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

**Anonymous authors**
Paper under double-blind review

## Abstract

Epidemics like the recent COVID-19 require proactive contact tracing and epidemiological analysis to predict and subsequently contain infection transmissions. The proactive measures require large scale data collection, which simultaneously raise concerns regarding users' privacy. Digital contact tracing systems developed in response to COVID-19 either collected extensive data for effective analytics at the cost of users' privacy or collected minimal data for the sake of user privacy but were ineffective in predicting and mitigating the epidemic risks. We present Silmarillion—in preparation for future epidemics—a system that reconciles user's privacy with rich data collection for higher utility. In Silmarillion, user devices record Bluetooth encounters with beacons installed in strategic locations. The beacons further enrich the encounters with geo-location, location type, and environment conditions at the beacon installation site. This enriched information enables detailed scientific analysis of disease parameters as well as more accurate personalized exposure risk notification. At the same time, Silmarillion provides privacy to all participants and non-participants at the same level as that guaranteed in digital and manual contact tracing.

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 two approaches: reactive and proactive. Reactive measures include testing and isolating infected individuals to prevent further spread of the disease. Proactive measures include contact tracing to identify other at-risk individuals, and performing epidemiological analysis to understand conditions for infection propagation, which can further inform policy decisions.

In principle, the data required for epidemiological analysis can be collected during contact tracing. Unfortunately, traditional manual contact tracing does not scale well and does not give good coverage as users tend to forget details of their recent encounters and visits. To scale manual tracing, several digital contact tracing systems have been proposed recently [2, 6, 9, 10, 16, 24, 27, 31], which record pairwise bluetooth encounters between users' smartphones to capture physical encounters (also referred as SPECTS[1]). Several countries adopted centralized contact tracing systems that supported extensive data collection for epidemiology [45, 46]. While these systems were effective for containing COVID-19, they raised important concerns about surveillance and users' privacy. Other countries decided to take a more conservative approach in the interest of users' privacy and adopted system designs that collected minimal data essential only for contact tracing but not epidemiology [3, 13–15, 17]. However, the importance of proactive epidemiological analysis can be understood from the fact that availability of such data early on could have helped in understanding the role of aerosols in spreading COVID-19 and enforcing social distancing and isolation much earlier [1, 11].

We seek to build a secure, robust, and scalable system that expands the utility of SPECTS by collecting additional data relevant to future epidemics, while preserving the privacy properties of SPECTS and manual contact tracing systems. We refer to this as an epidemic risk mitigation system. We address the following design goals in building such a system.

**G1. Rich data collection:** According to medical literature, epidemiology requires analyzing environmental conditions, demographics, and mobility patterns that promote disease transmission [29]. Thus, an epidemic risk mitigation system must collect circumstantial information associated with the user encounters, such as the location, location type, and time of encounter, as well as the environmental conditions under which the encounters occur (e.g., temperature, humidity, ambient noise levels, etc.). It must also support capturing non-contemporaneous encounters to determine if a disease could transmit through indirect exposure. Finally, the system must collect attributes of individuals (e.g., age, gender, occupation, etc.) to support identification of vulnerable demographics. **G2. Security and privacy:** Since the system collects sensitive user information, it must ensure security in collection, processing, and dissemination of the data, and balance utility and user privacy in the analytics. **G3. Timeliness:** The system must be able to collect accurate data and disseminate risk information in a timely manner even under a partial deployment,

[1] Smartphone-based Pairwise Encounter-based Contact Tracing Systems

low user adoption, and despite malicious or misbehaving participants. **G4. Inclusivity:** The system must be accessible to all demographic sections within a region of deployment.

The effectiveness of SPECTS was also limited by non-technical factors, such as low adoption rates. We do not address these factors in our work.

## 1.1 Our solution: Silmarillion

We present Silmarillion, a P2I system that relies on collection of location/environment-tagged encounters with BLE beacons installed in strategic locations to facilitate both contact tracing and epidemic analytics. At the same time, Silmarillion takes comprehensive measures to avoid indiscriminate collection and dissemination of users' encounter data, thus minimizing data leaks and misuse.

The deploying authority predetermines the analytics they wish to perform and accordingly the set of location and environmental attributes they wish to collect in beacon encounters. Beacons are then installed in strategic places that may be epidemiologically relevant, such as places where people tend to congregate (e.g., classrooms, markets, and theaters). Each beacon broadcasts (on short-range BLE radio) identifiers called *ephemeral ids* that are unique to the beacon, the current time (time is roughly quantized), and the beacon's location and environmental attributes (§3). Personal devices of nearby users record these ephemeral ids – in particular, two users in the vicinity of the same beacon at similar times will record the same ephemeral ids.

When a user tests sick, the ephemeral ids on their device from their period of contagion are collected at a backend. Users retain full control over what is sent to the backend (they may remove ephemeral ids corresponding to locations they consider sensitive), all uploads are anonymous, and a single user's ephermeral ids are divided into small chunks that are routed separately through a mixnet [49] to hide the user's trajectory from the backend (§4).

The backend periodically aggregates the ephemeral ids uploaded by sick individuals. It can utilize the data for epidemiological analysis, e.g., building mobility models, detecting superspreading events, predicting infection hotspots, determining environmental conditions that accelerate infection transmission, etc. (The design of the analytics backend and the analytics workloads that can be supported on the data collected by Silmarillion are beyond the scope of this work and have been covered in other work [34, 44].)

The backend also disseminates the ephemeral ids back to everyone for *decentralized* risk notification (§5). Other users match these disseminated ephemeral ids to those stored on their own devices, and assess their infection risk locally.

To ensure privacy of individual patients (who shared their encounter data) during risk notification, the backend adds differentially-private noise in the risk information disseminated to other users, which protects against a strong adversary

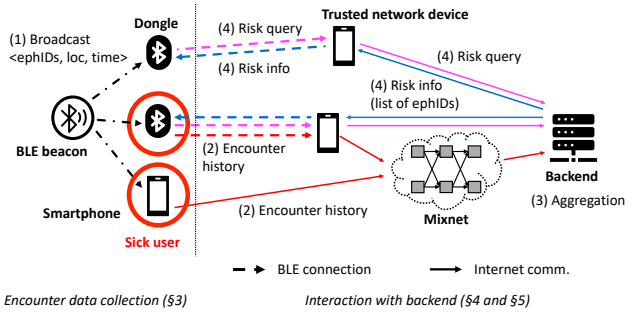


Figure 1: Silmarillion's architecture and workflow.

with auxiliary information about all other users in the system.

Silmarillion also provides privacy for users when they download risk information. Users query the backend for risk information relevant to them using an information-theoretic private information retrieval (PIR) protocol, without revealing their own data to the backend or an eavesdropper.

In summary, for the desired analytics support, Silmarillion's collection of location and environment information does not raise new privacy concerns for users. The Silmarillion backend can support rich analytics without learning sensitive information about individual participants in the system. Privacy of sick individuals is preserved from both the backend as well as other users and eavesdroppers. Similarly, privacy of other users is preserved.

We build (§6) and evaluate (§7) a prototype of Silmarillion with battery-powered BLE beacons, and user devices ranging from smartphones to low-end IoT devices. We demonstrate that Silmarillion can be deployed with low bandwidth, latency, and energy costs in data collection and dissemination.

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. Silmarillion can be deployed incrementally by placing beacons in locations of primary interest first. We envision Silmarillion to be deployed as a complement to recent contact tracing systems [16] (§8). While the latter are better suited for private or infrequently visited spaces, Silmarillion can better support crowded spaces, where non-contemporaneous transmissions may be prevalent and the existing systems would fail to capture such events.

# 2 Overview

Figure 1 shows an overview of Silmarillion's architecture and workflow. Silmarillion's main components include BLE beacons, personal devices, such as smartphones and dongles, and a backend platform that relays risk notifications and aggregates data to support epidemiological analysis.

(1) The beacons are placed in strategic locations (e.g., shops, restaurants) and continuously broadcast cryptographically-generated random strings called *ephemeral ids*

during typical operating hours of the location. Users' devices listen to these beacons passively (i.e., without transmitting anything) and store the beacons' ephemeral ids. (2) 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 personal device to the backend. The individual must explicitly authorize the transmission of data to the backend from their personal device. The user device chunks the encounter data into subsets of ephemeral ids, packages each subset into a separate message and uploads the messages to the backend via a mixnet. (3) 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. (4) Finally, user devices query the backend for risk information of specific regions, compare the ephemeral ids in their storage against those in the risk information disseminated, and notify their owners in case of non-zero matches.

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

## 2.1 Components

**Beacons.** Beacons are commodity, battery-operated, BLE-capable devices that may be installed in restaurants, squares, train stations, airports or even mobile locations, e.g., a city bus or a train. Beacons may be installed either by health authorities or by organizations that have received an approval from local health authorities. Each day, beacons remain active during during the typical operating hours of the location where they are installed, i.e., when the locations are visited frequently by many people.

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, and optionally a set of attributes, such as a location identifier comprising their stationary coordinate or a route id, a region (e.g., France), or other epidemiologically-relevant descriptors about their location (e.g., humidity, temperature). The descriptors are configured statically and can be used by the backend for intelligent risk estimation and/or epidemiological analysis, *thus addressing goal G1*. Furthermore, beacons remain active each day during the typical operating hours of the location where they are installed, i.e., when the locations are visited frequently by a lot of crowd. Thus, no individual user can be uniquely identified based on their encounter with a small subset of beacons, which *preserves privacy of patients sharing their data with the system and addresses goal G2.*

**User devices.** Silmarillion enables users to participate in the system with devices ranging from smartphones to simple dongles that can be attached to a keyring, or worn on the wrist or around the neck. The dongles are particularly useful for physically-, technologically-, or economoically-challenged individuals who cannot use smartphones.

To participate in Silmarillion, a user device must minimally include a coarse-grained timer, a counter, a small amount of flash storage, a UI to indicate risk status and battery condition (e.g., LED), and a button to control the LED notification. IoT boards already offer such capabilities today [4, 7, 8] and can be used as dongles. (Smartphones naturally have much higher capabilities.) Dongle users require an additional trusted networked device *only* to upload or download data from Silmarillion's backend. This device could be a smartphone of the user or a care provider.

By supporting diverse devices, Silmarillion provides an accessible and inclusive solution for different demographic sections of the society, *thus addressing goal G4*. In the rest of this paper, we describe Silmarillion's design and protocols mainly considering personal devices in the form of smartphones with a CT app, unless stated otherwise.

Similar to beacons, user devices are registered and authenticated with the backend and receive a public-private key pair from the backend. In addition, the users configure a password in their device, which they use to authenticate themselves to the device. The password and the counter are also used to control upload of data from the device.

**Testing authority.** Users get tested at a test center which is running by a trusted authority. If a user's test result is positive, the the test center issues a certificate for the result signed with the center's key. Furthermore, it issues several one-time signing keys to the user, with which the user signs their encounter entries prior to uploading to the backend. The signed uploads enable the backend to collect encounter data only from diagnosed individuals and only data corresponding to their period of contagion, thus ensuring use of accurate data for analytics and risk dissemination. *Thus, it ensures security in data collection, addressing goal G2.*

**Mixnet.** To enable users to upload their encounter entries to the backend without revealing their identity, Silmarillion relies on a mixnet similar to Vuvuzela [49], which mixes uploads from different users and hides the origin of uploaded data (see §4). *The mixnet ensures users' privacy during data collection, thus addressing goal G2.*

**Backend.** The backend may be managed by a health authority, the organization deploying beacons, or an independent entity. The backend maintains several databases. (i) `BeaconDB` contains each registered beacon's location/trajectory and the secret key used by the beacon to generate its unique sequence of ephemeral ids. (ii) `UserDB` contains registered users, their devices, and the public keys of the devices. (iii) `RiskDB` contains the encounter entries uploaded by diagnosed individuals and is used for risk dissemination and analytics.

To facilitate risk dissemination, the backend consists of two non-colluding servers in an IT-PIR setup. The servers derive a PIR database out of `RiskDB`, which they use to serve user queries for risk information of specific regions in a privacy-preserving manner (see §5). Additionally, the backend provides differential privacy in the number of entries uploaded by

an individual patient, thus preserving patients' privacy during risk dissemination. *The risk dissemination protocol addresses goals G2 and G3.*

## 2.2 Threat model

Silmarillion seeks to protect the privacy of users at a level comparable to manual tracing and SPECTS. The privacy of users can be violated when they transmit information, which happens at two points in Silmarillion: (1) When sick users upload their collected ephemeral ids to the backend, and (2) When healthy users query the backend for risk information in regions of interest to them. Silmarillion seeks to protect the privacy of users at both these points.

Threats in Silmarillion come from compromised network nodes, compromised users and beacons, and compromise of the backend. We assume a standard network adversary that can compromise a subset of the network nodes (routers, switches, servers), and monitor all traffic on the compromised nodes, but it cannot compromise a significant fraction of network nodes. This is particularly important for our use of a mixnet, where we assume that at least one mixnet node is uncompromised.

For users and beacons, we follow a *mostly honest* model, where users and beacons are generally honest but a small fraction may be controlled to act arbitrarily by the adversary. The backend is assumed to be honest-but-curious; if compromised, it follows the prescribed protocols but it may try to use information it sees and information that compromised users, beacons and network nodes see to break user privacy.

*Side channels.* 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 [20] to mitigate these attacks.

# 3 Encounter data collection

In this section, we describe the device configurations, the interactions of user devices with beacons, and the collection of encounter data on user devices. In §4, we discuss how user devices upload the data to a backend to facilitate epidemiological analysis and subsequent risk dissemination. In §5, we discuss how user devices interact with the backend to receive the risk information for contact tracing.

## 3.1 Initial configuration

Prior to its installation, a beacon is configured with a unique id $b$, an initial clock $C_b$ synced to real time, a secret key $sk_b$ that is known only to the beacon and the backend, and an optional descriptor $desc_b = \{a_{b,1}, a_{b,2}, ..., a_{b,n}\}$ that includes attributes, such as a location id, environmental conditions, indoor/outdoor, average temperature, ventilation or other important features of the place where the beacon will be installed.

Each user device is configured with a unique id $d$, the backend's public key, a initial clock $C_d$ synced to real time, a public-private key pair $(pk_{did}, sk_d)$, a monotonic counter $ctr$ from the backend, and a password from the user. In smartphones, the initial clock value may be the device's own wall-clock time. In dongles, the initial clock is set to the backend's wall-clock time at the time of device registration. The device's initial counter value is known only to the device and the backend and is used to ensure freshness of uploads from the device to the backend. The secret key $sk_d$ is stored only in the device and never leaves the device. The password is used to mutually authenticate the owner and the device whenever the owner interacts with the device (e.g., to initiate upload to the backend as in §4). For dongle users, the password is configured in both the dongle and the trusted networked device.

The beacon configurations are registered with the backend, which stores this data in the `BeaconDB` database in the form: {*device id*, *device key*, *initial clock*, *clock offset*, *descriptor*}, where *device key* is the secret key for a beacon.

Similar to beacons, user device configurations are also registered with the backend and stored in the `UserDB` database in the form: {*device id*, *device key*, *initial clock*, *clock offset*, where *device key* is the public key for a user device.

The clock offset in the backend is initialized to 0 during registration of a device and later used to track any divergence between the real time and the local timer of the device.

## 3.2 Capturing beacon encounters

Each user device 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 a user device and beacon, respectively). In our prototype, we use epoch length $L = 15$ minutes, similar to recent CT systems.

A beacon generates a new ephemeral id every epoch. In the $i^{th}$ epoch $i_b$, the beacon $b$ generates an id $eph_{b,i} = hash(sk_b, i_b, desc_b)$. Here $i_b = \lfloor (t_b - C_b)/L \rfloor$ and $hash$ is a one-way hash function. The beacon broadcasts $E_{b,i} = \{eph_{b,i}, b, i_b\}$ on legacy BLE advertisement channel and its descriptor $desc_b$ on a separate periodic advertising channel. Beacons broadcast each $E_{b,i}$ several times within an epoch. When a user device is in the bluetooth range of a beacon, it captures the beacon broadcasts. If the device encounters a new beacon id, it briefly listens to the periodic advertisement channel of the beacon to additionally capture the beacon's descriptor once. A user device persists a single entry for each unique ephemeral id along with the first beacon and device timestamps at which the id was received ($t_b^{start}$ and $t_d^{start}$), the duration for which the id was observed ($t_d^{int}$), and the average of the RSSI values observed ($rssi$). Thus, a log entry $enctr$ in user de-

vice database $d$ would be: $\{eph_{b,i}, b, t_b^{start}, t_d^{start}, t_d^{int}, rssi\}$. The device stores one instance of each unique $desc_b$ captured, which it may use to provide the device owner descriptive information about the owner's trajectory.

**Datastructure configurations.** The byte size of each field in an *enctr* is as follows. The ephemeral id $eph_{b,i}$ is 23 bytes, the device id $b$ is 4 bytes, the timestamps $t_b^{start}$, $t_d^{start}$ and the interval $t_d^{int}$ are 4 bytes each, and *rssi* is 1 byte. The $eph_{b,i}$ is generated by computing a SHA-256 hash of the inputs and taking the least significant 23 bytes of the result. Each beacon broadcast $E_{b,i}$ is 31 bytes and fits in a single legacy BLE advertisement; the descriptor $desc_b$ can have variable length. Each encounter stored in a user device is 40 bytes.

Assuming that users encounter on average no more than one unique ephemeral id every 10 min in a day, user devices 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 ∼79 KB of persistent storage. Assuming that each encounter also corresponds to a unique beacon and an average beacon descriptor length of 64 bytes, the device requires an additional 126 KB of persistent storage for the descriptors. In reality, a device is unlikely to encounter a unique ephemeral id every 10 min, much less a unique beacon, continuously for 14 days. Hence this estimate is very conservative. Overall, the storage requirement is satisfied by both smartphones and many IoT devices.

### 3.3 Security in encounter data collection

BLE beacons learn nothing about nearby users since they only transmit information unidirectionally. Similarly, no information is leaked during ephemeral id broadcast, since user devices only record information at this stage.

Risks during encounter data collection can arise from misconfigured devices and adversarial principals. These can generate inconsistent encounters causing false risk estimations. Inconsistent encounters may arise in three ways: (i) the clocks of beacons or user devices go 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 the encounters reported to the backend.

**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. While smartphones can directly re-sync clock time over the internet, the BLE-only beacons and small dongles cannot do the same. The backend can detect and fix such an inconsistency in its database based on the beacon and device timestamps uploaded in an encounter entry.

**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 [26]. 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. Users with GPS-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.

## 4 Encounter data upload

When users feel sick or are notified of potential exposure (§5), they may visit a test center or a clinic for testing. 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 device and the test kit used for them. Once the test results are available, the user is informed using their contact details, such as their email id or phone number. If the result is positive, the user may wish to or be required by law to upload their data to assist in dissemination of risk information and epidemiological analytics.

We start by discussing the competing challenges involved in designing a privacy-preserving upload mechanism.

### 4.1 Requirements

The upload mechanism needs to address four requirements. First, because encounters contain contextual information (e.g., beacon location), uploading encounters may reveal a user's entire trajectory, which would be a violation of their privacy. To ensure privacy of diagnosed individuals, the upload protocol must provide:

**U1.** *Anonymity*: the backend or a network adversary cannot learn the identity of any user uploading encounters.

**U2.** *Unlinkability*: the backend or an adversary cannot learn if two parts of a trajectory belong to the same user or not.

Furthermore, the protocol must be reasonably efficient in terms of overall network traffic:

**U3.** *Efficiency*: The network traffic generated by the protocol should be linear in the amount of data actually transferred from users to the backend, ideally higher by only a small factor.

Finally, the protocol must be robust against malicious users who may attempt to generate false alarms and panic among

users, for instance, by uploading fake entries to the backend or uploading legitimate entries without having been diagnosed positive. Specifically, the protocol must support:

**U4.** *Upload authentication*: the backend must verify that the uploads came from a registered user who tested positive.

We discuss Silmarillion's mixnet-based upload protocol in §4.2, which addresses the requirements U1-U3. We discuss authentication (U4) in §4.3 and initiation of encounter uploads from user devices in §4.4.

## 4.2 Upload protocol

**Message format.** Since uploading a user's complete encounter history can compromise the user's privacy, the user device chunks the history into small subsets of $t$ encounter entries and uploads them in separate messages. The privacy guarantees rely on a key assumption that a user cannot be uniquely identified by a small segment of $t$ ephemeral id records of her trajectory. This is a reasonable assumption since we expect Silmarillion beacons to be installed strategically in crowded places during busy hours, e.g., train stations, airports, markets, etc. (see §2).

A user device splits the encounter data into messages as follows. First, it shuffles the encounter entries in the device log, and then divides the shuffled log into 24 subsets (one for each hour of the day). Each subset contains at least $t$ and at most $2016/24 = 84$ ephemeral ids. (Recall from §3 that the max number of entries in a device log can be 2016.) Then, the device places each subset of entries in a separate message, pads each message with dummy entries as required up to a fixed message size $M$, encrypts the message with the backend's public key and signs the message with a unique key provided by the test center (see §4.3 for signing messages). Each message is then uploaded to the backend through a mixnet, as explained below.

**Mixnet rounds.** Silmarillion's upload protocol relies on a mixnet, such as [40], [33], [49]. We assume the mixnet consists of a chain of $r$ servers. We make the standard assumption about the mixnet service that at least one server in the mixnet is honest.

The upload protocol runs in synchronous rounds. Specifically, we divide each hour into $n_r$ rounds, each of them $60/n_r$ minutes long. Every hour, each user device sends messages to the mixnet in only one of those rounds. Each device is assigned a round randomly. A device uploads (a subset of) the encounter data in the message if available, and dummy data otherwise.

We now explain how **U1–U3** are attained.

**U1**: Given $n$ Silmarillion users, and an average participation rate of $R$ in any given round, the probability that any given round has $k$ participants is $\binom{n/n_r}{k} R^k (1-R)^{(n/n_r)-k}$. Even for a small city with $n = 100,000$, and $n_r = 10$, $R = 10\%$, the probability that there are at least 1,000 participants in a given round is more than 50%, which implies a high degree of anonymity.

**U2**: Users anonymize small subsets of their trajectories and beacons operate only in densely visited places, making it difficult for the an adversary to link two trajectory subsets to the same user.

**U3**: The average network traffic generated every round in the system due to the upload protocol is

$$T = \frac{n \cdot R \cdot M \cdot r}{n_r}$$

If a user is sick with probability $p$ on any given day, then the actually meaningful traffic would be $A = (n/n_r) \cdot R \cdot M \cdot p$ per round. Hence, the average traffic overhead is $T/A = r/p$. For $r = 4$ (a 4-round mixnet) and $p = 0.02$ (2% users sick at any given time), this overhead is 200x. Given that the actual traffic generated by each sick user's device is about 126KB in 14 days or 9KB per day (see section 3.2), this 200x overhead still amounts to only 1.8 MB traffic per device each day, which can be easily tolerated even when user devices have limited network connectivity.

## 4.3 Upload authentication

We now describe how a user device authenticates and initiates uploads. One concern for uploading is that users may upload incorrect or fake encounter entries, e.g., by uploading entries without having been diagnosed positive or by uploading entries that are older than the period of contagion. The backend can easily discard dummy and invalid entries that could not have been generated by any registered beacons, as well as entries with timestamps that are too old to be relevant for risk notification or epidemiology. To mitigate the risk of users uploading without being sick, we describe a mechanism to authenticate user uploads.

An upload authentication mechanism must enable the backend to verify that each entry has been uploaded by a user who was diagnosed positive by an authorized test center. A simple solution would be having users upload their encounter data along with a certificate from the test center, signed with the test center's key, indicating the test date, and the ids of the patient's device and test kit. However, the user would need to upload the certificate with each encounter entry, which would defeat the goal of ensuring unlinkability of the user's entries. Instead, the user must be able to attest each of their encounter entries independently. We describe the solution next.

The authentication mechanism relies on test centers playing the role of a trusted third-party. When a test center generates a positive result for a user, it releases a one-time password (OTP) to the diagnosed user and to the backend. The OTP may be derived from a master secret $M_T$ of the test center and a counter $C_T$ representing the number of users who tested positive at the center. The user then sends the OTP to the backend and downloads $N$ one-time signing keys from a database of

keys in the backend using an *oblivious transfer* (OT) protocol, which prevents the backend from learning sets of keys that were downloaded together. Subsequently, the user can sign each of their trajectory subsets with one of the downloaded keys each. When the user uploads the subsets, the backend can verify the signatures on the subsets by trying the verification keys. Since the backend does not know which keys were given to the same user, it cannot link the different uploads of the same user to each other.

The authentication mechanism can partially prevent a sick user from authenticating arbitrary entries and uploading them to the backend. A sick user may authenticate entries that their device never recorded, e.g., by copying entries from a colluder's device to their own device. The backend may be able to detect if a user uploads entries from multiple distant locations at the same time; however, it can do so only within a subset of entries uploaded together, but not across independent subsets. Thus, the length of the trajectory subsets trades off unlinkability and the ability to detect malicious behavior.

A malicious user could also simply upload a consistent encounter history of a different device. This risk cannot be entirely eliminated, since it is difficult to verify whether a device logged entries in the proximity of beacons or not. However, this *analog loophole* exists in all digital contact tracing systems, not just the one we are proposing here.

### 4.4 Initiating upload from a user device

Next, we discuss how user devices initiate upload of entries. Depending on the user device, the upload mechanism requires different steps as described below.

**Smartphone upload.** A user initiates the data upload after they receive a positive test result. A smartphone user downloads the OTP from the test center, forwards it to the backend and downloads the one-time signing keys from the backend, all over the internet.

**Dongle upload.** Dongle users need to download the signing keys with the help of a trusted network device. The trusted device could download the OTP from the test center on behalf of the dongle, forward it to the backend, download the one-time signing keys from the backend and finally forward the keys to the dongle over BLE. To initiate upload, the dongle user then establishes a secure connection between the dongle and the trusted device by entering their dongle's id and password on the trusted device's UI and instructing the device to establish an authenticated session with the dongle. The dongle encrypts each encounter entry with the backend's public key, signs it with one of the signing keys, and then uploads the encrypted and signed payload to the trusted device, which then uploads entries to the backend via the mixnet.

Note that the encounter history is not released in cleartext to the personal device.

### 4.5 Security analysis of the upload mechanism

The upload protocol authenticates a user's encounter entries to the backend without linking the trajectory subsets to each other or to the user. This is achieved as follows. If a backend can verify the signature on an uploaded trajectory subset, it knows the subset was signed using a one-time signing key provided by the backend. A user could have received the signing key only upon authenticating itself to the backend with a valid OTP that was generated by a trusted test center, which in turn would have generated the OTP only if the user was diagnosed positive. At the same time, the OT protocol prevents the backend from learning the keys downloaded by the user and the user from learning the keys that they do not wish to use.

Even when the backend has retrieved all the ephemeral ids that were deposited in all the mailboxes, it cannot piece together the full trajectory of any single user. This is because, by assumption, no ephemeral id is unique to any individual and the uploading protocol prevents linking a sequence of messages together and to a specific individual. Moreover, the backend cannot identify the sick users and, therefore, cannot know with certainty to which user a particular set of ephemeral ids belong.

Nevertheless, for added protection of the encounter data collected, the backend can be further secured using standard hardware and cryptographic techniques [34].

## 5 Risk dissemination

We start with an overview of the risk information structure and the risk notification mechanism in the user devices. 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 user device has previously recorded any of the ephemeral ids listed in the risk information, its owner may have been exposed to a diagnosed individual. The device computes a risk score based on the number of matched ephemeral ids and (optionally) other features of the matched encounters. If the risk exceeds a certain threshold, the device notifies the user so that they can self-isolate and get tested. In dongles, the notification can be generated by having the user press a button on the dongle and the LED blink with a specific pattern.

We now discuss how the risk information is disseminated from the backend to user devices. 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.

## 5.1 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 (§5.2), **D3.** the protocol must maintain privacy of the users seeking the risk information (§5.3), and **D4.** the relevant information must reach potentially affected users in a timely manner and with low bandwidth, power, and computational costs for user devices (see §5.3). Note that all risk information is signed by the backend to allow detection of any tampering, which addresses D1.

## 5.2 Noising the risk dissemination

We present two scenarios where the *number of entries* in the risk information 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 $n$th 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 device. 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), which we describe next.

We adapt a mechanism proposed in prior work [23]. Given a risk payload, 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 $\lambda$ *truncated* to the interval $[-t, \infty)$. The values of $t$ and $\lambda$ depend on the privacy required. To get $(\varepsilon, \delta)$-DP, we pick $\lambda = A/\varepsilon$ and $t = \lceil \lambda \cdot \ln\left((e^{(A/\lambda)} - 1 + \delta)/2\delta\right) \rceil$. Here, $A$ is the sensitivity of the risk payload 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 $\varepsilon = 0.1$ and $\delta = 0.01$, the 99th %ile noise required is 115991. We prove that our mechanism is $(\varepsilon, \delta)$-DP in appendix B.

## 5.3 Dissemination protocol

A key requirement for Silmarillion is to ensure that users can receive risk information without revealing their own encounter history. Additionally, traveling users must be able to access global risk information to receive reliable risk estimation.

A naïve way to satisfy these requirements would be to broadcast the complete risk information to users and let the users' devices filter the data for relevant matches. However, this could incur high latency, power, and computational costs for the user devices. Silmarillion simultaneously addresses requirements D3 and D4 by using a IT-PIR protocol that allows users to query the backend for risk information without revealing their own encounter entries. Below, we describe the PIR datastructures and the protocol.

**PIR datastructure.** The backend enables users to query for risk information in fixed-sized blocks, where blocks are derived by grouping the `RiskDB` entries based on a set of one or more beacon attributes. The grouping function $G$ may be, for instance, based on a region identifier attribute associated with the beacons (e.g., zip code or country code), or based on a location type attribute (e.g., airports, theatres, etc.). The backend may support one or more grouping functions. Suppose a function $G$ yields $N$ possible values for its attribute set $\{g_1, g_2, ..., g_N\}$. The backend maintains an $N$-length array $\mathtt{D}^{\mathrm{PIR}}$, where entry $\mathtt{D}^{\mathrm{PIR}}[\mathtt{i}]$ corresponds to a block for $g_i$. The entry is a data block if there are non-zero number of `RiskDB` entries with $g_i$ in their beacon attributes, otherwise it contains a dummy block. In other words,

$$\mathtt{D}^{\mathrm{PIR}}[\mathtt{i}] = \begin{cases} G(\mathtt{RiskDB}, \mathtt{g_i}), & \text{if } \exists\ e \in \mathtt{RiskDB},\ \mathtt{g_i} \in \mathtt{e.desc_b} \\ \text{dummy}, & \text{otherwise} \end{cases}$$

For each function $G$, the backend maintains a separate $\mathtt{D}^{\mathrm{PIR}}$.

To ensure that a user does not learn the actual number of ephemeral id entries within a block (e.g., in a region-based grouping) and to make the block size uniform, the backend adds dummy entries to each block, following the DP mechanism of §5.2, upto the uniform block size of $\mathtt{D}^{\mathrm{PIR}}$.

**Dynamic and hierarchical grouping.** A key practical challenge is that as the entries uploaded by users evolve each day, the distribution of risk entries may vary in the groups generated by a grouping function. Consequently, the block

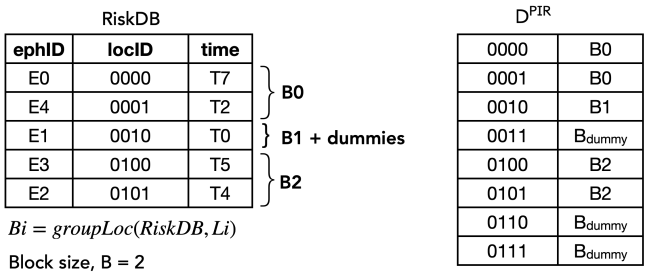


Figure 2: PIR DB in the backend on a given day.

size for a $\mathtt{D}^{\mathrm{PIR}}$ may need to be changed frequently. Furthermore, a skewed distribution of risk entries may require very large block sizes, leading to unnecessary bandwidth overheads. Therefore, the backend dynamically adjusts the grouping of `RiskDB` entries based on the prevailing infection rate distribution, while maintaining a uniform block size $B$.

For a grouping function, the backend organizes the attribute ids hierarchically like a B$^+$-tree, whose height and fanout depend on the desired block size. A $\mathtt{D}^{\mathrm{PIR}}$ block corresponds to a B$^+$-tree node at a certain level if the total number of risk entries for all nodes below it is $\leq B$. When the number of entries overflows $B$, the entries are split into new $\mathtt{D}^{\mathrm{PIR}}$ blocks that are associated with B$^+$-tree nodes at the next lower level. Partially-filled blocks are padded with dummy entries as above.

**Block encoding.** Once the blocks are generated, the backend then encodes each block into a cuckoo filter (CF) [37]. 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. The CFs also reduce bandwidth overheads, albeit at the cost of a small percentage of false positives. For instance, a CF 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. Figure 2 shows the PIR database.

**PIR protocol.** We use a PIR protocol based on [32]. Our scheme relies on two servers, $S_1$ and $S_2$, each of which has a complete copy of the $\mathtt{D}^{\mathrm{PIR}}$. We make the standard assumption that at least one server is non-malicious and non-colluding. Suppose $\mathtt{D}^{\mathrm{PIR}}$ contains $N$ blocks and let the size of the largest block in $\mathtt{D}^{\mathrm{PIR}}$ be $B$ bits. We assume that all blocks are padded with dummy entries up to size $B$.

First, the user device queries the backend for the grouping functions supported. In Silmarillion, we support the region-based grouping function which maps each encounter entry to an enumerated region id. The user device applies the function on its own encounter log to determine the unique regions visited and, therefore, for which it needs the risk information.

To query for the $n$-th region (block) in $\mathtt{D}^{\mathrm{PIR}}$, a user device 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 device 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 $S_j$ generates a response $A_j = \bigoplus_{k=0}^{N}(Q_j[k] \cdot \mathtt{D}^{\mathrm{PIR}}[\mathtt{k}])$, encrypts it with the client device's encryption key, and returns it to the client. The user device decrypts each response from the two servers and XOR's them to retrieve $\mathtt{D}^{\mathrm{PIR}}[\mathtt{n}]$.

The user device's complexity for generating a query-pair and uploading the queries, and each backend's computation complexity are $O(N)$ each. The cost for receiving the response shares and retrieving a block is $O(B)$ each.

For querying, the user device sets up an end-to-end secure session with each backend PIR server and then issues PIR queries for each unique block covering the logged encounters. A smartphone user can directly connect with the servers over Internet. A dongle user offloads risk querying to its trusted networked device, as is done for encounter history upload.

Furthermore, the user device generates fake queries using DP (similar to §5.2) to hide the actual number of queries issued in each round of risk querying (about once a day).

## 5.4 Security analysis of risk dissemination

Users 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.

For diagnosed individuals, the DP mechanism in the risk dissemination 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.

Silmarillion also ensures strong privacy of users receiving risk information. As long as one of the PIR servers is non-malicious, user's privacy is protected from the servers, since each server receives only a share of the query and iterates over the entire $\mathtt{D}^{\mathrm{PIR}}$ regardless of the query, and only the user can recover the DB block from the response shares of the servers. User generates a number of queries following DP, and all queries and results are encrypted end-to-end between the clients and servers, thus preventing leaks to eavesdroppers. The only remaining leak is through the timing and number of risk notification rounds.

Thus, the backend learns no information about queriers, particularly healthy users, except the times when they query for risk data.

**Limiting case.** A user can still learn the specific ephemeral ids it collected from a beacon they visited at the same time as a sick individual. 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 (§5.2) com-

bined 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

As a proof of concept, we implemented Silmarillion using low-cost BLE beacons, BLE-only dongles, and smartphones with both BLE and network capabilities. We implemented BLE beacons on Nordic nRF52832 development kits [4] and BLE dongles on SiLabs Thunderboard Kit SLTB010A [7]. All the Bluetooth devices support BLE 5.0. The BLE-only devices are powered through 3V/220mAh CR2032 coin cells. Only 64KB of flash storage is usable in dongles, with the remaining storage being used by the platform software. While the storage was sufficient in our evaluation (§7), we recommend using devices with at least 128KB of log storage.

**BLE beacons.** The BLE beacons are configured with a location id attribute, which is an 8-bit integer indicating geo-coordinates of the beacon within a 1 $km^2$ region. The beacons transmit ephemeral ids as legacy advertisements on BLE's advertising channels.

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

An *encounter handler* scans on the legacy advertisement channel at 1s intervals with a duty cycle of 10%. When the dongle receives a packet, 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 *query handler*, triggered on a button press, initiates the 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 pre-shared 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 a network device, 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, looks up each dongle log entry in the filter, and sets a bit for each matched entry.

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

Table 1: Dongle operations. (UI = User involvement)

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 query 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.

Although straightforward, we did not implement the upload pipeline from a dongle to a network device, since the costs of this pipeline would be much smaller than the costs for the network device to participate in the mixnet protocol for uploading to the backend.

**Smartphone.** We also implemented the user device functionalities as an Android 11 app. The app captures beacon encounters similar to the dongle. Additionally, it uploads the device's encounter history for the last 14 days to the backend over HTTPS and downloads the complete risk data of the last 14 days from the backend over HTTPS.

**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 maintains $\mathtt{D}^{\mathrm{PIR}}$ 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 use a PIR block size of 5 MB and a PIR database $\mathtt{D}^{\mathrm{PIR}}$ of ~430K blocks (regions). To enable incremental risk score updates in user devices, 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. The backend only transmits ephemeral ids in the risk information; extensions for intelligent risk estimation are left for future work.

## 7 Evaluation

In this paper, we evaluate the practicality and usability of Silmarillion's design and implementation for the use case of contact tracing and risk notification. An evaluation of the effectiveness of beacon-based tracing has been shown in prior work [25].

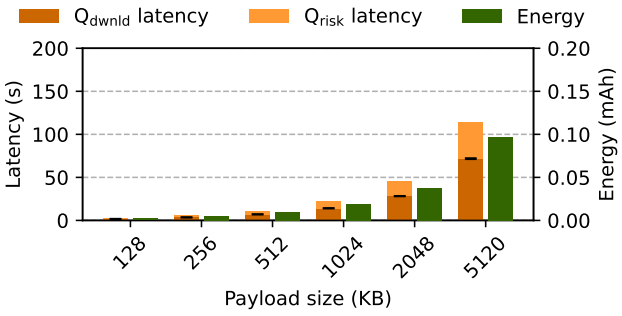


Figure 3: Risk payload sizes vs end-to-end risk dissemination latency and energy consumption. 128KB = 32768 ephemeral ids, 576 250B sized BLE packets.

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 frequency of interactions required with the devices, the timeliness of risk dissemination, the bandwidth costs, and the impact on the battery life of the devices. Table 1 shows the compute, bandwidth, and user involvement characteristics of the three main operations performed by the devices: (i) ephemeral id collection, (ii) encounter history upload, and (ii) risk calculation for a user. Given these characteristics, we care about the latency of (ii), since it impacts the overall timeliness of risk dissemination to other users, latency and bandwidth of (iii), 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. In §7.1 and §7.2, we present an evaluation of the latency and energy costs of realistic implementations of beacons and low-end dongles. In §7.3, we describe our experience from a pilot deployment of Silmarillion with BLE beacons and user devices implementing a basic, functional subset of all the features described in our design.

## 7.1 Risk estimation latency

Although Silmarillion supports smartphones, our evaluation for personal risk estimation focuses on dongles, which have fewer compute, storage, bandwidth, and power resources, and thus present a more challenging performance target.

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 device 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%.

Figure 3 shows the latency involved in downloading risk payload ($\mathbf{Q_{dwnld}}$) and for estimating and notifying the user of any matches (risk) as a function of different payload sizes ($\mathbf{Q_{risk}}$). The total risk estimation latency constitutes ∼37.3% of the end-to-end latency. This latency is dominated by lookups of each of a dongle's encounter entry in each CF chunk downloaded (see streaming lookup in §6). 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 $\mathtt{D}^{\mathtt{PIR}}$ in ∼170ms in the backend, which is negligible compared to the dongle's compute and communication latency.

The green bars in Figure 3 show the energy consumption for risk notification for different risk payload sizes.

## 7.2 Battery lifetime

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. Again, we do not analyze the battery life of smartphones, since they are already provisioned with much higher battery capacity.

**Beacons.** The beacons transmit on BLE channels with a power of -10 dbm or equivalently 0.1 mW. The average current draw of the beacon is 7 $\mu$A, which is dominated by the current draw during BLE transmission. 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

Our prototype does not yet support an end-to-end implementation of the upload protocol and the PIR protocol for risk querying. However, our phone app supports uploading a user's encounter history directly to the backend and downloading the complete risk information from the backend, thus enabling a pilot deployment.

We tested the end-to-end functionality of Silmarillion using a pilot deployment in the university building over 16 days[2]. We simulated infected users to trigger uploading of encounter

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

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.

**User experience.** Our users found both the dongles and the app intuitive and easy to use. In the future, the smartphone app could provide better visualization of the encounter data. The dongles could also be allowed to pair with a personal device, which the user is willing to trust, to provide similar visualizations of data as the phone app.

**Evaluation summary.** Our results indicate that Silmarillion can be practically deployed with low infrastructure and maintenance costs, and can be easily adopted by users. Both smartphones and low-end dongles can upload encounter data and receive risk information of their regions of interest with modest bandwidth, latency, and energy 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 [41], 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, user devices would then compare the beacon's timestamp and interval recorded in their own log and in the risk information for overlap. A user would be notified only if the time interval in device's log overlaps with and starts later than the interval in the risk information.

**Interoperability with existing CT systems.** Silmarillion's beacons can broadcast ephemeral IDs compatible with the Google/Apple Exposure Notification (GAEN) protocol used by most SPECTS [38], allowing the beacons to seamlessly interoperate with deployed apps.

Silmarillion can also achieve bidirectional interoperability with manual contact tracing [21]. Health authorities may manually obtain location data from consenting diagnosed individuals and insert records into Silmarillion. Thus, even users who do not carry a dongle can contribute to subsequent risk estimates and broadcasts. Conversely, by providing a user-comprehensible record of visited beacon locations, Silmarillion can be used as a diary aiding the memory of individuals participating in manual tracing. Finally, since Silmarillion can associate risk events with locations, information about potential superspreading events can be broadcast using traditional means of communication.

## 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 [5, 6, 10, 27] collect and manage user data centrally, placing a high degree of trust in the central authority. Decentralized SPECTS [9, 16, 28, 31] 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 [50]. Furthermore, an attacker could relay or replay captured ephemeral ids [26, 50]. 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. [43] 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 [25] 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 smartphones and low-end IoT devices, addressing technical challenges in achieving security, scalability and performance efficiency. PanCast assumes a pure broadcast-based risk dissemination architecture, which can be very expensive in terms of bandwidth and even latency during high infection rates. Silmarillion uses an IT-PIR based active querying protocol, thus minimizing bandwidth, latency, and power costs for user devices.

Systems that use QR codes [2, 12, 18, 39] 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 [19, 30], which is imprecise and invasive, or using encounters with WiFi access points [48], 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 [42]. 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. 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 [47] 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 user devices, particularly 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] Coronavirus disease (COVID-19): How is it transmitted? https://www.who.int/docs/default-source/coronaviruse/situation-reports/20200326-sitrep-66-covid-19.pdf. Accessed on 23 July 2023.

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

[3] Early Evidence of Effectiveness of Digital Contact Tracing for SARS-CoV-2 in Switzerland. https://github.com/digitalepidemiologylab/swisscovid_efficacy/blob/master/SwissCovid_efficacy_MS.pdf. Accessed on 22 October 2020.

[4] Nordic nRF52832 Development Kit. https://www.nordicsemi.com/Products/Development-hardware/nrf52-dk. Accessed on 4 August 2022.

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

[6] 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.

[7] SLBT010A EFR32BG22 Thunderboard Kit. https://www.silabs.com/development-tools/

thunderboard/thunderboard-bg22-kit?tab=overview. Accessed on 4 August 2022.

[8] SLWSTK6021A EFR32xG22 Wireless Gecko Starter Kit. https://www.silabs.com/development-tools/wireless/efr32xg22-wireless-starter-kit?tab=overview. Accessed on 4 August 2022.

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

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

[11] Why the WHO took two years to say COVID is airborne. https://www.nature.com/articles/d41586-022-00925-7. Accessed on 23 July 2023.

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

[13] Corona-Warn-App. https://www.coronawarn.app/en/, 2020. Accessed on 30 July 2023.

[14] COVID Alert. https://www.canada.ca/en/public-health/services/diseases/coronavirus-disease-covid-19/covid-alert.html#a6, 2020. Accessed on 15 Feb 2021.

[15] COVIDSafe Australia. https://www.abc.net.au/news/2020-06-02/coronavirus-covid19-covidsafe-app-how-many-downloads-greg-hunt/12295130, 2020. Accessed on 15 Feb 2021.

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

[17] NZ COVID Tracer. https://www.rnz.co.nz/national/programmes/checkpoint/audio/2018762292/2-point-1-million-download-covid-tracer-app-but-who-is-signing-in, 2020. Accessed on 15 Feb 2021.

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

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

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

[21] *Citation withheld for blind review*, 2022.

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

[23] 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.

[24] Apple. Privacy-preserving contact tracing. https://covid19.apple.com/contacttracing, 2020.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Philip S. Brachman. *Medical Microbiology*, chapter Epidemiology. Galveston (TX): University of Texas Medical Branch at Galveston, 4th edition, 1996.

[30] 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.

[31] 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.

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

[33] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable Anonymous Group Messaging. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[34] 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.

[35] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.

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

[37] 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.

[38] Google/Apple Exposure Notification (GAEN) system. Exposure Notifications: Using technology to help public health authorities fight COVID-19. https://www.google.com/intl/en_us/covid19/exposurenotifications/.

[39] 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.

[40] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[41] 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.

[42] 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.

[43] 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.

[44] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-Scale Distributed Graph Queries with Differential Privacy. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[45] Singapore Government. Safe Entry. https://www.ndi-api.gov.sg/safeentry. Accessed on 10 September 2020.

[46] The New York Times. In Coronavirus Fight, China Gives Citizens a Color Code with Red Flags. https://www.nytimes.com/2020/03/01/business/china-coronavirus-surveillance.html. Accessed on 03 August 2023.

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

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

[49] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*, 2015.

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

## A PIR Optimizations

In this section, we describe two optimizations on the basic PIR implementation described in §5.3.

Silmarillion's backend maintains a dynamic and hierarchical mapping ($\mathtt{D}^{\mathtt{PIR}}$) of group IDs to fixed-sized blocks containing the risk entries of for that group. Consider a three-level B$^+$-tree of group IDs $\mathcal{T} = \{\mathcal{T}^0, \mathcal{T}^1, \mathcal{T}^2\}$, where $\mathcal{T}^\ell$ represents the set of nodes at level $\ell$. The $\mathtt{D}^{\mathtt{PIR}}$ array maintains a mapping for each B$^+$-tree node to a PIR block, with the $\mathcal{T}^0$ nodes having lower indices than the $\mathcal{T}^1$ nodes, which in turn have lower indices than the $\mathcal{T}^2$ nodes. Suppose the group IDs are $n$-bit integers and the number of bits corresponding to the ID at each of the three levels are $\ell_0$, $\ell_1$, and $\ell_2$, respectively. The number of entries in $\mathtt{D}^{\mathtt{PIR}}$ equals $2^{\ell_0} + 2^{(\ell_0+\ell_1)} + 2^{(\ell_0+\ell_1+\ell_2)}$.

As described in §5.3, if the total number of risk entries in all the B$^+$-tree nodes at a level $j$ are less than the block size $B$, then the backend aggregates the risk entries into a single block that is mapped to the parent node at the level $j-1$. In this case, the $\mathtt{D}^{\mathtt{PIR}}$ may contain a large number of duplicate blocks for the nodes that map to the same block, 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.

**Database compression.** The $\mathtt{D}^{\mathtt{PIR}}$ contains entries for each B$^+$-node with the nodes at lower levels having lower indices than the nodes at higher levels. For nodes at a higher level (e.g., level $\ell_2$) that all map to the same block, the $\mathtt{D}^{\mathtt{PIR}}$ maps the block to the node at the higher level (e.g., level $\ell_1$), while the $\ell_2$ nodes point to a dummy zero-filled block. That is, $\mathtt{D}^{\mathtt{PIR}}[i'] \rightarrow B_j$ for index $i'$ corresponding to a node $\mathcal{T}^1_{\tau'}$ and $\mathtt{D}^{\mathtt{PIR}}[i] \rightarrow \phi$ for all indices $i$ corresponding to nodes $\mathcal{T}^2_{\tau}$ such that $\mathcal{T}^2_{\tau}$ is a child of $\mathcal{T}^1_{\tau'}$.

**Query compression.** 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 a B$^+$-tree node in $\mathcal{T}^2$, which does not correspond to any data block and suppose that the parent node in $\mathcal{T}^1$, $M$, does correspond to a data block. Let $\{j_{\mathcal{T}^2}\}$ be the set that contains all the indices of all the $\mathcal{T}^2$ nodes contained in $M$. Remember that if $M$ has not overflowed yet, all the nodes in $M$ are empty and point to the same block as $M$. In order to avoid unnecessary computations and overhead, Server 1 computes

$$Q_1[j_M] = Q_1[j_M] \oplus \left( \bigoplus_{j \in \{j_{\mathcal{T}^2}\}} Q_1[j] \right)$$

and then sets all $Q_1[j] : j \in \{j_{\mathcal{T}^2}\}$ to 0. Here, $Q_1[j_M]$ is the bit corresponding to the node $M$ in $Q_1$. The server performs this compression in the query $Q_1$ for each such node $M$ whose child nodes are unmapped. Effectively, the query now avoids dot products and XOR operations for the unmapped blocks and maps the operations to a single block mapped to the lower-level node. Server 2 performs similar operations on $Q_2$.

Subsequently, each server computes its answer share as usual. We are guaranteed that if the number of ones in the set $\{Q_1[\{j_{\mathcal{T}^2}\}]\}$ is even then the number of ones in the set $\{Q_2[\{j_{\mathcal{T}^2}\}]\}$ is odd and vice versa because $Q_1[i] \neq Q_2[i]$ and $i \in \{j_{\mathcal{T}^2}\}$. 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 the B$^+$-tree node in $\mathcal{T}^1$ $M$, i.e. the node in $\mathcal{T}^1$ that contains the $i$-th B$^+$-tree node in $\mathcal{T}^2$.

| $\varepsilon$ | $\delta$ = **0.001** | $\delta$ = **0.01** |
|---|---|---|
| 0.5 | 39098 | 29925 |
| 0.2 | 86969 | 64559 |
| 0.1 | 159131 | 115991 |
| 0.05 | 290088 | 210058 |

Table 2: 99th %ile noise required for various $\varepsilon$, $\delta$.

**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.

## B Proof of differential privacy of noise added to risk broadcasts

In Table 2, we first show the 99th percentiles of the number of noise entries required to achieve differential privacy with different levels of $\varepsilon$ and $\delta$. Below we prove the following differential privacy theorem, adapted from a similar theorem in the Appendix of [23].

**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 $\lambda$, truncated to the interval $[-t, \infty)$.[3] 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 $(\varepsilon, \delta)$-differentially private if:

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

*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 [36], it is enough to show that the function $g(x) = f(x) + \tilde{X}$ is $(\varepsilon, \delta)$-differentially private.

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

So, pick two adjacent inputs $x, x'$ and any output set $O$.[4] We need to show that

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

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 $\delta$ and $e^{\varepsilon} \Pr[g(x') \in O]$, 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 "$\delta$" 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 "$e^{\varepsilon} \Pr[g(x') \in O]$" 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 $\lambda$. 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 statement:

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

[4] 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.

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

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

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 $\Pr[X = o - f(x)]$.

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 $\Pr[X = o - f(x')]$. Continuing,

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

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

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

and, hence, that:

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

Combining everything we get the required differential privacy inequality:

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

□