# Silmarillion tech report

## ABSTRACT

We present Silmarillion, a novel, inclusive system for digital contact tracing and epidemic risk notification, which simultaneously provides utility as well as security. Silmarillion relies on a low-cost infrastructure of strategically placed beacons, inexpensive and low-maintenance user devices like dongles (if smartphones are inaccessible), and a backend that assists in epidemiological analysis and risk dissemination. Unlike today's smartphone-based contact tracing systems, Silmarillion records encounters between users' devices and beacons installed in well-known and strategic locations, which enables capturing contextual information relevant for epidemiological analysis, individual risk prediction, and prevention of spread of false risk information. Silmarillion keeps a user's encounter history local to their device and allows the user to control what information they share with the backend. Furthermore, it provides differential privacy for patients in risk dissemination and information-theoretic privacy for users receiving the risk information. We have evaluated a prototype of Silmarillion using small IoT boards and show that the battery and speed of risk dissemination is adequate for a practical deployment. Furthermore, we ran a small-scale deployment within a university building, demonstrating Silmarillion's practicality.

## 1. PANCAST TILING

### 1.1 High-Level tiling

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

### 1.2 Medium-Level tiling

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

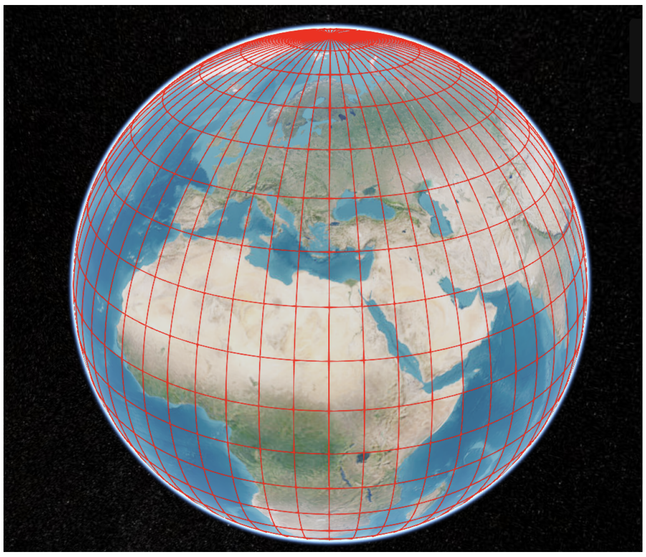
Figure 1: High-level tiling of earth's surface.

### 1.3 Low-Level tiling

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

## 2. Silmarillion'S PIR SCHEME

In this section, we present our complete PIR scheme. We start with a description of the basic scheme and then explain our optimizations.

We use an IT-PIR scheme based on ?. Our scheme relies on two servers, $S_1$ and $S_2$, each of which has a complete copy of the $\mathtt{DB}_{\mathtt{PIR}}$ $D$. Let $N$ be the number of blocks in $D$, each of which corresponds to a particular geographic L-tile, and let $B$ bits be the size in bytes of the largest data block in our dataset. In the basic protocol, we assume that all tiles are padded with dummy risk entries up to size $B$.

Suppose a dongle wants to query for the $n$-th block in our dataset. It generates two secrets shares of its query, $Q_1$ and $Q_2$: $Q_1$ is a random bit string of length $N$, and then it builds $Q_2$ as

$$Q_2[i] = \begin{cases} Q_1[i] \text{ if } i \neq n \\ \neg Q_1[i] \text{ if } i = n \end{cases} .$$

Then, the dongle sends $Q_1$ to $S_1$ and $Q_2$ to $S_2$. Since dongles do not have network connectivity, they rely on a network beacon or a trusted terminal to forward their queries to the PIR servers.

Then, each server computes its query answer as follows:

$$R_1 = \bigoplus_{i=0}^{N} Q_1[i] \cdot D[i]$$

and

$$R_2 = \bigoplus_{i=0}^{N} Q_2[i] \cdot D[i].$$

The query response blocks $R_1$ and $R_2$ are the exclusive-or of all the blocks whose corresponding indices in $Q_1$ and $Q_2$, respectively, are set to 1. Finally, the servers send back to the dongle the two query response blocks and the dongle XOR's the responses to retrieve the $n$-th block of the $\mathtt{DB_{PIR}}$. Effectively, the query answer $A$ is computed as:

$$A = R_1 \oplus R2 = \bigoplus_{i=0}^{N} (Q_1[i] \cdot D[i]) \oplus (Q_2[i] \cdot D[i])\,.$$

Now, $(Q_1[i] \cdot D[i]) = (Q_2[i] \cdot D[i])$ $\forall i \neq n$. Therefore,

$$(Q_1[i] \cdot D[i]) \oplus (Q_2[i] \cdot D[i]) = \begin{cases} 0 & \forall i \neq n \\ D[i] & \text{otherwise} \end{cases}$$

User's privacy is protected from the servers, since each server receives only a share of the query and iterates over the entire DB regardless of the query, and only the user can recover the DB block from the response shares of the servers.

We now describe two optimizations to make the PIR implementation efficient in the backend.

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

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

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

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

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

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

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

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

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

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

## 3. PIR DB

Recall from §??, Silmarillion dynamically generates a $\texttt{DB}_{\texttt{PIR}}$ based on static geographical tiling and the distribution of infection rates in the region of interest. Furthermore, it uses a separate $\texttt{DB}_{\texttt{PIR}}$ for each H-tile. For our experiment, we chose an H-tile over central Europe with a total of 434,448 tiles under it. We simulated a very high infection rate in 130,040 tiles, such that the maximal infection rate in a tile resulted in a block of risk entries of maximum size 5 MB. The total size of the resulting dataset is 16.58 GB.

## 4. CLOCK INCONSISTENCIES

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

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

### 4.1 Robustness analysis

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

Inconsistent encounters may arise in three ways: (i) the clocks of beacons or dongles are out of sync with real time; (ii) a beacon is misconfigured and placed at a location different from where it is registered; or, (iii) an illegitimate beacon re-transmits a legitimate beacon's transmissions at a different location. We discuss mechanisms to identify and mitigate inconsistencies in encounters reported to the backend[1].

**Clock inconsistencies.** Encounters become inconsistent when an ephemeral id is found to have been used for more than one epoch length in real time. This may happen when devices crash and reboot after a long time, leading to encounter timestamps that are out of sync with real clock time. The backend can detect and fix such an inconsistency if it receives at least two encounters of an inconsistent device with consistent devices, where one encounter occurred before and the other after the device's crash. That is, the backend can fix an inconsistent beacon if it receives at least two encounters with the beacon from consistent dongles, and similarly, an inconsistent dongle if it receives at least two encounters of the dongle with consistent beacons. We explain the mechanism for fixing inconsistencies in Appendix 4.

**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 **?**. All cases lead to the same inconsistencies due to the fact that the spoofed beacon is in a location different from where it is expected. First, dongles that travel between a spoofed beacon and nearby, legitimate beacons will appear to have traveled implausibly long distances in a short period of time. The backend can detect this when such a dongle uploads its history. Second, users with GPS-enabled smartphones can directly observe the problem when they see a beacon transmission with a signed location that is different from the phone's current location by more than the BLE range. Such phones may report the inconsistency to the backend.

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

**Tampering or withholding upload data.** In regimes mandating upload of the entire encounter history when diagnosed, users may attempt to evade the mandate by altering or withholding parts of their history. Modifying entries is impractical since it is infeasible to predict ephemeral ids without beacons' secret keys. Withholding entire periods of history will be easily detected; so a user may instead leave out all but one beacon's eph-

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

| **Operation** | **Lat. (s)** | **Current (mA)** | **Norm. current/h (mA)** |
|---|---|---|---|
| Legacy scan (31 B) | 0.001 | 3.77 | 0.1885 |
| Log append (48B) | 0.002 | 2.6 | 0.01 |
| CF hash (15 B) | $2 * 10^{-6}$ | 2.3 | $7.6 * 10^{-5}$ |
| Secret shares (700 Kbits) | 1.2 | 2.7 | $7.5 * 10^{-5}$ |
| Encryption (1 MB) | 1.3 | 2.5 | $7.5 * 10^{-5}$ |
| Decryption (1 MB) | 2.3 | 2.5 | $1.3 * 10^{-4}$ |
| LED blinking | 2 | 0.4 | 0.2 |

**Table 1: Latency and current draw of dongle ops TO revise**

emeral ids in each epoch to provide a contiguous history and evade detection. Such an attack can be performed in two ways: (i) by using a jammer to selectively prevent transmission of certain entries, or (ii) by tampering the dongle. (i) can be mitigated by encrypting and MAC'ing the entire upload payload as a single chunk, while (ii) can be made harder by using a strong tamper-proof hardware casing.

## 5. MITIGATION RISK BROADCAST ERRORS

PanCast risk information are encoded in cuckoo-filters, therefore, an error in receiving even a single BLE data packet would prevent a dongle from using the remaining filter for risk analysis. Let $F$ be the number of data packets in a cuckoo-filter and let $\rho$ be the packet error rate, the cuckoo-filter probability error, $p_C$ would be

$$p_C = 1 - (1 - \rho)^F.$$

Now, assuming that our risk information payload is formed by $N$ cuckoo filters, the payload error probability, $p_L$ would be

$$p_L = 1 - \sum_{e=1}^{n} \binom{n}{e} p_F^e \cdot (1 - p_F)^{n-e}.$$

Finally, the probability $P$ of needing $R$ rounds of retrasmissions to download a complete payload is

$$P = p_L^R \cdot (1 - p_L)$$

and, therefore,

$$R = \log_{p_L}\left(\frac{P}{1 - p_L}\right) = \frac{\ln \frac{P}{1-p_L}}{\ln p_L}.$$

Figure TO DO shows the number of rounds required to receive a $5MB$ risk information payload where each cuckoo-filter is formed by 2048 bytes.

## 6. POWER CONSUMPTION ANALYSIS

As described in §**??**, the dongle performs several operations: scanning on advertisement channels, scanning periodically on data channels, logging encounter entries to the dongle's flash storage, computing cuckoo filter hash indexes for log entries, and performing cuckoo filter lookups for log entries. For active querying, the dongle additionally needs to generate 4 secret shares per query, encrypt all shares, upload the queries and scan the responses on a connection-oriented channel, decrypt the response shares, and XOR the shares to retrieve the cuckoo filter for subsequent lookups.

Table 1 reports the average latency and current draw observed for each operation over 1 million observations. The last column reports the current draw normalized over a 1-hour period.

The dongle alternates between scanning for one minute and stopping for another minute. Within its 1-minute scan period, the dongle uses a scan operation duty cycle of 10%, i.e., , it scans for 100ms within a 1s interval. Thus, its normalized average current consumption is $0.1 * 0.5 * 3.77\ mA = 0.1885\ mA$.

Suppose a dongle encounters 1752 beacons in a short span of time in an area with dense beacon installation. The time required to fill the dongle's encounter log (1752 entries) is $1752 * 0.002\ s = 3.52\ s$. Thus, the worst case current consumption for encounter logging in a dongle, normalized over an hour is $\frac{2.6*3.52*4}{3600} = 0.01\ mA$.

Similar to encounter logging, the worst case current consumption for computing cuckoo filter indexes for all log entries, normalized over an hour is $\frac{2.3*1752*17\mu s*4}{3600} = 7.6 * 10^{-5}\ mA$.

Users typically query for risk information once a day. Therefore, the secret share and crypto operations are typically performed twice in 24 hours, once for each of the two shares. Thus, the current consumption for these operations, normalized for an hour is given by $(c * l * 2)/(24 * 3600)$, where $c$ is the current draw and $l$ is the latency of one instance of the operation. The normalized current values for secret share generation, AES-CBC encryption, and AES-CBC decryption are $7.5 * 10^{-5}\ mA$, $7.5 * 10^{-5}\ mA$, and $1.33 * 10^{-4}\ mA$, respectively.

Finally, the dongle LED blinks constantly with a 50% duty cycle. Therefore, the LED's current consumption normalized over one hour is $0.4\,mA * 0.5 = 0.2mA$.

| $\epsilon$ | $\delta$ **= 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 $\epsilon$, $\delta$.**

## 7. 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 $\epsilon$ and $\delta$. Below we prove the following differential privacy theorem, adapted from a similar theorem in the Appendix of **?**.

**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 $[-t, \infty)$.[2] 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 $(\epsilon, \delta)$-differentially private if:

1. $\lambda \geq A/\epsilon$, 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 **?**, it is enough to show that the function $g(x) = f(x) + \tilde{X}$ is $(\epsilon, \delta)$-differentially private.

So, pick two adjacent inputs $x, x'$ and any output set $O$.[3] We need to show that

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

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

$$\mathtt{Pr}[g(x) \in O] \;=\; \mathtt{Pr}[g(x) \in O_b] + \mathtt{Pr}[g(x) \in O \backslash O_b]$$

We now show that $\mathtt{Pr}[g(x) \in O_b]$ and $\mathtt{Pr}[g(x) \in O \backslash O_b]$ are bounded by $\delta$ and $e^{\epsilon}\,\mathtt{Pr}[g(x') \in O]$, 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 "$e^{\epsilon}\,\mathtt{Pr}[g(x') \in O]$" case of differential privacy.

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

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

We continue using assumption (2) of the theorem's statement:

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

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

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

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

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

[2]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 $\mathtt{Pr}[Q(\tilde{X})] = \mathtt{Pr}[Q(X) \mid X > -t]$ by definition of $\tilde{X}$.

[3]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.

simplifies to $\texttt{Pr}[X = o - f(x')]$. Continuing,

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

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

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

and, hence, that:

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

Combining everything we get the required differential privacy inequality:

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

□