information that can be leaked from the backend is

P	Registration	Encounter upload	Risk broadcast	Risk query
B	PII (e.g., phone#), user--dongle map, beacon-loc. map, devices' secret keys	patient's identity, loc. history, time of upload, terminal's IP, social graph	total risk info size	Q's time, # of queries, network beacon IP
N	none	none	broadcast size (DP)	Q's time, # of queries
T	none	P's password, time, size of upload, loc. history*	broadcast size (DP)	Q's time, # of queries
U	own data	–	common subset of patients' loc. history	common subset of patients' loc. history

Table 1: Information disclosed to each entity during Silmarillion's operation. Notes: *Only if user wishes to select data to upload. B: Backend, N: Network beacons, T: Terminal, U: Other user (healthy)

small. We discuss the leakage and potential mitigations.

The backend may leak the information provided by users at the time of dongle registration. For Sybil mitigation, the backend may have used personally identifiable information (PII), such as an email address, phone number, or another id to identify each registered user and their associated dongle. This PII could be leaked in case of a compromise. To minimize privacy concerns, the backend could instead rely on a pseudonym for identifying each registered user and the associated dongle.

A compromised backend may also learn the IP address of a dongle that is uploading data or making a query. To hide this, traffic from dongles can be routed through an anonymizing network, such as Tor [16] or a mixnet [28].

The backend learns the id of the dongle whose owner is sick when the dongle uploads its encounter history and the associated test certificate. It also learns a subset of the owner's recent whereabouts approved by the owner, at the granularity of beacon visits and epochs. Finally, it can also infer an over-approximation of the possible social contacts among sick users at the granularity of beacon visits and epochs. However, all this information is no more than that collected with manual tracing and, hence, raises no further privacy concern.

As long as one of the PIR servers is non-malicious, the backend learns no information about healthy users except the times when they query for risk data.

Leaks to network beacons/terminals. As long as a dongle listens to risk broadcasts passively, network beacons (terminals) cannot learn about the dongle's presence or identity. If a dongle queries the backend, a network beacon (or terminal) can observe the time and number of queries, but not the content of the communication, which is encrypted between the dongle and the backend.

When a dongle uploads data to the backend via a terminal, the terminal learns the user's password. It can observe the size of the upload, which could reveal the number of beacons a dongle has recently visited, but not the exact content of the upload. Uploads can be padded to obfuscate to the size information. If the user uses the terminal to select records prior to upload, then the terminal learns the part of the user's history that is stored on the dongle at the time of the selection. Hence, a user must use a trusted terminal for this purpose, e.g., their own computer or smartphone.

Leaks to other users. User dongles learn nothing about other users except through the risk notifications, and the only information they can learn is that which is uploaded by diagnosed users. Thus, it is impossible to learn anything about healthy users who never share any information with the system. A user learns the ephemeral ids it shares with sick individuals. Combining with auxiliary information (e.g., from local news), she might be able to isolate the location trajectory of an individual and ultimately identify an individual in the worst case. Note that *such leaks are inherent to all digital tracing systems*. Nonetheless, DP (§4.2) combined with our assumption that an adversary cannot record ephemeral ids widely ensures that users cannot learn the complete history of an individual without stalking them physically.

6. Silmarillion PROTOTYPE

Table 2 summarizes the hardware configuration of the devices used in Silmarillion's prototype implementation. We implemented BLE beacons on Nordic nRF52832 development kits [2] and BLE dongles on SiLabs Thunderboard Kit SLTB010A [6]. All devices are powered through 3V/220mAh CR2032 coin cells. The network beacons consist of a SiLabs EFR32xG22 SoC (SLWSTK6021A) [7], which is connected to a Raspberry Pi 4B+ kit (4x 1.5 CPUs, 8GB RAM) [5] over a UART interface. All the Bluetooth devices support BLE 5.0. Only 64KB of flash is usable in dongles, with the remaining used by the platform software. While the storage was sufficient in our deployment (§7.3), we recommend using devices with at least 128KB of log storage.

BLE beacons. BLE beacons transmit ephemeral ids as legacy advertisements on BLE's advertising channels.

Backend. The backend server runs on two Dell PowerEdge R730 Servers, each with 16 Intel Xeon E5-2667, 3.2GHz cores, 512 GB RAM, and 1TB SSD. It main-

Device	CPU (MHz)	RAM (KB)	Flash (KB)	Power (dbm)
BLE beacon	1.3	32	512	[-40, 4]
Nw. beacon	78.6	32	512	[-10, 8.5]
Dongle	78.6	32	512	NA

Table 2: Hardware configuration

Operation	Frequency	Compute	Bandwidth	UI
ephid collect	high	low	low	no
risk calc	med	high	high	low
history upload	rare	med	med	med

Table 3: Dongle operations. (UI = User involvement)

tains $\texttt{DB}_{\texttt{PIR}}$ as an in-memory array, uses AVX256 for PIR, and computes new CFs daily from the uploaded ephemeral ids of the last 14 days. For our experiment, we simulated 434,448 tiles for central Europe and a very high infection rate in 130,040 tiles, resulting in a PIR block of max size 5 MB and $\texttt{DB}_{\texttt{PIR}}$ of 16.58 GB. To enable incremental risk score updates in dongles, the backend splits the data into multiple cuckoo filters, which are transmitted as chunks with distinct ids. Each chunk includes a filter of 128 indices with 4 buckets per index.

Network beacons. The Raspberry Pi requests the risk data of the last 24 hours from the backend server over HTTPS and caches it locally. The bluetooth board broadcasts risk information using the chained periodic broadcast protocol over data channels and 1M PHY (available since BLE 5.0). At periodic intervals, the bluetooth board fetches 250 bytes of risk data from the Raspberry Pi and places it in a buffer used by the underlying hardware for transmission[3].

Dongle. The dongle's key datastructure is a circular log on flash, which is used to store records of beacon encounters. The dongle implements five event handlers.

An *encounter handler* scans on the advertisement channel at 1s intervals with a duty cycle of 10%, and performs two actions based on the type of packet received. If the packet is an extended advertisement indicating the presence of a periodic broadcast on data channels, the handler initiates periodic scanning on the data channels. If the packet is a legacy advertisement, the handler decodes the encounter payload. It adds a new encounter to an in-memory list of "active" encounters, or updates an existing encounter's interval since the first instance of the same encounter was observed.

A *clock handler*, triggered once per minute, increments the dongle's clock and performs different actions on encounter entries in memory and on store depending on their state. It deletes stored encounters older than 14 days. For the "active" encounters older than one epoch (and thus ready to persist in the encounter log), it computes their cuckoo filter lookup indexes and appends the encounter and the indexes to the log.

A *download handler* receives a packet on the data channel and adds it to an in-memory buffer tracking the risk information payload. When the payload is complete, the handler decodes the cuckoo filter in the payload, looks up each dongle log entry in the filter, and sets a bit for each matched entry.

A *query handler*, triggered on a button press, initiates the active querying protocol for risk information. The dongle aggregates regions to query from the location ids of the recorded encounter entries. For each query, the dongle generates two secret shares using a hardware TRNG and encrypts each share using a separate key generated with hardware AES-CBC256. The keys can be derived from the dongle key and counter preshared with each backend PIR server. The dongle then sends both queries to a network beacon over a BLE connection, and the beacon then forwards the queries to the respective server. Each server encrypts its PIR response with its AES key and sends the response to the network beacon, which then transmits the response to the listening dongle. The dongle decrypts and XORs the response shares to retrieve the final response, decodes the cuckoo filter in the response, and performs lookups as above.

An *LED handler* periodically toggles a device LED. Normally, the LED blinks 5 times at 1s intervals every 2 min to indicate that the device is alive. After downloading risk entries, the user can press a button to check for an exposure (new matches in CF), which is indicated by the LED blinking continuously at 0.25s intervals for 2 min before resetting to the normal rate.

Due to limited RAM, dongles compute risk scores in a streaming manner. The download handler downloads a chunk of risk payload, performs the necessary lookups, then discards the chunk before downloading another chunk. The chunk ids track pending chunks.

We also implemented the dongle functionalities as an Android 11 app. The app captures beacon encounters similar to the dongle, but downloads the complete risk data of the last 14 days from the backend over HTTPS.

Our prototype does not support an end-to-end PIR protocol for risk querying, although we benchmark the protocol separately (§7.1). It also does not support storage encryption and encounter history uploads from a dongle to a trusted terminal. However, our phone app supports uploading a user's encounter history to the backend, thus enabling a pilot deployment. Lastly, we only transmit ephemeral ids in the risk information; extensions for intelligent risk scoring are left for future work.

[3]Our prototype network beacons do not emit ephemeral ids, since our hardware cannot multiplex legacy advertisements and periodic broadcasts. In practice, a BLE beacon can be placed close to a network beacon to alleviate this limitation.

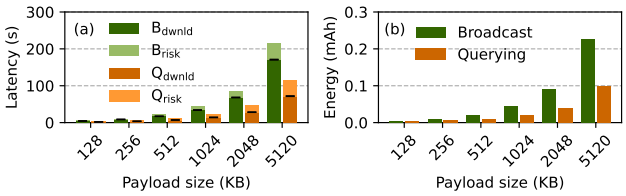
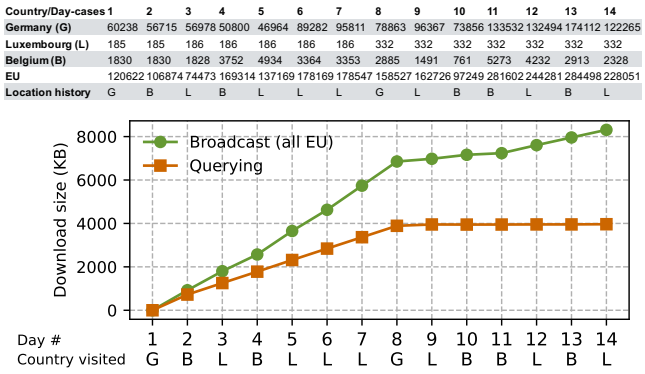


Figure 3: Risk payload sizes vs (a) end-to-end risk dissemination latency and (b) energy consumption. 128KB = 32768 ephemeral ids, 576 250B sized BLE packets.



Country/Day-cases	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Germany (G)	60238	56715	56978	50800	46964	89282	95811	78863	96367	73856	133532	132494	174112	122265
Luxembourg (L)	185	185	186	186	186	186	186	332	332	332	332	332	332	332
Belgium (B)	1830	1830	1828	3752	4934	3364	3353	2885	1491	761	5273	4232	2913	2328
EU	120622	106874	74473	169314	137169	178169	178547	158527	162726	97249	281602	244281	284498	228051
Location history	G	B	L	B	L	L	L	G	L	B	B	L	B	L

Figure 4: Download overheads based on regional infection rates [25] and a user's randomly generated location history (above). Broadcast includes all of EU; querying includes only countries visited in the last 14 days.

7. EVALUATION

We evaluate the practicality and usability of Silmarillion's design and implementation. BLE beacons perform a single task of generating ephemeral ids periodically; thus, they require low maintenance and the only practical concern is their battery life. For users' devices, the practicality and usability is determined by the time and frequency of interactions required with the devices and the battery life of the devices. Table 3 shows the compute, bandwidth, and user involvement characteristics of the three main operations performed by the devices: (i) ephemeral id collection, (ii) risk calculation for user, and (iii) encounter history upload. Given these characteristics, we care about the latency of (ii), since it requires high computation and bandwidth, and we care about power consumption of (i) and (ii), since they have the maximum impact on the device's battery life. We evaluate these metrics in §7.1-§7.2. We summarize our experience from Silmarillion's pilot deployment in §7.3.

Although Silmarillion supports smartphones, our evaluation focuses on dongles, which have no GUI and fewer compute, storage, power and bandwidth resources, and thus present a more challenging performance target.

All beacons transmit with a power of -10 dbm or equivalently 0.1 mW, unless stated otherwise.

7.1 Risk dissemination

One of the key challenges for Silmarillion is to ensure reliable delivery of risk information to dongles over Bluetooth, which is prone to packet losses and corruption due to sharing of the frequency spectrum with other technologies such as WiFi. We measure the end-to-end latency for risk notification and dongle power consumption when downloading risk payloads of various sizes using the broadcast and the active querying protocols. In all experiments, a dongle and a network beacon are placed 2m apart in an indoor, barrier-free environment. The numbers are averaged over three runs and the variance observed is less than 1%.

Latency. Fig. 3(a) shows the latency involved in downloading risk payload ($\mathbf{B_{dwnld}}$ and $\mathbf{Q_{dwnld}}$ for broadcast and querying, respectively) and for notifying the user of any matches as a function of different payload sizes ($\mathbf{B_{risk}}$ and $\mathbf{Q_{risk}}$ for broadcast and querying, respectively). First, we observe that the download latency over the periodic broadcast is 2.4x that over a connection, which is used for active querying. A beacon must use a large broadcast interval to minimize packet losses for all dongles. Furthermore, if a chunk packet is lost, a dongle must wait for the beacon to cycle through all chunks before it can retry downloading the chunk with the missed packet. In contrast, a BLE connection internally handles flow control and retransmissions, and provides reliable delivery at low latency.

The total risk estimation latency constitutes ~20% and ~37.3% of the end-to-end latency for the broadcast and querying protocols, respectively. The risk estimation latency is dominated by lookups of each of a dongle's encounter entry in each CF chunk downloaded (see streaming lookup in §6). In the querying protocol, the latency of decrypting and XOR-ing the PIR query results is negligible compared to the CF lookups. Similarly, uploading a query includes secret-share generation, encryption, and uploading the PIR queries. The querying latency is a constant 2.3s regardless of the risk payload size, and is at most 25% of the end-to-end latency. Finally, the query executes on the $\mathtt{DB}_{\mathtt{PIR}}$ in ~170ms in the backend, which is negligible compared to the dongle's compute and communication latency.

Energy consumption. Fig. 3(b) shows the energy consumption for risk notification as a function of different risk payload sizes. The broadcast protocol consumes ~90-130% more energy than the querying protocol.

Scalability. Broadcast is convenient for small downloads. However, the querying protocol is essential for scaling risk dissemination over large databases. Fig. 4 shows the bandwidth required for risk dissemination using each protocol. With broadcast, the bandwidth depends on the infection rates in the total region covered by the system regardless of users' location history. With

querying, the bandwidth is proportional to the number of beacons (regions) visited by the user and the infection rates of those regions.

7.2 Battery life

We measured the current consumption of BLE beacons and dongles using Simplicity Studio's energy profiler, and here we estimate the battery life of the devices.

BLE Beacons. The average current draw of a BLE Beacon is 7 μA. Thus, with a single coin cell of 220 mAh capacity, a beacon can last more than 3 years.

Dongles. The dongle's base current draw is 0.85 mA. In a 60-min period, the dongle's average current consumption is 0.9 mA when only scanning for beacon ephemeral ids, and 1.04 mA when additionally downloading risk information from a network beacon once using either periodic broadcast or connection-oriented communication. Thus, with a coin cell of 220 mAh capacity, the dongle is expected to last ∼8.8 days. Note that this is a conservative estimate, since users would download risk information only infrequently, e.g., once a day. With rechargeable cells of even 60 mAh capacity, the dongles can last ∼2.7 days on a single charge. This is practical, since users can be asked to place their dongles on a (wireless) charger overnight.

7.3 Deployment

We tested the end-to-end functionality of Silmarillion using a pilot deployment in the university building over 16 days[4]. We simulated infected users to trigger uploading of encounter histories and risk dissemination. The backend server was hosted in a single core Ubuntu 20.04.3 LTS VM with 1GB RAM and 32GB disk. We hosted users' data in a MySQL DB v8.0.27. We placed 8 BLE beacons, one each in various meeting rooms, labs, and social areas, and 2 network beacons in a subset of these spaces for risk broadcast. We involved 15 volunteers, 10 of whom carried a dongle and the rest used our smartphone app. The app users were asked to upload their encounter history to the backend on three random days. All users checked their simulated exposure risk everyday. Their devices downloaded the risk information over periodic broadcast channel only and recorded the number of encounters matched with the broadcast. App users saw the number of matched entries on their phone screens and the dongles' LEDs blinked faster to indicate non-zero matches.

Statistics. Over the period of the experiment, the beacons generated a total of 12288 unique ephemeral ids, and the user devices captured a total of 11670 of these ephemeral ids. When the app users uploaded their encounter history, they uploaded an average of 155 ephemeral ids to the backend. The number of other participants whose devices found matching entries for each of the three uploads was 8, 6, and 5.

The battery life of beacons and dongles was observed to be ∼80h and ∼100h, respectively. The low lifetimes are due to debug prints, which we retained for data collection and analysis. In addition, users were asked to reboot their dongle to trigger risk download at arbitrary times in the day. The dongle performs a long initialization step upon reboot, which lasts ∼1-2 min and draws ∼4 mA. A couple of reboots in a day consume significant power and bring down the battery life to just 4-5 days. However, as §7.2 indicates, the expected battery life is much higher in the absence of debug prints. In the future, the implementation can be optimized to eliminate the huge power consumption at risk data downloads.

User experience. Our users found both the dongles and the app intuitive and easy to use. In the future, the dongles could be allowed to pair with users' (trusted) personal devices to provide similar visualizations of data as the phone app.

Summary. Our evaluation indicates that dongles can receive risk information of their regions of interest with modest latency costs and that, with straightforward optimizations and larger batteries in devices, Silmarillion can be practically deployed with low maintenance costs.

8. DISCUSSION

Beacon placement. The density and placement of beacons is important for minimizing false negatives and false positives in Silmarillion. False positives arise when a user receives a risk notification even though they have not been in close contact with an infected user. For instance, a false notification may be generated when two users encounter a beacon placed on a glass door but from opposite sides of the door. False positives can be reduced by placing a sufficient number of beacons in a location and using well-known localization techniques [35], and by relying on the beacons' descriptors encoding information, such as temperature, humidity, etc.

False negatives arise when potential transmissions between users are missed, for instance, because of users meeting in locations where there are no beacons. Beacon deployments can be planned strategically to minimize false negatives. For instance, restaurants are likely to be more crowded than parks; therefore, restaurants must be prioritized in a partial rollout.

False positives in non-contemporaneous events. If Alice and Bob (who is infected) visit a beacon in the same epoch and Alice leaves before Bob's visit, she would still receive a risk notification for this beacon visit, even though she was not exposed to Bob. To eliminate such false positives, Silmarillion could additionally transmit in the risk information the beacon's start timestamp and interval as observed in the infected user's encounter with the beacon. For the matched ephemeral ids, dongles would then compare the beacon's timestamp and

[4]We received university IRB approval for the deployment.

interval recorded in their own log and in the risk information for overlap. A user would be notified only if the time interval in dongle's log overlaps with and starts later than the interval in the risk information.

9. RELATED WORK

We discuss digital CT systems that use various technologies, such as Bluetooth, GPS, or QR codes, to track users' trajectory and/or proximity to other users. We also discuss privacy-preserving techniques relevant for risk information retrieval.

P2P CT systems. SPECTS record instances of physical proximity with devices of other individuals via close-range Bluetooth exchanges between user devices. Centralized SPECTS [3, 4, 9, 21] collect and manage user data centrally, placing a high degree of trust in the central authority. Decentralized SPECTS [8, 11, 22, 27] minimize data collection to preserve privacy, but this prevents aggregation of data for epidemiology. Silmarillion facilitates analysis by enabling collection of contextual information with encounters.

In SPECTS, users' devices actively transmit messages and, thus, are vulnerable to eavesdropping and surveillance attacks [41]. Furthermore, an attacker could relay or replay captured ephemeral ids [20, 41]. Silmarillion overcomes these attacks because users' devices mostly listen passively, and beacons' location-time configurations can be corroborated with external trusted sources.

P2I CT systems. Reichert et al. [37] propose an architecture where all beacons ("lighthouses") and user devices are smartphones. Lighthouses collaborate with the backend for removing false positives in risk notification to a user who left a beacon before an infected user visited it (see §8). Consequently, users need to trust the lighthouses to not collude with the backend in leaking their data. Silmarillion eliminates the false positives in risk notification without requiring collaboration between the backend and the network beacons, which may be operated by untrusted third parties. Unlike lighthouses, Silmarillion can also handle relay/replay attacks.

PanCast [19] uses Bluetooth beacons and supports dongles similar to Silmarillion. However, PanCast focuses on evaluating—through simulations—the effectiveness of a beacon-and-dongle architecture for risk notification, and the benefits of interoperating with manual tracing. Silmarillion, on the other hand, builds a real system using low-end IoT devices, addressing technical challenges in achieving scalability and performance efficiency. PanCast assumes a pure broadcast-based risk dissemination architecture; however, Bluetooth is a bottleneck for large-scale data transfers. Silmarillion combines local risk broadcasts with an IT-PIR based active querying protocol, thus minimizing bandwidth, latency, and power costs for dongles.

Systems that use QR codes [1, 10, 12, 33] rely on static QR codes for each registered location. Static codes allow linking a user's multiple visits to the same locations, thus revealing more information about their location history. Other applications track location history using GPS [13, 24], which is imprecise and invasive, or using encounters with WiFi access points [40], which requires infrastructure that is relatively expensive compared to Silmarillion's infrastructure.

Privacy-preserving risk querying. Silmarillion's PIR technique with dynamic block sizes is similar to LBSPIR [36]. However, LBSPIR was designed for smartphone applications, which can retrieve the dynamic block layout of the PIR DB from the server and then adapt the size of each PIR query based on users' privacy preferences. Given the storage and network limitations of dongles, Silmarillion relies on a hierarchical geographical tiling, which enables PIR with minimal interaction between the server and a dongle, and provides a fixed privacy guarantee with fixed overheads for all queries.

EpiOne [39] proposes a two-party private-set intersection cardinality (PSI-CA) technique, to enable users to find *how many* entries in their encounter history match those of patients. Silmarillion reveals *which* location-time entries in a user's history match those in the risk information. Hence, Silmarillion provides more context for a user's exposure risk without compromising patients' privacy. Secondly, EpiOne relies on computational PIR, Merkle tree and zero-knowledge proofs to provide privacy for queriers and the infected individuals. These mechanisms have high computational and communication costs. Silmarillion provides similar guarantees but using IT-PIR, differential privacy, and cuckoo filters, which offload most computational costs to the server and thus are more suitable for low-end dongles.

10. CONCLUSION

We focus on building a systematic, inclusive, and scalable contact tracing and risk notification system in preparation for future needs. To this end, we present Silmarillion, a novel system for epidemic risk mitigation based on person-to-infrastructure encounters, showing that it is possible to extract significant utility without compromising on security. We presented the design and a prototype of Silmarillion along with a detailed analysis of its security, efficiency, and scalability. We demonstrated Silmarillion's practicality through a pilot deployment in a university building. We plan to evaluate Silmarillion in a real-world deployment in the future.

References

[1] CrowdNotifier - Decentralized Privacy-Preserving Presence Tracing. https://github.com/CrowdNotifier/documents. Accessed on 22 October 2020.

[2] Nordic nRF52832 Development Kit. https://www.nordicsemi.com/Products/Development-hardware/nrf52-dk. Last accessed on Aug 04, 2022.

[3] OpenTrace. https://github.com/opentrace-community/opentrace-android. Accessed on 28 June 2020.

[4] PEPP-PT (20 April 2020) Data Protection and Information Security Architecture. https://github.com/pepp-pt/pepp-pt-documentation/blob/master/10-data-protection/PEPP-PT-data-protection-information-security-architecture-Germany.pdf. Accessed on 28 June 2020.

[5] Raspberry Pi 4B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Last accessed on Aug 04, 2022.

[6] SLBT010A EFR32BG22 Thunderboard Kit. https://www.silabs.com/development-tools/thunderboard/thunderboard-bg22-kit?tab=overview. Last accessed on Aug 04, 2022.

[7] SLWSTK6021A EFR32xG22 Wireless Gecko Starter Kit. https://www.silabs.com/development-tools/wireless/efr32xg22-wireless-starter-kit?tab=overview. Last accessed on Aug 04, 2022.

[8] TCN Coalition. https://github.com/TCNCoalition/TCN. Accessed on 28 June 2020.

[9] TraceTogether. https://www.tracetogether.gov.sg/. Accessed on 28 June 2020.

[10] Canatrace. https://canatrace.com, 2020. Accessed on 15 Mar 2021.

[11] Decentralized Privacy-Preserving Proximity Tracing. https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf, 2020.

[12] SafeEntry. https://www.safeentry.gov.sg, 2020. Accessed on 15 Mar 2021.

[13] SafePlaces. https://github.com/Path-Check/safeplaces-dct-app, 2020. Accessed on 15 Mar 2021.

[14] https://github.com/SilmarillionPaper/TechReport/blob/master/silmarillion-tech_report.pdf, 2022.

[15] BearSSL. https://www.bearssl.org/constanttime.html, 2022.

[16] Tor. https://tb-manual.torproject.org/about/#:~:text=Tor%20is%20a%20network%20of,out%20onto%20the%20public%20Internet., 2022.

[17] Istemi Ekin Akkus, Ruichuan Chen, Michaela Hardt, Paul Francis, and Johannes Gehrke. Non-tracking web analytics. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[18] Benjamin M Althouse, Edward A Wenger, Joel C Miller, Samuel V Scarpino, Antoine Allard, Laurent Hébert-Dufresne, and Hao Hu. Stochasticity and heterogeneity in the transmission dynamics of sars-cov-2. *arXiv:2005.13689*, 2020.

[19] Gilles Barthe, Roberta De Viti, Peter Druschel, Deepak Garg, Manuel Gomez-Rodriguez, Pierfrancesco Ingo, Heiner Kremer, Matthew Lentz, Lars Lorch, Aastha Mehta, and Bernhard Schölkopf. Listening to Bluetooth Beacons for Epidemic Risk Mitigation. *Scientific Reports*, 2022.

[20] Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, Jonas Höchst, Joshua Kühlberg, Mira Mezini, Richard Mitev, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, Alvar Penning, Dermot Frederik Pustelnik, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. Mind the GAP: Security & Privacy Risks of Contact Tracing Apps. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020.

[21] Jason Bay, Joel Kek, Alvin Tan, Chai Sheng Hau, Lai Yongquan, Janice Tan, and Tang Anh Quy. BlueTrace: A privacy-preserving protocol for community-driven contact tracing across borders. *Government Technology Agency-Singapore, Tech. Rep*, 2020.

[22] Wasilij Beskorovajnov, Felix Dörre, Gunnar Hartung, Alexander Koch, Jörn Müller-Quade, and Thorsten Strufe. ConTra Corona: Contact Tracing against the Coronavirus by Bridging the Centralized–Decentralized Divide for Stronger Privacy. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2021.

[23] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: ARMing TrustZone with User-space Enclaves. In *Network and Distributed Systems Symposium (NDSS)*, 2019.

[24] MIT Media Lab Camera Culture Group. MIT Safe Paths Privacy Preserving WiFi Co-location for Contact Tracing without Prior Scanning of WiFi Signals. https://github.com/PrivateKit/PrivacyDocuments/blob/master/WiFiPrivacy.pdf, 2020. Accessed on 26 Oct 2020.

[25] CDC. Worldometers. https://www.worldometers.info/coronavirus/#countries. Accessed on 11 October 2020.

[26] David Cerdeira, José Martins, Nuno Santos, and Sandro Pinto. ReZone: Disarming TrustZone with TEE Privilege Reduction. In *USENIX Security Symposium*, 2022.

[27] Justin Chan, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, et al. PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing. *arXiv:2004.03544*, 2020.

[28] David L Chaum. Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2), 1981.

[29] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *IEEE Annual Foundations of Computer Science*, 1995.

[30] Roberta De Viti, Isaac Sheff, Noemi Glaeser, Baltasar Dinis, Rodrigo Rodrigues, Jonathan Katz, Bobby Bhattacharjee, Anwar Hithnawi, Deepak Garg, and Peter Druschel. CoVault: A Secure Analytics Platform. *arXiv preprint arXiv:2208.03784*, 2022.

[31] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014.

[32] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo Filter: Practically better than Bloom. In *ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2014.

[33] Andrew S Hoffman, Bart Jacobs, Bernard van Gastel, Hanna Schraffenberger, Tamar Sharon, and Berber Pas. Towards a seamful ethics of Covid-19 contact tracing apps? *Ethics and Information Technology*, pages 1–11, 2020.

[34] Lidia Morawska, Julian W. Tang, William Bahnfleth, Philomena M. Bluyssen, Atze Boerstra, Giorgio Buonanno, Junji Cao, Stephanie Dancer, Andres Floto, Francesco Franchimon, Charles Haworth, Jaap Hogeling, Christina Isaxon, Jose L. Jimenez, Jarek Kurnitski, Yuguo Li, Marcel Loomans, Guy Marks, Linsey C. Marr, Livio Mazzarella, Arsen Krikor Melikov, Shelly Miller, Donald K. Milton, William Nazaroff, Peter V. Nielsen, Catherine Noakes, Jordan Peccia, Xavier Querol, Chandra Sekhar, Olli Seppänen, Shin ichi Tanabe, Raymond Tellier, Kwok Wai Tham, Pawel Wargocki, Aneta Wierzbicka, and Maosheng Yao. How can airborne transmission of COVID-19 indoors be minimised? *Environment International*, 142:105832, 2020.

[35] Sharareh Naghdi and Kyle O'Keefe. Trilateration with BLE RSSI accounting for Pathloss due to Human Obstacles. In *Intl. Conf. on Indoor Positioning and Indoor Navigation (IPIN)*, 2019.

[36] Femi Olumofin, Piotr K Tysowski, Ian Goldberg, and Urs Hengartner. Achieving Efficient Query Privacy for Location Based Services. In *International Symposium on Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[37] Leonie Reichert, Samuel Brack, and Björn Scheuermann. Lighthouses: A Warning System for Super-Spreader Events. In *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021.

[38] Emily Seto, Priyanka Challa, Patrick Ware, et al. Adoption of COVID-19 Contact Tracing Apps: A Balance Between Privacy and Effectiveness. *Journal of Medical Internet Research*, 23(3):e25726, 2021.

[39] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight Contact Tracing with Strong Privacy. *arXiv preprint arXiv:2004.13293*, 2020.

[40] Amee Trivedi, Camellia Zakaria, Rajesh Balan, and Prashant Shenoy. WiFiTrace: Network-based Contact Tracing for Infectious DiseasesUsing Passive WiFi Sensing. *arXiv:2005.12045*, 2020.

[41] Serge Vaudenay. Centralized or Decentralized? The Contact Tracing Dilemma. Technical report, 2020.

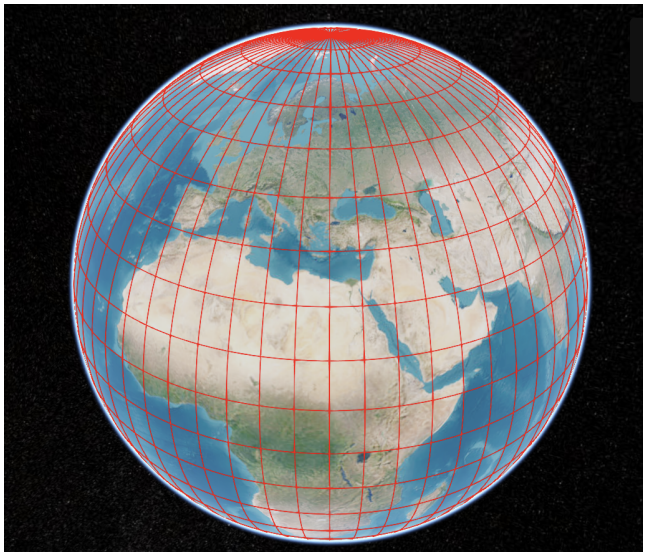


Figure 5: High-level tiling of earth's surface.

APPENDIX

A. Silmarillion TILING

We describe our tiling mechanism in detail here.

High-level tiling. Earth's surface is divided into 1.200 6° longitude × 9° latitude tiles. We will refer to this tiles as high-level tiles (H-tiles). 9° latitude degrees are, approximately, 1000 km, while 6° longitude degrees can vary between, approximately, 670 km at the equator to 110 km near the poles. Each H-tile has a unique 11 bit identifier.

Medium-level tiling. Each high-level tile is further divided into 100 km × 100 km squares. We will refer to these squares as medium-level tiles (M-tiles).The number of M-tiles in a single H-tile is inversely proportional to the latitude of the H-tile: as we get nearer to the equator the number of M-tiles in a single H-tile grows. The maximum number of M-tiles in a single H-tiles is 70. Each M-tile is uniquely identified by a 8 bits string. The full identifier of an arbitrary M-tile is the concatenation of the H-tile and M-tile identifiers.

Low-level tiling. Each medium-level tile is divided into 10000 1 km × 1 km tiles. We will refer to these small tiles as low-level tiles (L-tiles). Each L-tile has a unique 14 bits identifier and its full identifier is the concatenation of the H-tile, the M-tile and the L-tile identifiers.

B. PIR OPTIMIZATIONS

In this section, we describe two optimizations on the basic PIR implementation described in §4.3.

Deduplication. Silmarillion's backend maintains a dynamic mapping ($\texttt{DB}_{\texttt{PIR}}$) of L-tile IDs to fixed-sized blocks containing the risk entries of the L-tile. If the total number of risk entries in an M-tile is less than the block size, all L-tiles map to a single block. In this case, the $\texttt{DB}_{\texttt{PIR}}$ may contain a large number of duplicate blocks, leading to unnecessary costs of PIR query computation in the backend. We use an optimization in the backend to compress the database and the queries before performing the PIR operations on them. Our optimization removes any need for metadata while preserving information-theoretic privacy for users.

The $\texttt{DB}_{\texttt{PIR}}$ consists of unique blocks sorted by the longest prefix of the location IDs mapping to each block. Let Q_1 and Q_2 be the shares of query Q and let i be the index of the data block that the user wants to retrieve. Let $Q_1[j]$ be the j-th bit of query Q_1 and $Q_2[j]$ be the j-th bit of query Q_2. We know that

$$Q_1[j] = Q_2[j] \; \forall j \neq i \; .$$

Without any loss of generality, suppose that index i identifies an L-tile which does not correspond to any data block and suppose that the M-tile M that contains the i-th tile does correspond to a data block. Let $\{j_L\}$ be the set that contains all the indices of all the L-tiles contained in M. Remember that if M has not overflowed yet, all the L-tiles contained in M are empty. Server 1 computes

$$Q_1[j_M] = Q_1[j_M] \oplus \left(\bigoplus_{j \in j_L} Q_1[j] \right)$$

where $Q_1[j_M]$ is the bit correponding to M-tile M in Q_1 and then sets all $Q_1[j] : j \in \{j_L\}$ to 0 . Server 2 computes

$$Q_2[j_M] = Q_2[j_M] \oplus \left(\bigoplus_{j \in j_L} Q_2[j] \right)$$

and then sets all $Q_2[j] : j \in \{j_L\}$ to 0. Now, we are guaranteed that if the number of ones in the set $\{Q_1[j_L]\}$ is even then the number of ones in the set $\{Q_2[j_L]\}$ is odd and viceversa because $Q_1[i] \neq Q_2[i]$ and $i \in \{j_L\}$. Therefore, $Q_1[j_M] \neq Q_2[j_M]$. The rearranged query will now allow the user to retrieve the data block corresponding to M-tile M, i.e. the M-tile that contains the i-th L-tile.

Removing block padding. As a further optimization, the PIR implementation handles blocks of varying sizes without requiring padding to be persisted in the data blocks. It allocates a buffer for the response share corresponding to the max block size and initializes it to 0. Next, it reads each PIR block and XORs it into the response share buffer. Each block affects the XOR in the same way in both shares, regardless of the block size. Thus, if the client requested a small block the remaining bytes in the response shares will be automatically XOR'ed out, without revealing to the server which block was requested.

Note that each PIR block still includes dummy data to hide the number of risk entries uploaded by sick individuals; only the padding added to make all blocks uniform in size is removed.

C. ENCOUNTER INCONSISTENCIES

C.1 Verifying encounters

Before aggregating users' encounter history, the backend checks if each entry is well-defined and consistent. We describe the verification procedure.

Recall from §3.2 that an encounter entry is of the form: $\{eph_{b,i}, b, loc_b, desc_b, t^b_{start}, t^b_{int}, t^d_{start}, t^d_{int}, rssi\}$. Suppose the encounter happened at real time T. Taking timer offsets into account, we have: $t^d_{start} + \delta_d = T = t^b_{start} + \delta_b$. Here, δ_d and δ_b are the clock offsets of the dongle and the beacon known to the backend. The backend checks that $t^d_{start} + \delta_d = t^b_{start} + \delta_b$. Next, the backend calculates the beacon's epoch id $i_b = \lfloor (t_b + \delta_b - C_b)/L \rfloor$ at the time of encounter. The backend retrieves the beacon's secret key sk_b using the beacon identifier b from the encounter entry, and computes the expected ephemeral id $hash(sk_b, loc_b, i_b)$ for the epoch i_b. If the $eph_{b,i}$ in the encounter entry matches the expected ephemeral id, the encounter entry is consistent and the backend adds it to $\texttt{DB}_{\texttt{Risk}}$, else the entry is ignored.

C.2 Fixing clock inconsistencies

In this section, we explain with an example how the backend fixes beacon and dongle inconsistencies.

Suppose, at real time T, a dongle and a beacon encountered each other with local timers at t_d and t_b respectively and clock offsets at δ_d and δ_d, respectively. Suppose the beacon crashes and reboots at real time $T + \tau$ where $\tau > L$. Without loss of generality, assume that the same dongle encounters the beacon after the beacon has rebooted. The beacon's timer after reboot is $t_b + 1$, while the dongle's timer is $t_d + \tau$. Assume that t_b and $t_b + 1$ lie in a single epoch interval, i.e., the beacon's epoch id corresponding to the two times is the same. The dongle's encounter history includes an entry: $\{eph_{b,i}, b, loc_b, t_b, t_{b+1}, t_d, t_{d+\tau}, rssi\}$ When the backend observes that the dongle's local timers are more than one epoch length apart, it knows that the inconsistency was introduced due to a beacon restart. In this case, the backend updates the beacon's clock offset δ_b and the real time from when the offset comes into effect, and appends the tuple into the beacon's entry in the database. Specifically, the backend appends $\{C'_b, \delta'_b\}$ to a list of $\{clock, offset\}$ values stored with the beacon, where C'_b is the beacon's global clock in the backend at the time of encounter, and $\delta'_b = ((t_d + \tau) + \delta_d - (t_b + 1))$. A similar mechanism can be used to detect inconsistencies due to the crash of a user dongle. If there are multiple entries where the difference in the dongle timestamps does not match with the difference in beacon timestamps, the backend would update the dongle's clock offset appropriately. A beacon or dongle is restored to a consistent state when its clock offset equals the difference between its clock and local timer values.

D. ROBUSTNESS ANALYSIS

Security risks in Silmarillion can arise from misconfigured devices and adversarial principals. These can generate inconsistent encounters causing false risk estimations, or withhold risk information preventing timely notifications.

Inconsistent encounters may arise in three ways: (i) the clocks of beacons or dongles are out of sync with real time; (ii) a beacon is misconfigured and placed at a location different from where it is registered; or, (iii) an illegitimate beacon re-transmits a legitimate beacon's transmissions at a different location. We discuss mechanisms to identify and mitigate inconsistencies in encounters reported to the backend[5].

Clock inconsistencies. Encounters become inconsistent when an ephemeral id is found to have been used for more than one epoch length in real time. This may happen when devices crash and reboot after a long time, leading to encounter timestamps that are out of sync with real clock time. The backend can detect and fix such an inconsistency if it receives at least two encounters of an inconsistent device with consistent devices, where one encounter occurred before and the other after the device's crash. That is, the backend can fix an inconsistent beacon if it receives at least two encounters with the beacon from consistent dongles, and similarly, an inconsistent dongle if it receives at least two encounters of the dongle with consistent beacons.

Beacon misconfiguration. Inconsistencies also arise if a beacon transmits information inconsistent with its location. Such inconsistencies can arise if (i) a beacon was (accidentally or maliciously) installed in a location different from where it was registered, (ii) a spoofed beacon configured with the secret key of a legitimate beacon re-transmits the same ephemeral ids in a different location, or (iii) an adversary replays the ephemeral ids of a legitimate beacon in other locations [20]. All cases lead to the same inconsistencies due to the fact that the spoofed beacon is in a location different from where it is expected. First, dongles that travel between a spoofed beacon and nearby, legitimate beacons will appear to have traveled implausibly long distances in a short period of time. The backend can detect this when such a dongle uploads its history. Second, users with GPS-

[5]Fixing inconsistencies in beacons and dongles is unnecessary as long as no infections are being reported. Nonetheless, they can be fixed by re-syncing with help of, e.g., a bystander's smartphone.

ϵ	$\delta = 0.001$	$\delta = 0.01$
0.5	39098	29925
0.2	86969	64559
0.1	159131	115991
0.05	290088	210058

Table 4: 99th %ile noise required for various ϵ, δ.

enabled smartphones can directly observe the problem when they see a beacon transmission with a signed location that is different from the phone's current location by more than the BLE range. Such phones may report the inconsistency to the backend.

Network beacon failures. Failed network beacons can only cause denial-of-service, which can be mitigated by users moving to another network beacon.

Tampering or withholding upload data. In regimes mandating upload of the entire encounter history when diagnosed, users may attempt to evade the mandate by altering or withholding parts of their history. Modifying entries is impractical since it is infeasible to predict ephemeral ids without beacons' secret keys. Withholding entire periods of history will be easily detected; so a user may instead leave out all but one beacon's ephemeral ids in each epoch to provide a contiguous history and evade detection. Such an attack can be performed in two ways: (i) by using a jammer to selectively prevent transmission of certain entries, or (ii) by tampering the dongle. (i) can be mitigated by encrypting and MAC'ing the entire upload payload as a single chunk, while (ii) can be made harder by using a strong tamper-proof hardware casing.

E. PROOF OF DIFFERENTIAL PRIVACY OF NOISE ADDED TO RISK BROADCASTS

In Table 4, we first show the 99th percentiles of the number of noise entries required to achieve differential privacy with different levels of ϵ and δ. Below we prove the following differential privacy theorem, adapted from a similar theorem in the Appendix of [17].

Theorem 1. Let $t \in \mathbb{R}^+$, and let $\tilde{X}$ be a random variable sampled from the Laplace distribution with mean 0 and parameter λ, truncated to the interval $[-t, \infty)$.[6] Let f be a $\mathbb{Z}$-valued function with sensitivity A. Then, the function $\tilde{f}$ defined as

$$\tilde{f}(x) = f(x) + t + \lfloor \tilde{X} \rfloor$$

is (ϵ, δ)-differentially private if:

1. $\lambda \geq A/\epsilon$, and
2. $t \geq \lambda \cdot \ln\left((e^{(A/\lambda)} - 1 + \delta)/2\delta\right)$

[6] Note that if X is a standard (untruncated) Laplace random variable with mean 0 and parameter λ and Q is any predicate over real numbers, then $\Pr[Q(\tilde{X})] = \Pr[Q(X) \mid X > -t]$ by definition of $\tilde{X}$.

Proof. Because $f(x)$ is in $\mathbb{Z}$, we have $\tilde{f}(x) = f(x) + t + \lfloor \tilde{X} \rfloor = \lfloor f(x) + \tilde{X} \rfloor + t$. Hence, $\tilde{f}(x)$ is a function of $f(x)+\tilde{X}$. Consequently, by the post-processing theorem of differential privacy [31], it is enough to show that the function $g(x) = f(x) + \tilde{X}$ is (ϵ, δ)-differentially private.

So, pick two adjacent inputs x, x' and any output set O.[7] We need to show that

$$\Pr[g(x) \in O] \leq \delta + e^{\epsilon} \Pr[g(x') \in O]$$

Define $O_b = \{o \in O \mid o \leq f(x) - t + A\} \subseteq O$. Then,

$$\Pr[g(x) \in O] \quad = \quad \Pr[g(x) \in O_b] + \Pr[g(x) \in O \backslash O_b]$$

We now show that $\Pr[g(x) \in O_b]$ and $\Pr[g(x) \in O \backslash O_b]$ are bounded by δ and $e^{\epsilon} \Pr[g(x') \in O]$, respectively. Before delving into the details of these proofs, we explain the intuition behind these bounds and our definition of O_b. When $g(x) \in O_b$, because of the way we defined O_b, $g(x) \leq f(x) - t + A$. Since the distance between $f(x')$ and $f(x)$ can be A in the worst-case (x, x' are adjacent by assumption and A is the sensitivity of f), it is possible in this case that $g(x) \leq (f(x') - A) - t + A = f(x') - t$. Note that the lower end of $g(x')$'s range is exactly $f(x') - t$. Hence, in this case, it is possible that $g(x')$ will never equal $g(x)$, so differential privacy could "fail" in this case. This is why, this case corresponds to the "δ" part. Dually, when $g(x) \in O \backslash O_b$, we will have $g(x) = f(x) - t + A > f(x') - t$, so $g(x')$ will always have a non-zero probability of matching $g(x)$. Hence, this corresponds to the "$e^{\epsilon} \Pr[g(x') \in O]$" case of differential privacy.

Now we prove the bounds formally. We start by showing $\Pr[g(x) \in O_b] \leq \delta$. Let X denote a random variable sampled from an *untruncated* (standard) Laplace distribution with mean 0 and parameter λ. We have:

$$\begin{aligned}
\Pr[g(x) \in O_b] &= \Pr[g(x) \leq f(x) - t + A] \\
&= \Pr[f(x) + \tilde{X} \leq f(x) - t + A] \\
&= \Pr[\tilde{X} \leq -t + A] \\
&= \Pr[X \leq -t + A \mid X > -t] \\
&= \frac{\Pr[-t < X \leq -t + A]}{\Pr[X > -t]} \\
&= \frac{\Pr[X \leq -t + A] - \Pr[X \leq -t]}{\Pr[X > -t]} \\
&= \frac{\frac{1}{2}e^{\left(\frac{-t+A}{\lambda}\right)} - \frac{1}{2}e^{\left(\frac{-t}{\lambda}\right)}}{1 - \frac{1}{2}e^{\left(\frac{-t}{\lambda}\right)}} \\
&= \frac{e^{\left(\frac{A}{\lambda}\right)} - 1}{2e^{\left(\frac{t}{\lambda}\right)} - 1}
\end{aligned}$$

We continue using assumption (2) of the theorem's state-

[7] In our context, adjacent inputs are two situations that differ in exactly one user being sick or not. An "output" is the *length* of a noised risk broadcast, and O is any set of such possible lengths.

ment:

$$\begin{aligned}\texttt{Pr}[g(x) \in O_b] &\leq \frac{e^{\left(\frac{A}{\lambda}\right)} - 1}{2e^{\left(\frac{\lambda \ln\left((e^{(A/\lambda)} - 1 + \delta)/2\delta\right)}{\lambda}\right)} - 1} \\ &= \frac{e^{\left(\frac{A}{\lambda}\right)} - 1}{2\left((e^{(A/\lambda)} - 1 + \delta)/2\delta\right) - 1} \\ &= \delta\end{aligned}$$

Next, we compute the bound on $\texttt{Pr}[g(x) \in O \backslash O_b]$. Note that for any $o \in O \backslash O_b$:

$$\begin{aligned}\frac{\texttt{Pr}[g(x) = o]}{\texttt{Pr}[g(x') = o]} &= \frac{\texttt{Pr}[\tilde{X} = o - f(x)]}{\texttt{Pr}[\tilde{X} = o - f(x')]} \\ &= \frac{\texttt{Pr}[X = o - f(x) \mid X > -t]}{\texttt{Pr}[X = o - f(x') \mid X > -t]} \\ &= \frac{\texttt{Pr}[X > -t \wedge X = o - f(x)] / \texttt{Pr}[X > -t]}{\texttt{Pr}[X > -t \wedge X = o - f(x')] / \texttt{Pr}[X > -t]} \\ &= \frac{\texttt{Pr}[X > -t \wedge X = o - f(x)]}{\texttt{Pr}[X > -t \wedge X = o - f(x')]}\end{aligned}$$

Now note that by definition of O_b, $o \in O \backslash O_b$ implies $o > f(x) + A - t$. Hence, $o - f(x) > A - t > -t$. This implies that $(X > -t \wedge X = o - f(x)) \equiv (X = o - f(x))$. So the numerator simplifies to $\texttt{Pr}[X = o - f(x)]$.

Further, since x and x' are adjacent, and the sensitivity of f is A, we have $|f(x) - f(x')| \leq A$, which implies $f(x) > f(x') - A$. Hence, $o \in O \backslash O_b$ also implies $o > (f(x') - A) + A - t = f(x') - t$ or, equivalently, $o - f(x') > -t$. So, we also have $(X > -t \wedge X = o - f(x')) \equiv (X = o - f(x'))$. Hence, the denominator simplifies to $\texttt{Pr}[X = o - f(x')]$. Continuing,

$$\begin{aligned}\frac{\texttt{Pr}[g(x) = o]}{\texttt{Pr}[g(x') = o]} &= \frac{\texttt{Pr}[X = o - f(x)]}{\texttt{Pr}[X = o - f(x')]} \\ &= \frac{\frac{1}{2\lambda} e^{(-|o - f(x)|/\lambda)}}{\frac{1}{2\lambda} e^{(-|o - f(x')|/\lambda)}} \\ &= e^{((-|o - f(x)| + |o - f(x')|)/\lambda)} \\ &\leq e^{(|(-|o - f(x)| + |o - f(x')|)|/\lambda)} \\ &\leq e^{(|f(x) - f(x')|/\lambda)} \\ &\leq e^{(A/\lambda)} \\ &\leq e^{\epsilon}\end{aligned}$$

where the last inequality follows from assumption (1) of the theorem's statement. It then follows that for any $o \in O \backslash O_b$,

$$\texttt{Pr}[g(x) = o] \leq e^{\epsilon} \texttt{Pr}[g(x') = o]$$

and, hence, that:

$$\begin{aligned}\texttt{Pr}[g(x) \in O \backslash O_b] &= \textstyle\sum_{o \in O \backslash O_b} \texttt{Pr}[g(x) = o] \\ &\leq \textstyle\sum_{o \in O \backslash O_b} e^{\epsilon} \texttt{Pr}[g(x') = o] \\ &= e^{\epsilon} \cdot \textstyle\sum_{o \in O \backslash O_b} \texttt{Pr}[g(x') = o] \\ &= e^{\epsilon} \texttt{Pr}[g(x') \in O \backslash O_b] \\ &\leq e^{\epsilon} \texttt{Pr}[g(x') \in O]\end{aligned}$$

Combining everything we get the required differential privacy inequality:

$$\begin{aligned}\texttt{Pr}[g(x) \in O] &= \texttt{Pr}[g(x) \in O_b] + \texttt{Pr}[g(x) \in O \backslash O_b] \\ &\leq \delta + e^{\epsilon} \texttt{Pr}[g(x') \in O]\end{aligned}$$

□