# Trusted and Privacy-Preserving Sensor Data Onloading

Yin Liu[1]*, Breno Dantas Cruz[2], and Eli Tilevich[3]

[1] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China. `yinliu@bjut.edu.cn`

[2] Laboratory for Software Design, Iowa State University, USA. `bdantasc@iastate.edu`

[3] Software Innovations Lab, Virginia Tech, USA. `tilevich@cs.vt.edu`

**Abstract.** To personalize their services (e.g., advertisement, navigation, healthcare), mobile apps collect sensor data. Typically, they upload the collected sensor data to the cloud, which returns the inferred user profiles required to personalize mobile services. However, privacy concerns and network connectivity/congestion issues can render cloud-based processing inapplicable. If different apps collect the same type of sensor data, app providers can collaborate by combining their data collections to infer on-device the user profiles required for personalization. Although major mobile platforms provide on-device data sharing mechanisms, these direct data exchanges provide no privacy protection. As an alternative to direct data sharing, we present *differentially privatized sensor data onloading* for app providers' collaboration. With our approach, app providers can safely collaborate by using shared sensor data to personalize their mobile services. We realize our approach as a middleware that acts as a trusted intermediary. The middleware aggregates the sensor data contributed by individual apps, which execute statistical queries against the combined datasets. Furthermore, the middleware's adaptive privacy-preserving scheme 1) computes and adds the required amount of noise to the query results so as to balance utility and privacy; 2) introduces a Trust-Data Theory so as to detect and remove spurious data from the combined collections; 3) rewards active contributing app providers so as to incentivize data contribution; 4) integrates a Trusted Execution Environment (TEE) so as to secure all data processing. Our evaluation shows that it is feasible and useful to personalize mobile services while protecting data privacy: queries' execution time is within 10 ms; participants' dissimilar privacy/utility requirements are satisfied; untrustworthy data are effectively detected; mobile services are personalized, and data privacy of both app providers and users are preserved[4].

**Keywords:** data onloading, adaptive privacy preservation, sensor data, trusted middleware

## 1 Introduction

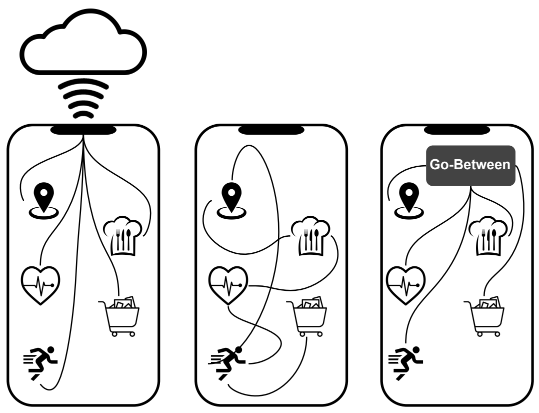


**Fig. 1.** Three Data Sharing Approaches.

Mobile services have become a crucial part of the digital economy [9], generating large and growing revenues for application providers [33]. Following the long-tail business model, app providers focus on personalizing their mobile services by constructing detailed user profiles, including inferred frequent routes, preferred activities, and daily body vitals, with services ranging between targeted advertising to healthy living tips [11]. To optimize personalization, app providers continuously collect sensor data by means of mobile apps, linked into data-sharing networks within the same device or across other media (e.g., clouds), thereby creating larger collections for constructing user profiles [18]. For example, numerous location-based apps (e.g., Google Maps, Uber, Yelp, and TripAdvisor) collect geolocations when each respective app is in operation. If the user frequents the same geolocations when using different mobile apps, these locations are "favorite," a piece of information that can be used to personalize location-based services.

* Corresponding author

[4] This article is a revised and extended version of our prior paper, published in the 12th EAI International Conference on Mobile Computing, Applications and Services (MobiCASE 2021) [62]

However, due to data privacy concerns, app providers often hesitate to share sensor data: their collaborators may accidentally expose or even intentionally disclose the shared data, damaging reputation and the bottom line [79,81]. *Since it is the end user who owns all device data, the app provider's privacy directly impacts user privacy.* That is, leaking the shared data threatens the privacy of app providers and users. Hence, there is a great need and potential benefit in providing holistic mechanisms for sharing sensor data that preserve the privacy of both app providers and users.

More importantly, when it comes to data sharing, bad actors may try misleading other contributors by intentionally contributing spurious data. A simple sanity check can quickly filter out physically impossible data contributions (e.g., systolic blood pressure >1k mm Hg); however, such checks would fail for attacks that contribute fake data within an expected range. Similarly, detecting by means of classic statistical criteria, such as Benford's law, 3-$\sigma$ rule, Chauvenet Criterion, and Dixon Criterion, either requires a specific data distribution or imposes restrictive conditions (e.g., only one outlier exists). Modern anomaly detection (i.e., outlier detection) uses machine learning or deep learning algorithms to identify outliers that deviate from the general data distribution [22, 28, 50, 60, 77, 84] and has been applied to many research areas (gaze estimation [23], cyber-physical systems [19], wireless sensor networks [8], data streams [78], and Internet of Things [54]), albeit suffering from false positives/negatives. Besides, a detector cannot distinguish whether the outliers come from an intentional (i.e., maliciously adding fake data) or unintentional (e.g., collecting data from an inaccurate sensor) operation, so none of the existing approaches can detect all data outliers. Thus, to mitigate the threat of spurious data contribution, this problem domain requires new and effective solutions that allow data contributors to safely collaborate while preserving user privacy.

To share the sensor data collected by their mobile apps, app providers can use cloud-based services (the left-most option in Figure 1). Each app uploads its collected data to the cloud, which aggregates and analyzes the results. Although the state of the art leverages attribute-based encryption [59] and blockchain [87] to help preserve user privacy in data sharing, data privacy preservation remains an open problem in the cloud-based data sharing process, which incurs various concerns: (a) cyber attackers can steal uploaded data by exploiting the cloud server's vulnerabilities [1–7]; (b) insiders or careless employees can expose private data to the public [95]; (c) governments can legally force IT companies to reveal their cloud-stored data [92]; (d) network connectivity/congestion issues can easily render cloud-based processing infeasible, and the high overheads of current cloud-based privacy-preserving solutions (e.g., blockchain, attribute-based encryption) further worsen the availability and feasibility of cloud-based data sharing. In fact, a growing number of privacy tips recommend disabling cloud-based storage and processing altogether with restrictive network access permissions [43] and network blocking apps [45–47].

Mobile apps can also share their sensor data locally on the same device (i.e., the middle option in Figure 1). This on-device data sharing and processing—referred to as *data onloading*—has been studied widely in the research literature [49, 61, 63, 94, 97] and adopted in industrial settings. In fact, major mobile platforms do provide standardized mechanisms for the installed apps to share data locally (i.e., "App Groups" [12] in iOS; `Intent`, `SharedPreferences`, and `ContentProvider` in Android). However, these mechanisms are designed for apps to exchange data directly. As such, they are vulnerable to privacy exploits: the receiver apps can be leaking the received data unwittingly or intentionally. An alternative is for mutually distrustful app providers to discover the commonalities of their data collections (i.e., obtain data intersections) via encryption-based Private Set Intersection (PSI) [52]. However, intersections alone are hardly ever sufficient to infer the profiles of mobile users.

Another alternative is to keep the exchanged data private, while permitting the querying of its statistical properties [97]. Unfortunately, this alternative's vulnerabilities can be exploited. For example, exhaustive frequency queries over a complete finite set can exfiltrate the other contributor's data. A differential privacy mechanism can be applied to alleviate such risks (e.g., PINQ [69], GUPT [75]). However, one cannot directly apply differential privacy due to the unique challenges of our problem domain: 1) how to assign privacy levels to all collaborators (i.e., app providers) that may have dissimilar privacy/utility requirements; 2) how to prevent some app providers from contributing only minimal data but inferring lots of user profiles from the combined datasets; 3) how to defend against attacks that lead to data and operations being illicitly accessed or tampered with.

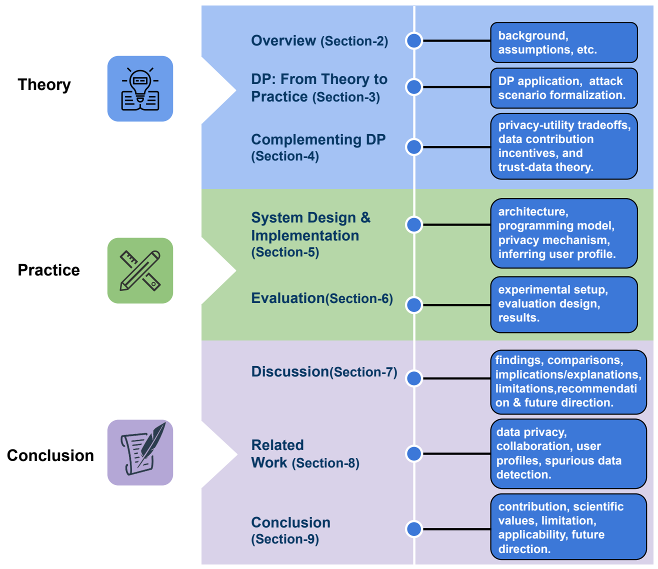
To overcome the above challenges, we present a trusted middleware for privacy-preserving sensor data onloading, serving as a trusted intermediary that aggregates the sensor data contributed by the collaborating apps and executes expressive statistical queries against the inaccessible combined datasets (i.e., the right-most option in Figure 1). By introducing a *trust-data theory*, our approach detects and removes spurious data from the combined collections. Besides, it also achieves the *privacy-utility tradeoffs* that satisfy given privacy/utility requirements, incentivizes app providers to keep contributing data, and secures the execution of these query functions by placing them in a Trusted Execution Environment (TEE), whose trusted storage persists the shared data collections.

We target the dominant mobile platform (≈85% of the global mobile market [53]), the Android platform, on which apps commonly share data with each other [17,91]. The reference implementation of our approach—Go-Between—offers a system-level service that aggregates into combined datasets the data contributions of the collaborating apps, which can then query the service to infer user profiles. By adapting differential privacy for our problem domain, Go-Between adds adaptively customized *Laplace noise* to the query results, thus properly preserving app providers' data privacy. Significantly, by applying a new theoretical detection model, Go-Between detects and removes spurious data contributions from the combined dataset. Besides that, Go-Between balances collaborating apps' dissimilar utility/privacy requirements (i.e., privacy can be increased at the cost of decreasing utility and vice versa.) Further, Go-Between incentivizes data contributions: the more data an app contributes, the more accurate and useful its inferred user profiles are. Moreover, Go-Between applies TEE to the predefined statistical queries (e.g., `Count`, `Mean`, and `Std`), so as to safeguard the operations and their data. Finally, Go-Between keeps the end-user in control of their data by informing them of the data sharing events and explicitly allowing them to restrict apps to share data. The contributions of this article are as follows:

1. A trusted middleware for **differentially privatized** onloading of sensor data that is:
   - ***(a) usable:*** it dynamically adapts and balances privacy/utility, as driven by the properties of the contributed data;
   - ***(b) resilient:*** it detects and removes spurious data contributions from the combined dataset;
   - ***(c) incentivizing:*** it rewards active contributing app providers with higher utility;
   - ***(d) secure:*** it protects all operations and the contributed data in a Trusted Execution Environment (TEE).
2. A general system design for privacy-preserving data onloading, whose building blocks include differential privacy and TEE. The applicability of this design extends beyond our target domain.
3. A reference *implementation*—Go-Between—an Android system service, empirically *evaluated* to demonstrate its efficiency, utility, and safety: all queries execute in $< 10$ ms; mobile services are effectively personalized, while preserving app providers' privacy.

This article extends our earlier conference paper, presented at the 12th EAI International Conference on Mobile Computing, Applications and Services (MobiCASE 2021) [62]. In comparison to that conference publication (15-page, single-column), this article reports on additional unpublished research that extends our prior work as follows:

(1) We introduce Trust-Data Theory, which we created to formalize the description of our mitigation strategies for the threat of contributing spurious data. We concretely apply this theory to create a mechanism for detecting and removing spurious data. Furthermore, by simulating an attack of contributing spurious data, we validate that our theory and its reification can effectively defend against such attacks.


Theory
Practice
Conclusion
Overview (Section-2)
DP: From Theory to Practice (Section-3)
Complementing DP (Section-4)
System Design & Implementation (Section-5)
Evaluation(Section-6)
Discussion(Section-7)
Related Work (Section-8)
Conclusion (Section-9)
background, assumptions, etc.
DP application, attack scenario formalization.
privacy-utility tradeoffs, data contribution incentives, and trust-data theory.
architecture, programming model, privacy mechanism, inferring user profile.
experimental setup, evaluation design, results.
findings, comparisons, implications/explanations, limitations,recommendation & future direction.
data privacy, collaboration, user profiles, spurious data detection.
contribution, scientific values, limitation, applicability, future direction.
**Fig. 2.** The roadmap of the article.

(2) We explain how our programming model, with its reactive/functional programming interfaces, enables Android developers to seamlessly add privacy-preserving sensor data onloading to their mobile apps.

(3) We also evaluate and report on our approach's programmability by analyzing two widely-used software metrics among our approach and similar Android system services.

(4) We formulate an Honest-But-Curious attack as a differential privacy problem and simulate such an attack as a control group; we show how without our approach's privacy preserving mechanism, an attacker can always reconstruct the combined dataset by executing particular queries.

**Roadmap:** As shown in Figure 2, our research methodology proceeds from theory to practical applications, completing with discussion and conclusions. Our theoretical contribution adapts differential privacy for a new problem domain; our practical application introduces the design, implementation, and evaluation of our approach in practice; our discussion contextualizes our approach and its evaluation results.

Specifically, the rest of this article is organized as follows. Section 2 discusses our application scenarios, threat model, and solution overview. Section 3 presents how our approach applies differential privacy. Section 4 provides our mechanisms to complement differential privacy, including privacy&utility tradeoffs, data contribution incentives, and trust-data theory. Section 5 details design and implementation of our approach. Section 6 presents our evaluation results. Section 7 discusses app provider's data privacy, and our approach's applicability. Section 8 compares our work to the related state of the art. Section 9 presents concluding remarks.

## 2 Go-Between Overview

To motivate our approach, we present two typical application scenarios and how Go-Between addresses their requirements. Then, we give an overview of differential privacy and key technologies used by Go-Between.

### 2.1 Typical Application Scenarios

**I. Geolocations** can be used to infer a user profile's location-based properties (e.g., favorite areas). To optimize personalization, app providers frequently collect and share geolocations. An empirical study has revealed how within 14 days 10 different mobile apps, not only mapping and navigation, but also social media (e.g., Facebook) and shopping (e.g., Groupon) [10], shared geolocations 5,398 times.

Consider a navigation app N that collects the user's geolocations to provide real-time traffic information. On the same device, a shopping app R records the user's geolocations independently to learn about the frequently visited areas in order to recommend shopping and dining options. Finally, an exercise app E collects the geolocations of the user's regular running routes. Since all three apps collect geolocations for different purposes, their providers may want to personalize their services, as informed by the combined dataset of their respective collections of geolocations. By querying the combined dataset (e.g., how many times the user visited a given area?), each provider can identify the user's "favorite" areas. This information can improve how each app provider tailors its services for the user, such as displaying ads specific to the favorite areas.

**II. Body vitals**, another common type of sensor data, enables app providers to infer a user's health condition. Typical body vitals include temperature, pulse rate, and blood pressure. Health wearables and trackers continuously collect body vitals, sending them for processing and storage to paired devices with specialized apps. For example, a smartwatch or a blood pressure monitor would record a user's blood pressure, with the records transferred to an app running on the user's mobile phone [74, 80]. A mobile app can also receive body vitals from its user's healthcare provider. For example, a recent news report points out that a healthcare record can now be downloaded to its user's mobile apps, so their providers can potentially share the downloaded records with healthcare providers and insurers [89].

Consider a blood pressure monitor app M that periodically measures and records the user's blood pressure. A smartwatch app W records the user's blood pressure at specified intervals. A personal health records app H keeps track of the user's blood pressure readings, taken during doctor's appointments. Since all these three apps collect blood pressure readings, analyzing the combined dataset of their respective collections can provide additional value to the user. For example, the frequency, the mean, and the standard deviation of all the collected readings can indicate a possible hypertension condition rather than experiencing occasional spikes of high blood pressure (due to stress).

### 2.2 Solution Overview

App providers[5] specify their privacy and utility requirements (e.g., high privacy/medium utility), and then deposit their sensor data (e.g., geolocation/blood pressure datasets) with Go-Between, which aggregates the deposited data into combined datasets for app providers to query. Go-Between differentially privatizes the query results in accordance to both the properties of the deposited data and the specified requirements. Through these queries, the collaborating app providers then personalize their mobile services, without revealing their raw sensor data to their collaborators.

Specifically, an app first secures a user's permission to deposit a certain type of sensor data with Go-Between, which maintains a trusted record of all user-authored apps/data types. Any permitted app can query the combined dataset of the deposited sensor data type. The apps collaborate via a four-phase process: (1) apps specify their privacy and utility requirements and transfer their sensor data to Go-Between[6]; (2) upon each data deposit, Go-Between starts computing the noise scale for each built-in query operation, while detecting and removing spurious data from the combined dataset; (3) the collaborating apps *black-box query* the combined dataset to infer the user's profile; (4) Go-Between pads the query results with a suitable amount of noise, determined by the pre-computed noise scale, and returns them.

### 2.3 Threat Model

Since the user owns all the collected data, the privacy of *app providers* is an integral part of *user privacy.* Nevertheless, *app providers* and *users* incur different data privacy threats, which we discuss in turn next:

***I. App Providers.*** The process of app providers depositing their sensor data is subject to the following threats:

(a) to optimize mobile service personalization, every app provider strives to get access to as much sensor data as possible. To that end, a provider could attempt to extract their collaborators' raw data from the combined datasets. This behavior is described by a classical threat model—*honest-but-curious attack* [83]: an adversary tries to legally learn all possible information about the combined datasets.

(b) to prevent the above attack, some app providers may limit their data contribution as much as possible, while taking advantage of their collaborators by inferring user profiles from the combined datasets.

(c) to illicitly obtain the collected sensor data, malicious parties may perpetrate attacks to access the combined datasets.

In all three threats above, a dishonest app provider or an attacker assumes the deposited data are entirely accurate and real, so if they illicitly access the combined dataset, they would benefit as a result. However, this assumption does not hold once some bad actor has deposited fake data into the dataset:

(d) to gain an unfair business advantage, some app providers may deposit inaccurate or outright fake data, thus impairing the utility of the combined datasets.

For example, a navigation app could intentionally put fake locations into the combined dataset in order to paralyze its competitors' navigation services that rely on the dataset.

***II. Mobile Users.*** Irrespective of how app providers deposit sensor data, mobile users deeply care about (e) which part of their data will be used and which app providers are involved in the process of data sharing. Specifically, when a mobile app queries a Go-Between's combined dataset, the mobile users are eager to find out what kind of data will be queried and returned. In the meantime, they also care about which mobile apps (i.e., app providers) are querying and exchanging their collected data with Go-Between.

***III. Countermeasures.*** To ensure data privacy of app providers, we introduce the following countermeasures:

(a) to defend against *honest-but-curious attacks*, all query results are differentially privatized, so the participating app providers cannot recreate the combined datasets (§ 3§ 4.1).

[5] An app provider can have multiple apps, while an app has one provider only. For ease of exposition, we assume a one-to-one correspondence between a provider and an app, and use terms "app provider"/"app" interchangeably.

[6] Each data type has its own combined dataset.

(b) to discourage limited data sharing, a query result's accuracy is positively correlated with the size of the querier's data contribution, thus incentivizing large-scale sharing (§ 4.2)

(c) to prevent the combined datasets from being illicitly accessed, all above operations and the deposited data take place in a Trusted Execution Environment (TEE) (§ 5.1).

(d) to mitigate the threat of contributing inaccurate/fake data, such spurious data is detected/removed from the combined datasets, as informed by our *trust-data theory* (§ 4.3)

(e) to keep the user in control, all sharing-related information (the list of apps, the data type, and query, etc.) can be routed to the user for examination and approval. The user can opt out from receiving this information.

### 2.4 Enabling Theory & Technologies

**I. Differential Privacy(DP) [29]** protects an individual's private information from unauthorized discovery (*hereafter,* individual *refers to* an app provider*, and* private information *refers to the sensor data collected by a provider.*) More formally, a *database* $D$ is a database of records in a *data universe* $U$. Each record contains an individual's private data. Differential Privacy defines two databases $D$ and $D'$ as *neighboring databases* if they differ by exactly one record. A *mechanism* $M$ is a randomized function that maps $D$ to output $R$.

**Definition 1: $\varepsilon$-differentially private mechanism.** Given $\varepsilon \geq 0$, $M$ is $\varepsilon$-differentially private, iff for all neighboring databases ($D$, $D'$), and for any sets of outputs $S \subseteq R$:

$Pr[M(D) \in S] \leq e^{\varepsilon} Pr[M(D') \in S]$ (1)

**Definition 2: sequential composition.** Given a set of *mechanisms* $M = M_1, ..., M_n$, sequentially executed on a database, with each $M_i$ providing $\varepsilon_i$-differential privacy guarantee, the total guarantee provided by $M$ is:

$\sum_{i=1}^{n} \varepsilon_i$ (2)

**Definition 3: global sensitivity.** For a query $f : D \to R$; $D$,$D'$ are *neighboring databases*, the global sensitivity of $f$ is:

$\Delta f = Max_{D,D'} \, |f(D) - f(D')|$ (3)

The value of $\Delta f$ (i.e., global sensitivity of $f$) indicates the maximal difference between the query results on $D$ and $D'$.

**Definition 4: upper bound of $\varepsilon$ [58].** Given a database $D'$ with $n-1$ records sampled from $D$ (i.e., $D' \subset D$ and $|D'| = |D| - 1$), the probability of discovering the record in the database $D$ (i.e., $\rho$), the number of records ($n$), the global sensitivity of query $f$ (i.e., $\Delta f$), and the maximal difference between query results of each possible combination of $D'$ (i.e., $\Delta v$):

$\varepsilon \leq \frac{\Delta f}{\Delta v} ln \frac{(n-1)\rho}{1-\rho}$ (4)

**II. Laplace Mechanism** [30, 31] adds independent noise to the actual query results. $Lap(\mu, b)$ represents the noise sampled from a *Laplace Distribution* with the scale factor of $b$ and location factor of $\mu$. The *Laplace distribution* [56] is a double exponential distribution, in which the scale factor $b$ is positively correlated with the amplitude, thus determining the confidence level in the noisy results. Briefly, $b$ determines the amount of *Laplace noise* to add. Usually, we omit $\mu$ and use $Lap(b)$ as the added noise.

**Definition 5 — noise scale.** To satisfy $\varepsilon$-differential privacy for query $f$, use scaled symmetric noise $Lap(b)$ with $b = \Delta f/\varepsilon$, that is:

$Lap(\Delta f/\varepsilon)$ (5)

By setting the location factor of $\mu$ with the actual result of query $f(D)$, we can get the privatized value: $f(D) + Lap(\Delta f/\varepsilon)$ that ensures the $\varepsilon$-differential privacy.

**Definition 6 — noise scale for a query sequence.** To satisfy $\varepsilon$-differential privacy for a query sequence $f_1, ..., f_n$, use scaled symmetric noise:

$Lap(\sum_i \Delta f_i/\varepsilon)$ (6)

**III. Trusted Execution Environment (TEE)** [35] provides hardware support for handling sensitive data. TEE (1) partitions the CPU into the normal world for common applications and the secure world for trusted applications; the secure world prevents external entities without authorization from accessing trusted applications; (2) provides trusted storage to persist sensitive data, which can only be accessed via the provided API; (3) provides a secure communication channel for external peripherals. *Open-TEE* [67] virtualizes TEE via a software framework. By conforming to the GlobalPlatform

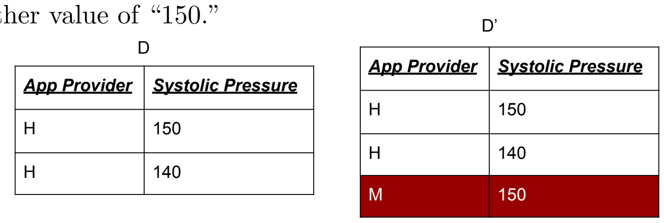
Specifications of TEE, Open-TEE hosts trusted applications, in lieu of a hardware-based TEE. Known as an efficient “virtual TEE,” Open-TEE features small storage and memory footprints as well as short start and restart latencies for the trusted applications.

## 3 DP: From Theory to Practice

In this Section, we first explain by example how we apply differential privacy (DP) to defend against the aforementioned honest-but-curious attacks.

***Honest-but-curious attacks.*** We further develop the scenario in § 2.1 that deposits body vitals. Consider the worst-case scenario: only two apps—H and M—deposit their collected blood pressure readings. As shown in Figure 3, H stores its blood pressure readings into the combined dataset (i.e., D — the table on the left). Then, H queries for the frequency of “150”, which returns “1”, as“150” occurs only once in the combined dataset. After that, M adds one more reading of “150” to the combined dataset (i.e., D’ — the table on the right). Then, H repeats the same frequency query on the updated dataset, getting “2” as the result, meaning that “150” now appears twice. In this worst-case, H may also discover that M has stored its dataset between H’s two frequency queries. Armed with this fact, H can determine it was M that stored the other value of “150.”

***Counter-measuring with DP.*** Consider how DP can be applied to defend against such honest-but-curious attacks. The worst-case scenarios above can be formalized as a differential privacy problem (see the formalization below). To put it briefly, a DP mechanism would pad each query result with noise. As an illustration, assume that H’s first and second queries are padded with the noise amounts of “0.6” and “-0.5”, respectively, so the final results would become “1.6” (i.e., 1 + 0.6) and “1.5” (i.e., 2 - 0.5), respectively. These fractional results about the frequency of “150” in the combined dataset still provide useful information (e.g., “1.6” and “1.5” are between 0 and 2). However, now H can no longer infer if M has contributed “150” to the dataset.



D

| App Provider | Systolic Pressure |
|---|---|
| H | 150 |
| H | 140 |

D'

| App Provider | Systolic Pressure |
|---|---|
| H | 150 |
| H | 140 |
| M | 150 |

**Fig. 3.** The worst-case scenario of the attack.

***Formalizing an attack scenario.*** As shown in Figure 3, let $D$ denote the combined database of blood pressure readings contributed by H, and $D'$ denote the combined readings database after M inserts one record. Record $x$ denotes the delta between $D$ and $D'$, such that $D = D' - \{x\}$ (in our case, $D = D' - \{150\}$). H can perform any number and kind of legitimate queries against $D$ (such as the frequency query above). In addition to the information obtained through the legitimate queries, we assume that H also possesses additional background knowledge (e.g., H discovers that M stores a record between H’s two frequency queries). Hence, H can act as an adversary that attempts to extract both the raw data content of $D$ and determine which party has contributed which data elements. To discover $x$, H queries $D$ and $D'$. It can do so by executing the same query on $D$ and $D'$ and computing the delta of the results. The other contributors’ privacy becomes breached, as the adversary learns their raw data.

## 4 Complementing DP

In this Section, we discuss how we complemented DP to meet the privacy requirements in our target domain.

### 4.1 Privacy & Utility Tradeoffs

As discussed in § 3, differential privacy can prevent the potential breaches described in our threat model by adding the *Laplace noise* to the query results to obtain an $\varepsilon$-differential privacy guarantee. However, *the resulting noise scale must balance the tradeoffs between privacy and utility.* The former represents how much noise to add, while the latter indicates how usable the noisy results are for inferring user profiles. Privacy/Utility are negatively correlated: the higher is the level of privacy, the lower is utility, and vice versa.

**I. Privacy.** Definition (4) determines the upper bound of $\varepsilon$, and Definition (5) shows the noise scale. By combining (4)(5), we obtain the lower bound of scaled noise $Lap(b)$ with:

$b = \Delta v / ln \frac{(n-1)\rho}{1-\rho}$ (7)

As per Definitions (4) and (5) (discussed in § 2.4), $\rho$ is the probability that the adversary can correctly guess the absence/presence of a record in the combined dataset. $n$ is the number of records. $\Delta v$ is the maximal difference between the query results of each possible combination of $D'$ (i.e., the neighboring database discussed in § 2.4-I). Thus, $n$ and $\Delta v$ can be calculated based on the dataset's properties. $\rho$ can be configured by apps in order to control the privacy level based on their specific requirements.

**II. Utility.** *How accurate the query results are* and *how frequently the query is executed* determine utility:

*a) For accuracy*, we define the *accuracy level* ($a$) as the distance between the actual query result and the result with noise. We determine $a$ via the *percent error* formula:

$a = 100 \cdot \left| \frac{Result_{noise} - Result_{actual}}{Result_{actual}} \right|$ (8)

$Result_{noise}$ is the query result with noise, and $Result_{actual}$ is the actual query result. The collaborating apps can set the required accuracy level (i.e., a). After adding noise, if the result of a query's accuracy level cannot meet the level set by the app, the query fails.

*b) For usage frequency*, we define the *usage frequency level* ($u$) as the invocation number of a certain query. Based on the Definition 2, for example, if an app performs a query (providing $\varepsilon$-differential privacy guarantee) 10 times, then the query's total differential privacy guarantee is $10 \cdot \varepsilon$. Each collaborating app can configure its usage frequency level, used to adjust the noise scale. See *Noise Increase Scheme* (§ 4.2-III) below for details.

## 4.2 Data Contribution Incentives

For app providers to be willing to keep contributing data to Go-Between, three conditions must be met:

- The privacy level $\rho$ should be a parameter shared across all collaborating apps. If the specified privacy level affects only the app that specifies it, the resulting perverse incentive would suggest specifying the lowest privacy level to obtain the highest utility.
- The amount of contributed data should be commensurate with the obtained utility.
- The more an app queries Go-Between, the more noise should be added to its privatized query results.

To meet above conditions, we introduce *global privacy level*, *bonus mechanism* and *noise increase scheme*, respectively.

**I. Global Privacy Level:** For each collaborating app, we define a *contribution rate* ($c$):

$c_i = \frac{\omega_i}{\sum \omega_i}$ (9)

where $\omega_i$ is the amount of data contributed by the $i$th app.

By weighting the *average* value of app-configured privacy levels by their contributed data's amount, Go-Between determines the *global privacy level*:

$\rho_{global} = \sum c_i \rho_i$ (10)

where $\rho_i$ is the *privacy level* configured by the $i$th app.

The *global privacy level* is used to calculate the noise scale ($b$) by using (7) (discussed in § 4.1). That is, the more data an app contributes to a combined dataset, the higher the impact of the app's privacy level on the overall global privacy level. This design prevents apps with only a small data contribution from specifying the lowest privacy level with the goal of accurately inferring user profiles.

**II. Bonus Mechanism:** We establish a bonus mechanism that reduces the noise scale (i.e., increases the accuracy) for apps in proportion to the amount of their contributed data. To that end, Go-Between selects 10% of a given query's noise scale as the bonus: $BONUS = 10\% \cdot b_{query}$, where $b_{query}$ is the query's noise scale. When adjusting $i^{th}$ app's noise scale ($b_i$), we use the app's *contribution rate* ($c$) to calculate its bonus: $c_i \cdot BONUS$, which is subtracted from the noise scale:

$b_i = b_{query} - c_i \cdot BONUS$ (11)

**III. Noise Increase Scheme:** Since every query invocation accumulates $\varepsilon$ (Definition 2), the likelihood of an attacker discovering the raw dataset is positively correlated with *usage frequency level*

$(u)$ (i.e., the number of query invocations). To reduce the risk of such discovery, Go-Between scales $b_i$ up by $u_i$ times (i.e., $b_i * u_i$). By increasing the noise scale proportionally to the number of query invocations, Go-Between thus maintains the $\varepsilon$-differential privacy.

## 4.3 Trust-Data Theory

Determining the trustworthiness of data contributions is a hard problem. Existing solutions — sanity checks, statistical criteria (e.g., Benford's law, 3-$\sigma$ rule, Chauvenet Criterion, Dixon Criterion), and modern anomaly detection [50,82] — suffer from false positives/negatives. Besides, none of these solutions can distinguish whether an outlier was caused by an intentional (i.e., fake data) or unintentional (e.g., inaccurate sensor reading) contribution.

For our target domain, we present *trust-data theory* that reconsiders spurious data as a special kind of noise to enhance our privacy model. Although contributing spurious data perpetrates an attack, we recast this attack to perturb the combined dataset in order to achieve the required privacy/utility tradeoff. Further, we develop a statistical testing method that determines whether app-contributed data, irrespective of its trustworthiness, protects privacy.

**I. Using spurious data.** We observe that *spurious data can be used to preserve data privacy.* Consider how differential privacy (DP) works: compute a noise value, add it to the actual data, and return the result. That is, after adding the noise, differential privacy converts the "actual data" to the "spurious data" to protect data privacy. Since the contributed "spurious data" changes the original dataset's distribution within an acceptable range, DP still provides useful query results. Hence, unless the newly contributed data completely changes the original dataset's distribution, the data contribution, irrespective of its trustworthiness, can be accepted. In the following discussion, we explain how we determine the threshold at which the contributed data changes the original dataset's distribution.

**Definition 7: Trust Sensitivity (TS).** For a query $f : D \to R$; $D_{original}$ is the original database, $D_{new}$ is the new database including the newly contributed data from a collaborating app; based on $D_{new}$ and $D_{original}$, $b_{new}$ and $b_{original}$ are the noise scale factors computed by the aforementioned equations (3)(4)(5) in § 2.4-Definitions 3-5. $TS$ is:

$$TS = Overlap\{Lap(b_{new}), Lap(b_{original})\} \quad (12)$$

Figure 4 shows that the value of $TS$ (the hatched area[7]) is the overlap between the *Laplace Distribution* with the scale factors of $b_{new}$ and $b_{original}$, which indicate the similarity of the privatized results before and after adding the new data (irrespective of its trustworthiness). That is, the larger $TS$ is, the more similar the privatized result is. A larger $TS$ indicates that the newly contributed data only unexcessively impacts the privatized result and thus can be accepted. The algorithm of computing $TS$ is discussed in Appendix-B.

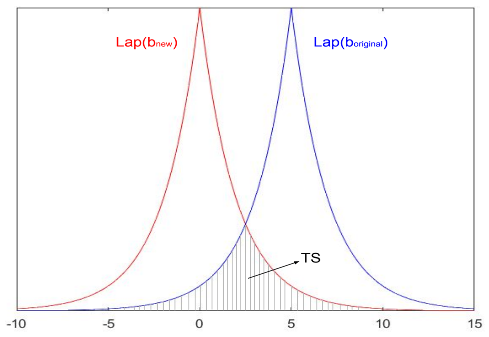


**Fig. 4.** The Go-Between Trust Sensitivity.

**Definition 8: $\alpha$-trust data.** For $0 \leq \alpha \leq 1$, the newly contributed data $\Delta D$ (i.e., $\Delta D = D_{new}$ - $D_{original}$) is $\alpha$-trust if $TS \geq \alpha$. In contrast, If $TS < \alpha$, $\Delta D$ is untrustworthy.

[7] We compute probability density function's $\int$ to obtain overlapped area.

*$\alpha$ is a threshold (i.e., lower-bound of $TS$): the smaller $\alpha$ is, the less stringent the trust condition is; the more privacy is preserved, the less utility is provided.* That is, having a smaller $\alpha$, a larger amount of untrustworthy data can be accepted. In the extreme (i.e., $\alpha = 0$), data privacy is well-protected if all the data is untrustworthy. By using $\alpha$, Go-Between determines whether the newly contributed data is acceptable. The value of $\alpha$ can be decided differently. By default, Go-Between sets $\alpha$ to each app's reputation score.

**II. Determining the value of $\alpha$.** The app reputation score metric has been widely applied to app markets (e.g., Google Play [40]). In addition, many anti-malware providers, such as McAfee, Trend Micro, and Sophos Mobile, develop their own app reputation systems and report the apps with low reputation scores as posing a high security and privacy risk [66, 90, 93]. The app reputation score systems provide a straightforward guide to distinguish which apps can be trusted. We also assume that, by employing crowdsourcing and sophisticated program analyses, modern reputation management systems would be unlikely to report the High trust level for any malware. Hence, we set the value of $\alpha$ to each app's reputation score (e.g., the score can be collected from the Google Play). Specifically, for the collaborating apps with low scores, we assign a high $\alpha$ value, i.e., only accept the newly contributed data that preserves the original privatized result's distribution; for those with high scores, we assign a low $\alpha$ value, i.e., accept their contributed data even if it may change the privatized result's distribution (see § 5.3-III).

**III. Defending progressive data distribution attack (PDDA).** Determining whether the newly contributed data ($\Delta D$) is trustworthy proceeds as follows. Based on the supported queries' noise scale, calculate *Trust Sensitivity* ($TS$) and compare it with the data contributor app's threshold $\alpha$. However, consider the contributor app sharing a small size of $\Delta D$ (in the extreme, only one record in $\Delta D$). Obviously, growing the deposited dataset by a small chunk is unlikely to impact the overall distribution. Hence, for large original datasets, $\Delta D$, in all likelihood, will be recognized as trustworthy data. Then, an attacker can continuously grow the dataset by contributing small chunks until the dataset's distribution is impacted. We call this attack *progressive data distribution attack* (*PDDA*).

To defend against *PDDA*s, Go-Between applies *statistical sampling*: before calculating the noise scales for the new and original datasets, it first compares their sizes, samples the larger one, creating a sampled dataset of the *same size*. In addition, to ensure the reliability of data distribution, Go-Between requires that the newly contributed data ($\Delta D$) must contain at least 20 items (i.e., smaller datasets are not accepted as contributions)[8]. Because $TS$ is calculated from the sampled datasets (i.e., containing the same number of items), malicious contributors would be unlikely to succeed in growing the combined dataset by contributing small chunks without impacting the overall distribution. For example, consider the original dataset of size 100,000 and $\Delta D$ of 20. Go-Between randomly selects 20 records from the original dataset as a sampled dataset, and uses this sampled dataset and $\Delta D$ to calculate $TS$. That is, although the original dataset's size of 100,000 is much greater than the $\Delta D$ of 20, the $TS$ is calculated from the sampled dataset of size 20 (i.e., the same as $\Delta D$), thus defeating PDDAs. We detail all the above mechanisms in § 5.3 below.

## 5 System Design & Implementation

Our approach is reified by the Go-Between framework, whose design and implementation we describe in turn next.

### 5.1 Architecture

`Mobile Apps` configure their privacy and utility requirements, persist their individual collections of sensor data, and perform predefined queries on the combined dataset via the `Go-Between API` (step 1). The `API` interacts with `Go-Between service` (step 2), a system-level Android service that encodes the data via the `Encoding Protocol` (step 3) and executes service calls (i.e., query, persist, and configuration) via `Native Interface` (step 4). Finally, the data is passed to TEE to be securely operated on (step 5).

[8] As a rule of thumb of practical statistical analysis, the minimum sample size is typically between 20 and 30 [72].

Then, TEE-based operations (i.e., `Data Ops`:data management, `query Ops`:query, and `DP Ops`:differential privacy operations) execute on `Combined_Set`, with all configurations stored securely in `Configs` (step 6). Finally, the results are returned from `TEE` to `Mobile Apps`. More importantly, based on the persisted data's content and configuration, `Accessibility Components` calculate the noise scale for each supported query type and detects whether the newly contributed data is trustworthy. These two features run on dedicated worker threads, synchronized by means of Android's `Handler` and `Message`. In addition, whenever an app issues a query request, `Go-Between Service` can be configured to notify the permission granting app (i.e., `User Consent`), so the end-user can approve the request to proceed.

Go-Between is integrated into the Android Platform as part of its standard SDK: `Go-Between API` and `Accessibility Components` into the Android Framework Layer, `Go-Between Service` into both the Framework and Native Library Layers, `Encoding Protocol` into the Native Library Layer, and `Native Interface` into the Hardware Abstraction Layer [41]. Since inadvertent misconfigurations or system attacks can cause data leakage, `Combined_Set` and `Configs` are placed in TEE to become hard-to-compromise, while `Data Ops`, `Query Ops`, and `DP Ops` run in TEE as trusted operations.

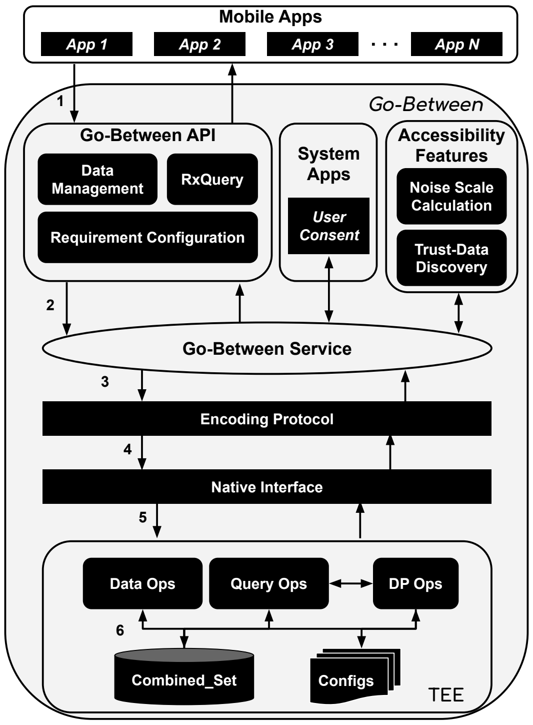


**Fig. 5.** System Architecture.

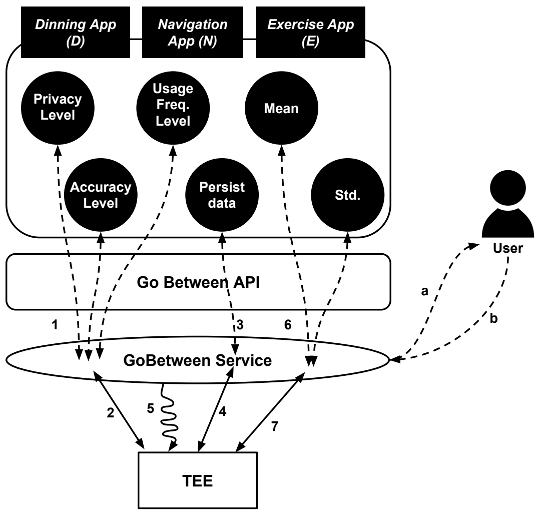


**Fig. 6.** Go-Between Programming Model.

## 5.2 Programming Model

In the application scenario in § 2.1, a blood pressure monitor app M, a smartwatch app W, and a personal health records app H cooperate to infer whether their user is suffering from hypertension. When in operation, each app is recording the user's blood pressure. These apps must be retrofitted to use Go-Between services, and their installation procedure must secure a permission to do so, with Go-Between keeping track of the permitted apps. Once the installed apps start persisting their data collections in Go-Between, the combined dataset becomes available for querying.

The code snippet below and Figure 6 show the Go-Between programming interface and its interactions with the rest of the framework, respectively. First, M, W, and H each obtains a service instance from the Android service manager (line 1). Then, method `setContext` sets `application context` (line 2), which determines a unique interaction id between an app and Go-Between. Next, the apps configure their privacy and utility requirements (steps 1,2; lines 3). Step 3 in Figure 6 identifies the data persistence phase; method `storeData` (line 4) takes the parameters containing a private data collection and its data type. These parameters are transferred to `Go-Between Service` (step 4). The data type parameter identifies the type of the passed dataset's items. Then, Go-Between combines the data collections with the same data type in a combined dataset in TEE for inferring user profiles.

```
1 GoBetweenManager<Double> gb = (GoBetweenManager) getSystemService(GOBETWEEN_SERVICE);
2 gb.setContext(application_context);
3 gb.setRequirements(query_type, privacy_level, accuracy_level, usage_freq);;
4 gb.storeData(data_type, data);
5 MathObservable.Mean().subscribe(callback);
```

For each predefined query, Go-Between service calculates the noise scale using the contributed data and configured privacy level, while detecting and removing spurious data from the combined dataset (step 5). These two features run on dedicated worker threads, synchronized by means of Android's `Handler` & `Message`. Go-Between adopts the function query interface of RxJava [86] to provide reactive queries (`Mean` in example, line 5). By executing the actual data calculations in TEE, the design is invulnerable to malicious tampering or interception. By adding noise to the returned query results, the design prevents apps from uncovering the raw collections of their collaborating apps. If a query cannot be executed with the required levels of privacy and utility, it throws a runtime exception that must be caught and handled.

Before using any Go-Between services, an app must secure an explicit permission from the mobile user. To that end, Go-Between first notifies the user about the apps involved, the data type they want to share, and which query is performed (step a). Then, the user communicates with Go-Between to grant the permission (step b). If the permission is declined, the involved parties can request it again at some later point. Once the permission is granted, the parties start collaborating via Go-Between.

Go-Between effectively reconciles the requirements of protecting data privacy and of inferring user profiles from the protected data. We describe how Go-Between addresses these two requirements in turn next.

### 5.3 Privacy Mechanism

Go-Between's adaptively parameterized privacy mechanism: 1) configures each app's privacy and utility requirements, 2) determines the required noise scale, 3) detects and removes spurious data, and 4) adds noise and verifies the results still meet the requirements.

**I. Privacy & Utility Configuration.** With Go-Between, developers of mobile apps can configure the privacy and utility levels for each query. However, unless those developers are data privacy experts, determining the exact required privacy levels is hard. To address this problem, the Go-Between API provides human-readable levels, as shown in Table 1, to express the requirements, which include the *privacy level* (i.e., privacy requirement) and the *accuracy level* & *usage frequency level* (i.e., utility requirement). Each of them is divided into five consecutive levels from lowest to highest. Queries with higher *privacy levels* need more noise added to the result, and vice versa. The lower the *accuracy level*, the more the noisy and the original results differ, and the less useful the noisy results are for inferring user profiles. With the restriction on the *usage frequency level*, $\varepsilon$-differential privacy can be ensured even if the apps continuously invoke a certain query or perform a fused sequence of queries.

**Table 1.** Privacy & Utility Requirements

| Privacy$^{\alpha}$ | | Accuracy$^{\beta}$ | | Usage Freq.$^{\gamma}$ | |
|---|---|---|---|---|---|
| Level | Pr. | Level | Err. | Level | Times |
| Lowest | 70% | Lowest | 50% | Lowest | 1 |
| Public | 50% | Estimate | 30% | Rare | 5 |
| Default | 20% | Default | 20% | Default | 10 |
| Critical | 5% | Exact | 10% | Frequent | 50 |
| Highest | 1% | Highest | 5% | Highest | 100 |

$^{\alpha}$ *Privacy level* is the probability of an adversary correctly discovering the combined dataset's raw data, used to calculate the noise scale.

$^{\beta}$ *Accuracy level* is the *% error* as per formula (8), a noisy result's utility for inferring user profiles.

$^{\gamma}$ *Usage frequency level*, a query's invocation #, used to calculate a query's noise scale, especially in a sequence of queries.

Consider applying the predefined query `Mean`, which obtains the *mean* value of the combined dataset, to our running example of sharing blood pressure readings. Suppose the personal health records app H prefers higher privacy and utility levels of `Mean`, so it may set the privacy level to `Critical`, the accuracy level to `Exact`, and the usage frequency level to `Frequent`; the smartwatch app W prioritizes privacy only, so it could configure the requirements as `Highest`, `Estimate`, and `Rare`, respectively; the blood pressure monitor app M, with regular privacy and utility requirements, may set all parameters to `Default`. Based on a given configuration, Go-Between automatically calculates the required noise scale, and determines how to execute each query.

**II. Noise Scale Calculation.** Once apps specify their privacy/utility requirements and persist their datasets into TEE, Go-Between calculates the noise scale for each query in two steps: 1) determine the *global privacy level* (i.e., $\rho_{global}$) for the combined dataset contributed by the collaborating apps, 2) use $\rho_{global}$ to determine the noise scale (i.e., $b$) required to fulfill the privacy/utility requirements of each app. In addition, Go-Between incentivizes the collaborating apps to keep contributing data (as discussed in § 4.2).

To illustrate how Go-Between determines the global privacy level, consider the running example of apps H, W, and M setting their respective privacy levels for the `Mean` query to `Critical` (i.e., $\rho_H$), `Highest` (i.e., $\rho_W$), and `Default` (i.e., $\rho_M$), respectively. Go-Between first looks up the actual probability values for these human-readable levels as per Table 1:$\rho_H$=5%, $\rho_W$ =1%, $\rho_M$=20%. Then, by weighting the average value of these probabilities by the data collection size of each app, Go-Between determines the global privacy level: $\rho_{global}$=$c_H\rho_H$+$c_W\rho_W$+$c_M\rho_M$, where $c$ is the data *contribution rate* of each app (as per *formula-10* in § 4.1). Because Go-Between updates $\rho_{global}$ whenever new apps join an existing data sharing collaboration, $\rho_{global}$ always reflects the actual privacy requirement of the collaborating apps.

Having determined the global privacy level ($\rho_{global}$), Go-Between plugs the resulting value into the *formula-7* (§ 4.1) that calculates the noise scale of each predefined query: $b = \Delta v/ln\frac{(n-1)\rho}{1-\rho}$, with $\rho$ becoming $\rho_{global}$, $n$ becoming the *size of the combined dataset*, $\Delta v$ becoming the maximal difference between the query results of each possible combination of the database $D'$ ($n-1$ records sampled from previous combined dataset $D$). To determine $\Delta v$, $n-1$ records are sampled from the combined dataset by performing each query on $n$ different data combinations (i.e., $\binom{n}{n-1}$). For example, for a combined dataset of 1000 items, select 999 (i.e., 1000 - 1) records, and perform a given query on them obtaining a result. Then, repeat this process to obtain the query results of 1000 different data combinations. Next, use the max and min results to calculate the $\Delta v$ for this query. Finally, calculate this query's noise scale using the *formula-7* above. Go-Between obtains the noise scale for each predefined query, persisting the results into TEE.

To determine the final noise scale for each app, Go-Between executes the *bonus mechanism* and applies the *noise increase scheme*. In our running example, suppose the *contribution rates* ($c$) of apps H, W, and M are $c_H$, $c_W$, and $c_M$, respectively, while their *usage frequency levels* ($u$) for the `Mean` query are `Frequent` (i.e., $u_H$), `Rare` (i.e., $u_W$), and `Default` (i.e., $u_M$), respectively. These levels correspond to the max # of invocations: $u_H = 50, u_W = 5, u_M = 10$ (as per Table 1). Then the actual noise scale of `Mean` for each app is calculated using: $b_i = u_i \cdot (b_{query} - c_i \cdot BONUS)$, where $b_{query} = \Delta v/ln\frac{(n-1)\rho_{global}}{1-\rho_{global}}$, with $\{u_i|u_H, u_W, u_M\}$ and $\{c_i|c_H, c_W, c_M\}$ (as per *formula-11* in § 4.2). For example, each time app H performs `Mean` on the combined dataset, Go-Between ensures $\varepsilon$-differential privacy by adding the *Laplace noise* to the query result with the noise scale: $b_H = u_H \cdot (b_{mean} - c_H \cdot BONUS)$.

**III. Detecting Spurious Data Contributions.**

*a) Calculate Trust Sensitivity (TS):* Having the newly calculated noise scale $b_{new}$ and the original scale $b_{original}$ for each query (e.g., mean, standard deviation, and frequency), Go-Between calculates each query's $TS$ independently to produce $TS_{query}$ (as per *formula-12* § 4.3) (e.g., $TS_{mean}$, $TS_{std}$, and $TS_{freq}$). Then, Go-Between obtains $TS$ by selecting the minimum value of $TS_{query}$, i.e., $TS$ = min{$TS_{mean}$, $TS_{std}$, $TS_{freq}$}. Appendix-B provides the specific steps of computing $TS$.

**Table 2.** Trust Levels

| Trust Level | App Reputation | $\alpha$ value |
|---|---|---|
| 5 | built-in/system or $>4$ stars | 0 |
| 4 | $3\sim4$ stars | 0.3 |
| 3 | $2\sim3$ stars | 0.5 |
| 2 | $1\sim2$ stars | 0.7 |
| 1 | $\leq1$ star | 1 |

*b) Determine $\alpha$:* The contributing app's $\alpha$ is extracted by consulting the app's reputation score (i.e., collected from the Google Play), with Table 2 showing all trust levels. The built-in or system apps are assigned the highest trust level. The trust level of other apps corresponds to their reputation scores. The higher the trust level, the lower the $\alpha$ value, allowing to accept more "spurious" data.

*c) Determine Trust-Data:* If $TS \geq \alpha$, $\Delta D$ is trustworthy, so Go-Between keeps it in the combined dataset. If $TS < \alpha$, $\Delta D$ is declined as a spurious contribution.

**IV. Verifying Utility of Noisy Results.** With the generated *Laplace noise* added to a query result, Go-Between verifies whether the query satisfies the required accuracy level (i.e., $a$). Recall that app H set its accuracy level ($a_H$) to `Exact`. To verify the utility of the noisy result, Go-Between calculates `accuracy level` ($a_{lvl}$) as per formula(8) (discussed in § 4.1): $a_{lvl} = 100 \cdot \left|\frac{Result_{noise} - Result_{actual}}{Result_{actual}}\right|$. If $a_{lvl} \leq a_H$, Go-Between returns the noisy result or failure otherwise.

## 5.4 Inferring User Profiles

Go-Between offers trusted, TEE-based operations, exposed via the reactive and functional programming interface of RxJava, a popular reactive programming framework with a large set of predefined data operations that can be flexibly fused.

***1) TEE-based Reactive Query.*** The reactive programming in RxJava framework creates asynchronous data streams, which can be observed and operated on. Go-Between moves the creation of data stream into TEE. To that end, Go-Between provides a set of statistical queries (e.g., count, mean, std.) whose execution creates a data stream. These queries operate in TEE and obtain the $\varepsilon$-differentially private results as data streams, operated on by means of the customized RxJava framework to combine and filter them.

Consider how App M can infer whether the user suffers from hypertension: invoke the predefined queries of `Mean` to obtain the distribution of the combined dataset of blood pressure readings. M creates a data stream by invoking `Mean` API to obtain the *mean* value, with the request being passed to `Go-Between service`. The permitted request is encoded and passed to TEE, which executes the actual `Mean` operation. The operation's result is padded with noise, verified, and returned as a data stream.

***2) Flexible Inference with Function Fusion.***

A powerful functional programming mechanism, function fusion, creates new functions by efficiently combining and customizing existing ones [24]. Go-Between adopts this mechanism, allowing to fuse both predefined and user-defined functions via the client interface of the RxJava framework.

Consider our running example , app M needs to calculate the probability that a given blood pressure reading is in the combined dataset. Although Go-Between provides no predefined operation for this calculation, M can fuse the predefined `CountAll` and `CountOne`, as follows:

```
1  Observable<Integer> obsAll, obsOne;
2  obsAll = MathObservable.CountAll();
3  obsOne = MathObservable.CountOne(bp);
4  Observable.zip(obsAll, obsOne,
5    new Func2<Integer, Integer, Double>(){
6       @Override
7       public Double call(Integer all,Integer one)
8       {return one/Double.valueOf(all); } }
```

App M first calls `CountAll` to create a TEE-based data stream of the combined dataset size, referred to by an `Observable` object (line 2). Similarly, calling `CountOne` produces another `Observable` that refers to a data stream that contains the number of items in the dataset as per the `bp` parameter (i.e., the given blood pressure reading) (line 3). Go-Between fuses these two operations together with standard API `zip` (line 4), which passes each operation's data stream to the downstream function (i.e., `call`) as the parameters (line 7). Finally, `call` calculates the probability of the given blood pressure reading being present (line 8).

To fuse a predefined operation with a custom function, Go-Between uses function `subscribe`, as follows:

```
1  MathObservable.Std()
2    .subscribe(new Action1<Double>(){
3         @Override
4         public void call(Double d){...} };
```

Having created a TEE-based data stream of standard deviation (`Std`), the developer subscribes a custom function for this data stream with `subscribe` (line 2). Then, the data stream is passed to downstream custom function `call` (line 4).

## 6 Evaluation

The following questions drive our evaluation.

– **Q1. Feasibility**: Does Go-Between offer acceptable performance levels?
– **Q2. Utility**: Do Go-Between's data sharing collaborations satisfy dissimilar app requirements?
– **Q3. Safety**: How effectively does Go-Between eliminate the threats of apps uncovering their competitors' raw data?
– **Q4. Programmability**: How do the software metrics of Go-Between compare to those of similar Android system services?

### 6.1 Experimental Setup

***1) Experimental Environment Choice.*** We implement and evaluate Go-Between using the official Android source code release, Android Open Source Project (AOSP), which provides an official virtualized execution environment[9] for testing and debugging Android apps. Because its standard distribution excludes a TEE component, we integrated Open-TEE[10] with AOSP by adding the Open-TEE source code to the main codebase of AOSP and building them together into a single executable image. To maximize the number of Android apps compatible with Go-Between, while having access to as many of advanced Android features as possible, we use the Android Lollipop 5.1 release, currently run by 14.4% Android devices (the third highest percentage among the 13 most popular Android platform versions [38]) and has cumulatively covered 80.2% devices (cumulative distribution [36]).
***2) Software & Hardware.*** The main system components of our experiments are: platform version is Lollipop; host OS is Linux; CPU (MHz) is 3599.943; cache size (KB) is 2048; and TEE solution is Open-TEE.
***3) Benchmarks.*** To establish a performance baseline, we create a set of benchmarks that isolate the Go-Between's operations that store data collections and perform the pre-defined queries. Real-world Android apps can directly invoke these operations (§ 5.1).

In addition, the software-based virtualization of TEE is bound to exhibit performance levels inferior to those offered by hardware-based implementations. Hence, our performance results not in any way unfairly benefit our approach. Since w.r.t. performance, our evaluation goals are only to demonstrate that it is feasible to offer a Go-Between-like service locally on the device, in the presence of an actual hardware-based TEE, the overhead of using Go-Between can only decrease.

### 6.2 Evaluation Design

***Q1.Feasibility:*** Despite the computationally intensive nature of data processing, Go-Between must operate unintrusively, with performance overheads comparable to those of similar Android system services. In light of these evaluation objectives, we design a set of representative micro-benchmarks and application scenarios, and measure the performance of the Go-Between-related functionality. Then we compare these results with Android App's standard response time limitation (i.e., Application Not Responding (ANR) error). Specifically, we measure the total execution time ($T_{total}$) it takes to execute a Go-Between operation with realistic data by a single app to understand service's performance impact. Specifically, we measure the total execution time taken by the predefined `storeData`, `Mean`, `Std`, `CountOne`, and `CountFreq` operations. Go-Between calculates noise scales and detects untrustworthy data concurrently on separate worker threads, whose performance we chose not to evaluate as they leave the main execution unperturbed.
***Q2.Utility:*** We follow our *running example-I (§ 2.1)*: with apps N (App1), E (App2), and R (App3) collecting geolocation data and collaborating via Go-Between to discover their user's favorite locations. The user in this experiment is the Yeti[11], who according to Nepali folklore haunts the Himalayas [57]. As it turns out, the Yeti is a sophisticated and demanding mobile user. Due to the need

[9] a common practice for studying various performance trade-offs on the Android platform [13, 25, 98].
[10] a portable framework for code to run on any standard TEE implementation.
[11] The Yeti is a metaphor that describes any real-world user with a similar behavioral profile in this application scenario.

to keep his existence inconspicuous, the Yeti refuses to upload any of the sensor output of his apps to the cloud; besides, the network infrastructure of Himalaya renders any cloud services inaccessible. Nevertheless, the Yeti demands highly personalized mobile services that customize the actively used apps to his usage profile.

App1, App2, and App3 deposit with Go-Between 100, 50, and 20 Himalayan geolocations, respectively. The experiment queries the combined dataset to determine the Yeti's favorite locations (i.e., `CountFreq`). The input is a square area, while the output is the number of times the Yeti visited the area. Each of the apps queries three areas, with different privacy and utility requirements. We seek answers to these questions: 1) how beneficial is Go-Between in discovering the Yeti's favorite locations as compared to using only the data of individual apps? and 2) how do the privacy and utility requirements affect the Go-Between's results?

***Q3.Safety:*** We follow our *running example-II (§ 2.1)*: a healthcare app H collects snapshots of systolic blood pressure (SYS). H applies Go-Between to persist its data collection of 100 records into TEE for collaborating with other apps, and sets its privacy and utility requirements as `Highest` (i.e., privacy level $\rho_H$), `Default` (i.e., accuracy level $a_H$), `Default` (i.e., usage frequency level $u_H$). Then, we simulate two attack scenarios:

*(1) Revealing raw data:* Because the reasonable range for SYS is between 90 and 180 mm Hg, H's competitor app C performs `CountOne`[12] on all possible values to discover the combined dataset's raw data. Further, to maximize the opportunity to uncover the raw data of H, the app C sets its requirement as `Lowest` (i.e., privacy level $\rho_H$), `Highest` (i.e., accuracy level $a_H$), `Default` (i.e., usage frequency level $u_H$). Hence, it can contribute a large number of records to heighten the weights, thus decreasing the privacy level of the entire dataset. Then, app C performs `CountOne` on the combined dataset to obtain the frequency of each possible SYS occurrence. Finally, it compares these query results with its own dataset to infer app H's raw data. We evaluate with app C's data sizes of 20, 100, 1000, 5000 to a) verify whether our approach can preserve the data privacy for each collaborating app; b) to determine the resiliency of our privacy protection as a relation to the size of the attacker's contributed dataset. To demonstrate how other data sharing approaches behave under this attack, we reproduce a classic *honest-but-curious* attack as the control group (released online: drive.google.com/open?id=1GHhF65WbTLS7ybnKkrWtcJdZTx78CHCR).

*(2) Contributing spurious data:* Suppose H persists its data collection of 100 SYS values ($\approx$120 mm Hg) recorded from a healthy person. Then, app C generates and persists random numbers to the combined dataset. Without loss of generality, we assume that both H and C set their privacy and utility requirements as `Default`, and H's trust level is 3 (Tab 2, i.e., set the threshold $\alpha$ to 0.5). As discussed in § 4.3, if $TS < \alpha$ (i.e., 0.5), Go-Between recognizes the newly contributed data as spurious, removing it from the combined dataset. C can generate data collection within the range of (0, 90), [90, 180], and [110, 130], and the generated data collections can be 20, 100, 1000 records. To measure the worst possible case, we assume C can generate and persist random data collections many times to make its $TS$ value greater than $\alpha$. Specifically, we evaluate C's attempts numbered at 100, 10000, and 1000000 to identify the maximum $TS$ value, thus verifying how effectively Go-Between detects the contributed spurious data.

***Q4.Programmability:*** We count the uncommented lines of code (ULoc) and the cyclomatic complexity in the most common usage scenario: an app persists its data collection via `storeData`, and then performs statistical queries (via single or fused operations). Besides, we select two standard Android APIs (Backup service[13], and Location service[14]) as the control group, and count their ULoc as based on the Android official usage guides [37, 39].

## 6.3 Results

***Q1.Feasibility:*** As discussed in § 6.2, we benchmark the respective latencies of persisting data (i.e., `storeData`), and querying the combined datasets (i.e., `Mean`, `Std`, `CountOne`, and `CountFreq`) with dissimilar data sizes (i.e., 20, 100, 1000, 5000). As shown in Fig.7, neither of the predefined queries exceeds

[12] `CountOne(m)` queries "how many times does value 'm' appear in the combined dataset?"

[13] Similar to Go-Between's data persistence, it enables local mobile apps to back up their data at a remote server.

[14] Similar to Go-Between's statistical queries, it provides a query interface for obtaining geolocations from sensors.

10 ms in latency, due to their low asymptotic complexity O(n). For `storeData`, as the data size grows, so does the execution time (12 to 48ms), as the increases in data size are directly proportional to the work performed by the data encoding/decoding and storing processes. To sum up, the Go-Between API operations execute within the maximal threshold imposed by the Android platform (response time<5s [44]) and expected by end-users (response<1s [71]); even the longest observed response time taken by `store-Data` (≈48 ms) is still within these boundaries.

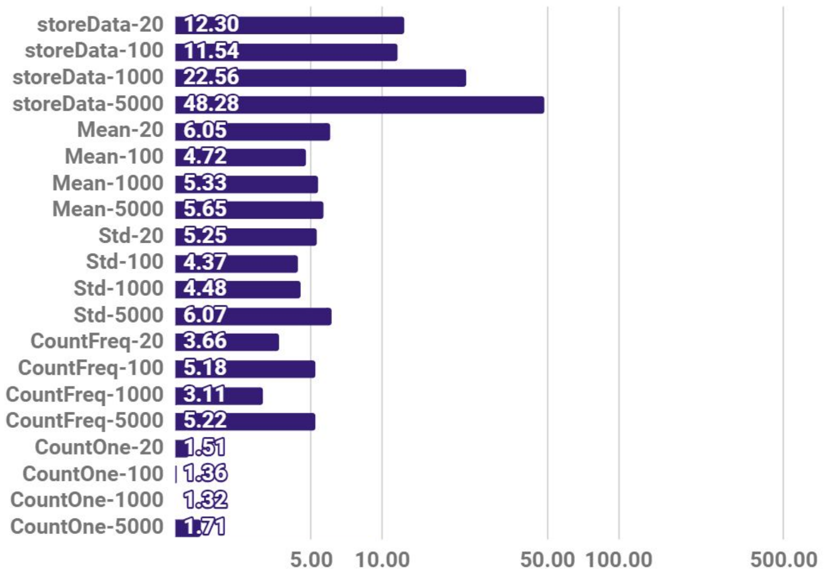
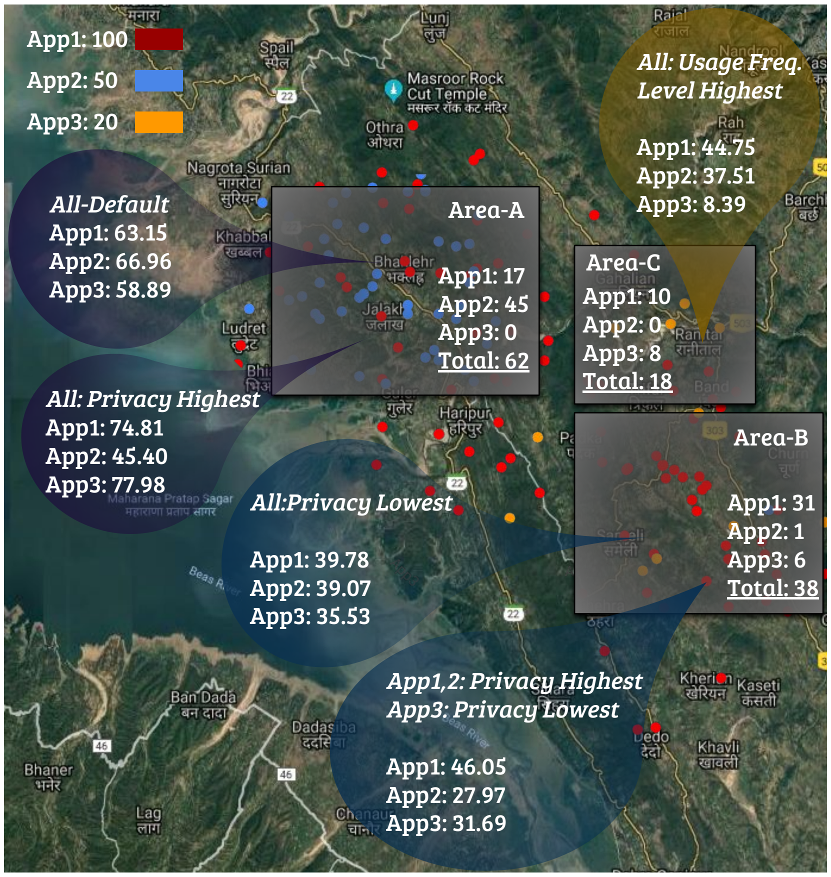


**Fig. 7.** Performance (log scale with millisecond).

***Q2. Utility:*** As shown in Fig. 8 (in square), without Go-Between, the query of "how many times the Yeti visited a given area", performed by App1, App2, and App3 returns 17, 45, and 0, respectively, for *Area-A*; 31, 1, and 6 for *Area-B*; and 10, 0, and 8 for *Area-C*. Hence, App1 would think *Area-B* as Yeti's favorite, App2 *Area-A*, and App3 *Area-C*. However, in fact, the Yeti's favorite area is A (62 visits overall), so without Go-Between the Yeti would be left very unhappy with the customizations of App1 and App3.

We study the utility of Go-Between under different requirements. Fig. 8 shows (in bubble shape), with the "Default" settings for all requirements, the subject apps obtain 63.15, 66.96, and 58.89 visits of *Area-A* (i.e., the bubble in the upper left corner). The $\varepsilon$-differential privacy of these results is based on the configured "Default" privacy level for each app. The noise added to these results is based on each app's noise scale ($n$) of this query. Because App1 contributes more data than App2, which contributes more data than App3, $n_1$¡$n_2$¡$n_3$. By adding the least noise to App1's result, Go-Between makes it most accurate (i.e., 63.15 is the closest to the actual result of 62). Although one cannot guarantee that App1's results would always be the closest to the actual value, the smallest noise scale maximizes such chances, in conformance with *Laplace distribution*.

Setting the privacy level to "Highest" reduces accuracy the most. As shown in the bubble in the middle left of Fig. 8, with the highest level, the query results for *Area-A* become: 74.81 (App1), 45.50 (App2), and 77.98 (App3), deviating greatly from the actual result of 62. Hence, one can increase privacy at the cost of accuracy, and each app can specify the accuracy level as required for a given scenario. To maximize accuracy, apps set their privacy level to "Lowest" and query for *Area-B* (actual value: 38). Indeed, the results' accuracy increases: 39.78 (App1), 39.07 (App2), 35.53 (App3) (i.e., the bubble in the middle). Notice that the result for App2 (i.e., 39.07) is closer to the actual value than that of App1 (i.e., 39.78). For very small noise scales, the amount of added noise is small as well, making the results of App1 and App2 close to each other. Despite the differences in the added noise, the results still conform to the *Laplace distribution*.



(a) the **square** (i.e., Area-A, B, and C) contains the number of the Yeti's visits to the area, as reflected in each app's individual data collection (i.e., without Go-Between and the combined dataset). E.g., square"Area-A" shows that App1 records 17 visits of the Yeti to this area, App2 45 visits, and App3 0 visits. So, the actual number of the Yeti's visits to the Area-A is 62 (i.e., 17 + 45 + 0).
(b) the **bubble shape** contains the query results for each app using Go-Between with the combined dataset. E.g., the bubble in the upper left corner shows that with the "Default" settings for all requirements ("All-Default"), the subject apps obtain 63.15 ("App1"), 66.96 ("App2"), and 58.89 ("App3") visits to "Area-A".

**Fig. 8.** Utility of Go-Between.

As discussed in § 5.3, the app contributing more data increases its power to impact the combined dataset's privacy level. To evaluate this feature, we let App1 and App2 (respectively contributing 100 and 50 geolocations) set their privacy levels to "Highest", while keeping it "Lowest" for App3 (contributing only 20 geolocations). The results (i.e., the bubble in the bottom middle of Fig. 8—App1: 46.05; App2: 27.97; App3: 31.69) show that even with "Lowest" privacy level for App3, the global privacy level increases, as the other apps contribute more data.

As discussed in § 5.3, the usage frequency level also trades privacy for accuracy: the more a query executes, the easier it is for an adversary to discover the raw data of others. Go-Between mitigates this risk by increasing the noise scale in accordance with the observed usage frequency, which corresponds to the max number a query has been invoked. To evaluate this feature, we let all apps set their usage frequency to "Highest", meaning that the query can be repeated up to a 100 times. The results (i.e., the bubble in the upper right corner) become 44.75 (App1), 37.51 (App2), and 8.39 (App3), while the actual value is 18. That is, by setting the usage frequency level to "Highest", apps can continuously invoke frequent queries whose results would not be as accurate.

***Q3.Safety:*** *(1) Defending against the attack of revealing raw data:* App H deposits with Go-Between a dataset containing 20 duplicate systolic blood pressure (SYS) snapshots of 150 mm Hg. App C perpetrates an attack to discover H's raw data, by setting its privacy level to "Lowest" and performing the query `CountOne(150)` (i.e., "how many times does value 150 appear in the combined dataset?"). Table 3 shows that, with the size of its contributed dataset growing (the "Size" column), C's contribution rate increases (the "Contribution Rate" column), noise scale decreases (the "Noise Scale" column), and query results (the "Noisy Value" column) approach the actual value (i.e., 20).

With C's "Lowest" privacy level, as the size of C's contributed dataset increases, the global privacy level decreases, producing a fairly small noise scale to privatize the query results. Therefore, with little added noise, the C's query results are close to the actual values. Note that the added noise conforms to the *Laplace distribution*, whose peak's sharpness is controlled by the noise scale. The smaller the noise scale, the sharper the *Laplace distribution*'s peak, so the added noise fluctuates less, increasing the confidence in the accuracy of the query results. That is, with the actual value of 20, querying a dataset of 1000 items produces 20.39, while querying a dataset of 5000 items produces 19.49. Although 20.39 is closer to 20 than 19.49 is, the second query's noise scale is lower, increasing the attacker's confidence in its ability to discover the actual value. However, Go-Between still prevents the attacker from determining what the exact raw data is.

**Table 3.** Safety of Go-Between *

| Size | Contribution Rate | Noise Scale | Actual Value | Noisy Value |
|---|---|---|---|---|
| 20 | 16.7% | 3.23 | 20 | 11.26 |
| 100 | 50.0% | 2.02 | 20 | 22.14 |
| 1000 | 90.9% | 1.31 | 20 | 20.39 |
| 5000 | 98.0% | 1.07 | 20 | 19.49 |

* As a control group, we also reproduced a classic *honest-but-curious* attack: *Without* Go-Between, the attacker always uncovers the actual readings (this experiment is released online: drive.google.com/open?id=1GHhF65WbTLS7ybnKkrWtcJdZTx78CHCR).

To summarize, a) with high contribution rates, attackers may roughly guess what the raw data is, while being unable to discover the exact values; b) to reduce the risk of such attacks discovering the raw data, collaborating apps can increase the global privacy level by contributing more data.

*(2) A Control Group: an Honest-But-Curious Attack:* We develop the application scenario in § 2.1-II, in which the apps M, W, and H collect the user's blood pressure readings. Firstly, we implemented an Android service whose access API provides two methods—`store` and `frequency`. The intended usage scenario is for the collaborating apps to first store their sensor data with the service, which is then queried for the occurrence frequency of given elements in the resulting collection. We then reproduce the honest-but-curious attack: M, W, and H store their respective collected blood pressure readings with the service. Afterwards, H, which acts as an honest-but-curious party, performs frequency queries with the goal of uncovering the combined collection of blood pressure readings. Background knowledge suggests that the reasonable range for systolic pressure are integer values between 90 and 180 mm Hg. By exploiting this fact, H performs 91 frequency queries for each possible value in this range. The return result of each query is then compared with the H's own dataset. By subtracting the number of occurrences of the queried value from those in its own set, H deduces the actual values stored in the combined dataset. In this experiment, we randomly generate the blood pressure collections for M, W, and H. In the 1000 repeats of the experiment, H was always able to uncover the combined blood pressure readings.

*(3) Defending against the attack of contributing spurious data:* Tab. 4 shows the evaluation results of detecting untrustworthy data. *First, the randomly generated data collection's $TS$ value is positively correlated with the degree of similarity between the generated data and the original dataset.* Since the original dataset contains 100 records of SYS values that are around 120mm Hg, the generated data collection within the range of [110, 130] obtains large $TS$ values (>0.7), and that within the range of (0, 90) obtains small $TS$ values (<0.3). *Second, the more attempts of generating data are, the more opportunity is for the data's distribution to be close to the original one, and the larger the $TS$ values are* (see the difference between the "1M" and "100" columns). *Third, the more records are in the generated random data collection, the more stable the $TS$ value is.* The 100-

**Table 4.** Trust Data detection of Go-Between

| Newly Contributed | Size | TS value | | |
|---|---|---|---|---|
| | | *attempt times:* 100 | 10K | 1M |
| SYS value: (0, 90) | 20 | 0.13 | 0.19 | 0.26 |
| | 100 | 0.08 | 0.10 | 0.11 |
| | 1000 | 0.08 | 0.11 | 0.12 |
| SYS value: [90, 180] | 20 | 0.40 | 0.53 | 0.62 |
| | 100 | 0.27 | 0.32 | 0.33 |
| | 1000 | 0.30 | 0.32 | 0.33 |
| SYS value: [110, 130] | 20 | 0.75 | 0.76 | 0.80 |
| | 100 | 0.76 | 0.76 | 0.76 |
| | 1000 | 0.75 | 0.76 | 0.76 |

record data collection's $TS$ values are always stable, similar to 1000-record's ones (a large collection is sampled to 100 records to calculate its $TS$ value). However, with 20 records, because it is more likely to randomly generate numbers having a similar distribution to such a small amount of data, the data collection's $TS$ values fluctuate but still has an upper bound. That is, in the range of [90, 180], the generated collection's $TS$ is fluctuating between 0.40 and 0.62, but is always lower than the $TS$ ([0.75, 0.8]), whose collection falls in [110, 130] (closer to the original data).

To sum up, in our case (i.e., $\alpha = 0.5$), Go-Between successfully detects the untrustworthy data (i.e., the entire row of SYS value: (0, 90), and the row of SYS value: [90, 180] with 100 and 1000 records) and removes it from the combined dataset. In contrast, once a given data collection's distribution is similar enough to the original dataset, Go-Between considers contributions as trustworthy data (i.e., distribution impacts are acceptable), keeping them in the combined dataset.

***Q4.Programmability:*** Table 5 presents the software engineering metrics of using Go-Between and two representative Android system services. To assess the programming effort required to apply Go-Between, we measure its metrics in a normal use case and compare the results with similar cases in the control group (receive location updates for the location service [42], and back up SharedPreferences for the backup service [37]). The Go-Between metrics are within the expected range, 11 ULoc (calculate mean in § 5.2) and 21 ULoc (calculate probability in § 5.4), with the cyclomatic complexity of 2. In contrast, the representative services take between 10 and 13 ULoc with comparable cyclomatic complexity.

**Table 5.** Programmer effort comparison

| Service | ULOC | Cyclomatic Complexity |
|---|---|---|
| Go-Between | $\approx$ 11(mean); $\approx$ 21(probability) | 2 |
| Location | $\approx$ 10 | 2 |
| Backup | $\approx$ 13 | 1 |

## 7 Discussion

In this section, we start by discussing the main findings of our approach, comparing it with standard Android services and other studies. We then illustrate the implications and explanations of the findings as well as the strengths and limitations of our approach. Finally, we conclude these discussions by recommending to practitioners how to effectively preserve data privacy within the device and drawing future directions.

**(1) Main findings**

*Finding-1 The performance and implementation costs of privacy-preserving operations for on-device data sharing (i.e., data on-loading) are not prohibitively expensive.* As per § 6.3-Q1 and Q4, Go-Between's differential privacy-based queries and interfaces impose modest performance and implementation overheads on the underlying Android system: all queries execute in < 10 ms, with the extra functionality taking fewer than 21 ULoc with the cyclomatic complexity of 2 to implement.

*Finding-2 It is possible to satisfy data sharing participants' dissimilar privacy and utility requirements to a large extent.* As per § 6.3-Q2, by tuning the privacy, accuracy, and usage frequency levels, one can trade their data privacy for utility and vice versa.

*Finding-3 Defending against the attacks of revealing raw data and contributing spurious data is possible for data sharing.* As per § 6.3-Q3, with a differential privacy-based mechanism, Go-Between effectively reduces the possibility of guessing the raw data from the shared dataset. Moreover, guided by Trust-Data Theory, Go-Between can successfully discover untrustworthy data.

**(2) Comparisons**

One could have highlighted our approach's traits by comparing our evaluation results with existing works. However, to the best of our knowledge, the relevant data sharing solutions either do not work on Android or support different data types and queries. Hence, it remains an open question of how to construct a suite of reasonable and consistent criteria and provide a valid comparison with these existing works, despite their dissimilar usage scenarios, execution environments, and application scopes.

To mitigate the above problem, we carried out the following comparisons:

– we compared the execution time between our approach's API operations and the maximal threshold imposed by the Android platform (response time<5s) and expected by end-users (response<1s). The evaluation results show that Go-Between's operations of persisting data and querying the combined datasets execute within 10 ms, which is far less than the above thresholds;
– we reproduced a classic *honest-but-curious* attack of revealing raw data as the control group. The evaluation result shows that, without our approach's privacy-preserving mechanism, an attacker can always reveal the combined dataset by executing particular queries;
– we mimicked an attack of contributing spurious data. The evaluation result demonstrates that Go-Between can successfully detect untrustworthy data;
– we compared the cyclomatic complexity between our approach's API and two standard Android APIs (Backup, and Location service) that support usage scenarios similar to ours. The evaluation result shows that Go-Between's interfaces add 21 ULoc with the cyclomatic complexity of 2, similar to these two standard Android APIs above.

**(3) Implications & Explanations**

*(a) App providers' privacy vs. mobile users' privacy:* Since all deposited data belongs to the device user, improving the data privacy of app providers also improves user privacy. In contrast to those privacy preservation approaches that focus exclusively on user privacy without any regard for the business aspirations of app providers, we strive to take a more holistic approach. We acknowledge that app providers need to achieve their business objectives of personalizing mobile services and provide them with a convenient privacy-preserving framework to accomplish that. In essence, our goal is to take away a major motivation for app providers to illicitly bypass the privacy protection mechanisms in place. Our middleware enables app providers to accomplish their business objectives in a privacy-preserving fashion, thus implicitly improving end-user privacy.

*(b) Privacy & utility tradeoffs:* Our evaluation results in § 6.3 illustrate that, with dissimilar accuracy, privacy, and usage frequency levels (discussed in table 1), the same query will return different values from the combined dataset. The reason is that the value of these levels directly impacts how to calculate the noise scale, which controls how much noise will be inserted into the original data. Hence, by tuning these levels, app providers can balance their mobile service's utility and data privacy.

*(c) About contribution rates:* Recall that we introduce the contribution rates for incentivizing app providers' data contribution: the more data an app contributes, the more accurate its query results are (discussed in § 4.2). As shown in Table 3, the noisy value (i.e., 19.49 and 20.39) will be very close to the actual value (i.e., 20) when the contribution rate is over 90%. This phenomenon implies that an attacker could intentionally contribute a large amount of data, so they can roughly guess what the raw data is. To mitigate this risk, other apps can also contribute more data to the combined dataset, which can dilute their collaborators' contribution rate and act against the monopolizing of data contribution by a single party.

*(d) About threshold-$\alpha$:* Recall that we introduce $\alpha$ to detect the untrustworthy data: the smaller $\alpha$ is, the less stringent the trust condition is, the more the out-of-distribution data can be accepted (as discussed in § 4.3). As shown in Table 4, if $\alpha = 0.7$, then only the row of SYS value: [110, 130] is considered trustworthy data (i.e., its TS value $>0.7$), the other two rows will be treated as untrustworthy data and removed from the combined dataset. However, all the data can be accepted if $\alpha = 0$. Since the value of $\alpha$ depends on an app's trust level (discussed in Table 2), it effectively adjusts how stringent the spurious data detection is for apps with different trustworthiness.

**(4) Limitations**

*(a) Applicability:* Our approach is applicable to scenarios in which different mobile apps collect the same type of sensor data. Specifically, we designed Go-Between to allow collaborating apps to deposit their individually collected data. Because the collected datasets are of the same type, Go-Between can aggregate the deposited data into combined datasets, while differentially privatizing

the data-sharing process. Such scenarios do frequently occur in real-world mobile apps. For example, location-based apps, including Google Maps, Uber, and Yelp, collect geolocations during their execution. In addition, an empirical study reports that ten different mobile apps, in categories ranging between mapping and navigation to social media (e.g., Facebook) and shopping (e.g., Groupon), shared geolocations 5,398 times within 14 days [10]. Hence, our approach specifically targets application scenarios like that. In the cases in which the collected data happens to be of dissimilar types, our approach would be inapplicable as is. However, we plan to investigate if our approach can be extended to provide comparable privacy guarantees even to dissimilar data, exchanged by collaborating apps.

Moreover, our reference implementation, GO-BETWEEN, is an Android-based middleware. However, our approach's complemented differential privacy (§ 4) and system design (§ 5) can apply to other mobile platforms. Further, it can benefit other data sharing scenarios, such as sharing data at the edge, (e.g., smart home, autonomous driving), which often involves mutually distrustful parties that can apply our approach to personalize their services by sharing data without compromising their data privacy. Finally, to increase its applicability, our approach can incorporate functional and reactive query interfaces to express, fuse, and extend queries.

*(b) Trust-data theory limitations:* Despite its demonstrated benefits, our trust-data theory would be unable to detect and eliminate *all* untrustworthy data. In essence, the detection takes advantage of the difference in distribution between the newly contributed dataset and the original one, as well as the threshold for that difference. Put another way, a combination of an unreasonable threshold value and a close similarity between the distributions of the original and the newly contributed data collections would cause our approach to consider the contributed data trustworthy, keeping it in the combined dataset. So far, we assumed that the threshold value would be pre-set according to the apps' reputation score (discussed in § 5.3, Table 2). However, how to select a reasonable threshold value remains an open research question. As a future work direction, we plan to investigate the thresholds with different values and scenarios to be able to automatically adjust the threshold values.

*(c) Comparing with other studies:* As discussed above, we concluded that it would be counterproductive to try to compare our evaluation results with existing studies, due to the intrinsic differences in usage scenarios, execution environments, and application scopes. As an alternative, we compared our results with Android's built-in execution time threshold, standard Android APIs, and simulated attacks. Furthermore, the overriding goal of this article is to demonstrate how our approach can help personalize mobile services while protecting data privacy with affordable overheads (e.g., execution time is acceptable, data privacy is preserved, etc.). Hence, our evaluation was designed to support the overriding goal of this article, as explained above.

**(5) Recommendations and Future Directions**

*(a) Set prudent privacy requirements:* As mentioned above, privacy requirements (i.e., privacy, accuracy, and usage frequency levels) determine the utility of query results. In fact, these requirements directly depend on apps' specific usage scenarios. For example, a dining app that recommends a restaurant in an area may not need highly accurate user geolocation, while a health monitor app may need precise values of the user's body vitals. Hence, to set prudent privacy requirements, we recommend app providers carefully review all possible usage scenarios and assess how stringent privacy protection they would need.

*(b) Maintain a reasonable contribution rate:* As discussed above, the possibility of revealing raw data would increase if an attacker's app contributes most of the data in combined datasets. Such risk can be mitigated if there is no monopoly on data contributions: all collaborating apps contribute enough data as compared to others. In the future, we plan to provide interfaces that allow app providers to check their contribution rates and detect the potential presence of monopolization attempts at contributing data. Also, we recommend that the providers carefully decide whether or not to share their data in the presence of risks of monopolization attempts at data contribution.

*(c) Be careful in granting data sharing permissions:* Because of our on-device data sharing approach, end-users can always keep in control of their data. Also, GO-BETWEEN can be configured to inform end-users of the details of data sharing events and explicitly allow them to restrict apps to share data. We recommend that end-users periodically turn on this feature and check if everything proceeds as expected. Further, it remains the case that end-users should avoid installing apps with low trust levels from unauthorized app stores.

## 8 Related Work

***Data Privacy.*** Differential privacy [29] prevents attackers from discovering private information. Based on this concept, Airavat [85] provides differential privacy guarantees for cloud-based MapReduce computations. The PINQ [69] and GUPT [75] libraries add a unified amount of noise to raw data for privacy-preserving data analysis. Vu et al. [96] automatically detect the user's privacy requirements by determining the noise scale with a neural network model. Miller et al. [70] provide security protocols for clients to securely communicate with a non-trusted server. Gaboardi et al. [34] enable users, unaware of differential privacy mechanics, to generate privacy-preserving datasets that support statistical queries. GO-BETWEEN differs by inferring user profiles on-device, with its $\varepsilon$-differential privacy augmented with *privacy & utility tradeoffs*, *spurious data detection*, and *data contribution incentives.*

***Collaboration Among Distrustful Parties.*** By using encryption techniques, Private Set Intersection (PSI) enables two parties to find the intersection of their private datasets, without revealing any data outside the intersection. Kiss et al. [55] applies PSI techniques to find the set intersection of differently sized datasets in mobile applications, but not on-device as in our approach. PSI protocols have been also implemented for smartphones [14,21,51]. For example, CrowdShare [14] shares Internet connectivity and other resources across Android devices. Secure Function Evaluation (SFE) techniques allow mutually distrustful parties to evaluate the properties of private sets without revealing them. Previous work on SFE defines the adversarial models, decreases communication complexity, and improves efficiency/security definitions [20,65,76]. In contrast, GO-BETWEEN relies neither on encryption nor on the SFE techniques, allowing distrustful parties to effectively collaborate, while balancing their privacy/utility requirements by automatically determining the required noise scale.

***Applications of user profiles.*** user profiles are inferred to improve the quality of various services. Datta et al. [27] infer user profiles from an IoT architecture to personalize health care. Barnaghi et al. [16] explore how to use IoT architectures for gathering user profiles. Other uses of user profiles include detecting security threats [15,48,73], uncovering safety abnormalities [26,64], and enhancing education [32,68]. GO-BETWEEN also improves the quality of mobile services by enabling mutually distrustful parties to collaborate without having to exchange data under adaptively parameterized differential privacy. Our approach is likely applicable beyond mobile computing and can benefit the above domains.

***Spurious data detection.*** Detecting and filtering out spurious data from a data set remains an open problem. The current solutions can be roughly categorized into two types, which we discuss in turn next.

(1) statistics-based approaches: a series of classic statistical criteria, such as Benford's law, 3-$\sigma$ rule, Chauvenet Criterion, and Dixon Criterion, can help detect data points that are out of distribution from a given dataset. However, these approaches either require a specific data distribution or impose restrictive conditions (e.g., only one outlier exists).

(2) machine learning or deep learning-based approaches: Modern anomaly detection (i.e., outlier detection) makes use of machine learning or deep learning algorithms to identify outliers that deviate from the general data distribution. Specifically, OE discovers out-of-distribution data via training on the so-called "outlier exposure dataset" that contains out-of-distribution samples [50]. Similarly, ATOM trains on outlier data that is estimated as possible out-of-distribution data by an existing detector [22]. To improve the quality of training datasets, Li et al. re-sample given datasets and create an informative outlier dataset [60], while Du et al. create the training dataset by synthesizing outliers [28]. Furthermore, Ren et al. and Morningstar et al. enhance the deep learning-based detection performance for outlier data with likelihood ratio [84] and nonparametric density estimators [77], respectively. Besides, outlier detection (or out of distribution detection) has been proved that can be applied to gaze estimation [23], cyber-physical systems [19], wireless sensor networks [8], data streams [78], and Internet of Things [54].

However, all these approaches focus on detecting the outlier data but neither consider taking advantage of the outliers nor have applied the detection techniques to differential privacy. Our approach reconsiders the outliers to preserve data privacy while detecting and removing untrustworthy ones, which is a novel application of a known security exploit for benign purposes.

## 9 Conclusion

We have presented an on-device user profiles inference approach that personalizes mobile services while protecting data privacy via an $\varepsilon$-differentially private mechanism.

*Contribution:* Our approach augments differential privacy by (1) mitigating the threat of contributing spurious data, (2) satisfying the dissimilar privacy and utility requirements of the collaborating apps, (3) incentivizing the apps to keep contributing data, and (4) featuring extensible programming interfaces and secure operations. The evaluation demonstrates our approach's feasibility, utility, safety, and programmability: all queries' execution time is within 10 ms (feasibility); participants' dissimilar privacy/utility requirements are satisfied (utility); untrustworthy data are effectively detected and raw data are hard to reveal (safety); extra 21 ULoc with a cyclomatic complexity of 2 are added (programmability).

*Scientific value:* The scientific value of this article is as follows. First, we demonstrate that it is feasible and useful to personalize mobile services while protecting data privacy on a single device. Second, we show that it is practical to identify spurious data by leveraging the perturbation of Laplace distribution between the original dataset and the one with newly added data (i.e., Trust-Data theory).

*Limitations:* Existing studies' dissimilar usage scenarios, execution environments, and application scopes make it difficult to meaningfully compare our approach with prior approaches concerned with similar problems. To mitigate this limitation, we compared our evaluation results with Android's built-in execution time threshold, standard Android APIs, and simulated attacks.

*Applicability:* Although our reference implementation is Android-based, one can apply our approach's complemented differential privacy and system design to any other mobile platforms, as well as other data-sharing scenarios.

*Future directions:* As future work, we plan to revise and migrate our privacy-preserving mechanism to network communication protocols that can benefit edge computing. We also plan to explore whether it is possible to define reasonable and common privacy criteria in order to meaningfully compare approaches with different usage scenarios, execution environments, and application scopes.

## Acknowledgements

The authors thank the anonymous reviewers, whose insightful comments helped improve this article. This research is supported by Beijing Natural Science Foundation (Grant No. 4232019), NSF Grants #1717065 and #2232565, and Cisco Research Grant CG# 1367161.

## A List of Abbreviations

AOSP — Android Open Source Project
ANR — Application Not Responding
DP — Differential Privacy
PSI — Private Set Intersection
SYS — Systolic Blood Pressure
SFE — Secure Function Evaluation
TEE — Trusted Execution Environment
TS — Trust Sensitivity
ULoc — Uncommented Lines of Code

## B Calculating Trust Sensitivity – TS

### B.1 Probability Density Function of Laplace Distribution [88]

A random variable $x$ has a $Laplace(\mu, b)$ distribution if its probability density function is:

$$p(x) = \frac{1}{2b} exp(-\frac{|x-\mu|}{b}) = \begin{cases} \frac{1}{2b} exp(\frac{x-\mu}{b}) & x < \mu \quad (13) \\ \frac{1}{2b} exp(-\frac{x-\mu}{b}) & x \geq \mu \quad (14) \end{cases}$$

### B.2 Cumulative Distribution Function of Laplace Distribution [88]

By integrating the probability density function, the cumulative distribution function becomes:

$$P(x) = \int_{-\infty}^{x} p(u)du = \begin{cases} \frac{1}{2}exp(\frac{x-\mu}{b}) & x < \mu \quad (15) \\ 1 - \frac{1}{2}exp(-\frac{x-\mu}{b}) & x \geq \mu \quad (16) \end{cases}$$

### B.3 Calculating TS

According to the Laplace mechanism (*formula-5* in § 2.4), the privatized value is: $f(D_{original}) + Lap(\Delta f_{original}/\varepsilon_{original})$. After adding $\Delta D$, the privatized value becomes: $f(D_{new})+Lap(\Delta f_{new}/\varepsilon_{new})$. $\Delta D$ is the the newly contributed data, $D_{original}$, $\Delta f_{original}$, and $\varepsilon_{original}$ are corresponding values (discussed in § 2.4) before adding $\Delta D$, and $D_{new}$, $\Delta f_{new}$, and $\varepsilon_{new}$ are those after $\Delta D$.

To simplify the equation, let $f(D_{original})$ be $\mu_1$, $f(D_{new})$ be $\mu_2$, $\Delta f_{original}/\varepsilon_{original}$ be $b_1$, and $\Delta f_{new}/\varepsilon_{new}$ be $b_2$. As shown in Fig 9, when $\mu_1 \leq \mu_2$, we have five different cases (other cases would occur when $\mu_1 > \mu_2$).

$TS$ is calculated in three steps: (1) compare the parameters in terms of their respective sizes ($b_1$ vs. $b_2$ and $\mu_1$ vs. $\mu_2$); (2) compute the intersection points $x_1$ and $x_2$ (if necessary) by applying $b_1$,$\mu_1$ and $b_2$,$\mu_2$ to their probability density functions $p(x)$. Note that, different $b_1$,$\mu_1$ and $b_2$,$\mu_2$ yield different probability density functions $p(x)$, thus producing dissimilar intersection points; (3) calculate the integral of the probability density function $p(x)$, i.e., the overlapped area. The algorithm below shows the details of calculating $TS$ in the case of $\mu_1 = \mu_2$ and $b_1 < b_2$ (Figure 10).

**Algorithm 1** get overlap between $Lap(\mu_1, b_1)$ and $Lap(\mu_2, b_2)$

```
// step 1: compare $\mu_1$ $\mu_2$ and $b_1$ $b_2$
**if** $\mu_1 == \mu_2$ && $b_1 > b_2$ **then**
    // step 2: get intersections
    **get intersection point** $x_1$ via equation (13)
    **get intersection point** $x_2$ via equation (14)

    // step 3: find the integal
    **area** = $\int_{-\infty}^{x_1} Lap(\mu_1, b_1) + \int_{x_1}^{x_2} Lap(\mu_2, b_2) + \int_{x_2}^{\infty} Lap(\mu_1, b_1)$
    // $\int_{x_1}^{x_2} Lap(\mu_2, b_2)$ — hatched area (horizontal lines in Figure 10)
    // $\int_{-\infty}^{x_1} Lap(\mu_1, b_1)$ and $\int_{x_2}^{\infty} Lap(\mu_1, b_1)$
    // — hatched area (vertical lines in Figure 10)

    return *area*
**end if**
```

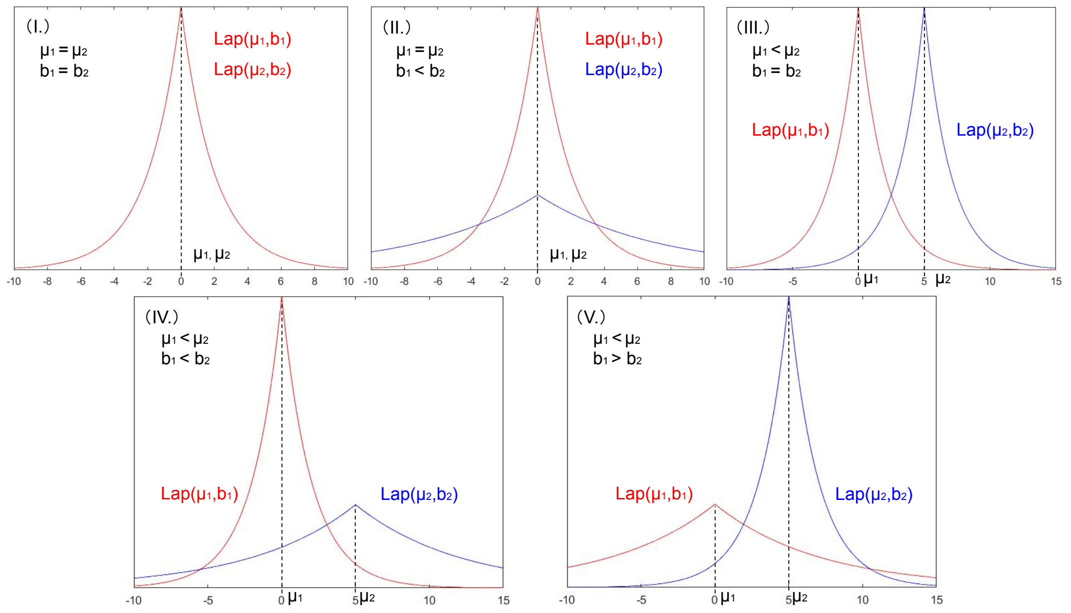


**Fig. 9.** Five Different Cases of $\mu_1 \leq \mu_2$.

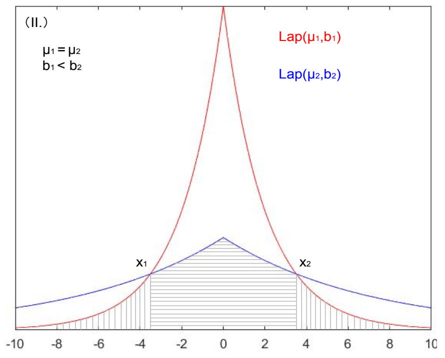


**Fig. 10.** *TS* in the case of $\mu_1 = \mu_2$ and $b_1 < b_2$.

## References

1. CVE-2016-6540., 2016. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6540`.
2. CVE-2017-8047., 2017. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-8047`.
3. CVE-2018-2409., 2018. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-2409`.
4. CVE-2018-3827., 2018. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-3827`.
5. CVE-2018-5560., 2018. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5560`.
6. CVE-2019-8063., 2019. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-8063`.
7. CVE-2019-9945., 2019. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9945`.
8. A. Abid, A. Kachouri, and A. Mahfoudhi. Outlier detection for wireless sensor networks using density-based clustering approach. *IET Wireless Sensor Systems*, 7(4):83–90, 2017.
9. A. Acquisti, C. Taylor, and L. Wagman. The economics of privacy. *Journal of economic Literature*, 54(2):442–92, 2016.
10. H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal. Your location has been shared 5,398 times! a field study on mobile app privacy nudging. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 787–796, 2015.
11. C. Anderson and M. P. Andersson. Long tail. 2004.
12. Apple Inc. App groups entitlement, 2017. `https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_security_application-groups`.
13. A. Armando, A. Merlo, M. Migliardi, and L. Verderame. Would you mind forking this process? A denial of service attack on Android (and some countermeasures). In *IFIP International Information Security Conference*, pages 13–24. Springer, 2012.

14. N. Asokan, A. Dmitrienko, M. Nagy, E. Reshetova, A.-R. Sadeghi, T. Schneider, and S. Stelle. Crowdshare: Secure mobile resource sharing. In *International Conference on Applied Cryptography and Network Security*, pages 432–440. Springer, 2013.
15. A. Atifi and E. Bou-Harb. On correlating network traffic for cyber threat intelligence: A Bloom filter approach. In *Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 384–389. IEEE, 2017.
16. P. Barnaghi, W. Wang, C. Henson, and K. Taylor. Semantics for the internet of things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 8(1):1–21, 2012.
17. B. Barth. Analysis of popular apps finds rampant sharing of personal data, 2020. `https://www.scmagazine.com/home/security-news/analysis-of-popular-apps-finds-rampant-sharing-of-personal-data/`.
18. R. Binns, U. Lyngs, M. Van Kleek, J. Zhao, T. Libert, and N. Shadbolt. Third party tracking in the mobile ecosystem. In *Proceedings of the 10th ACM Conference on Web Science*, pages 23–31, 2018.
19. F. Cai and X. Koutsoukos. Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 174–183. IEEE, 2020.
20. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.
21. H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. For your phone only: custom protocols for efficient secure function evaluation on mobile devices. *Security and Communication Networks*, 7(7):1165–1176, 2014.
22. J. Chen, Y. Li, X. Wu, Y. Liang, and S. Jha. Atom: Robustifying out-of-distribution detection using outlier mining. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 430–445. Springer, 2021.
23. Z. Chen, D. Deng, J. Pi, and B. E. Shi. Unsupervised outlier detection in appearance-based gaze estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.
24. W.-N. Chin. Safe fusion of functional expressions. In *ACM SIGPLAN Lisp Pointers*, number 1, pages 11–20. ACM, 1992.
25. S. R. Choudhary, A. Gorla, and A. Orso. Automated test input generation for Android: Are we there yet? In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 429–440. IEEE, 2015.
26. S. Das, B. L. Matthews, A. N. Srivastava, and N. C. Oza. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In *Proceedings of the 16th ACM SIGKDD*, pages 47–56. ACM, 2010.
27. S. K. Datta, C. Bonnet, A. Gyrard, R. P. F. Da Costa, and K. Boudaoud. Applying internet of things for personalized healthcare in smart homes. In *Wireless and Optical Communication Conference (WOCC), 2015 24th*, pages 164–169. IEEE, 2015.
28. X. Du, Z. Wang, M. Cai, and Y. Li. Vos: Learning what you don't know by virtual outlier synthesis. *arXiv preprint arXiv:2202.01197*, 2022.
29. C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.
30. C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.
31. C. Dwork and A. Smith. Differential privacy for statistics: What we know and what we want to learn. *Journal of Privacy and Confidentiality*, 1(2):135–154, 2009.
32. J. Fritz. Classroom walls that talk: Using online course activity data of successful students to raise self-awareness of underperforming peers. *The Internet and Higher Education*, 14(2):89–97, 2011.
33. Fueled's marketing team. How much money can you earn with an app in 2019, 2019. `https://fueled.com/blog/much-money-can-earn-app/`.
34. M. Gaboardi, J. Honaker, G. King, K. Nissim, J. Ullman, and S. Vadhan. PSI: a private data sharing interface. *arXiv:1609.04340*, 2016.
35. GlobalPlatform. GlobalPlatform, TEE system architecture, technical report. *www.globalplatform.org/specificationsdevice.asp*, 2011.
36. Google. Android studio – select the minimum api level, 2018. `https://developer.android.com/studio/projects/create-project`.
37. Google. Back up key-value pairs with Android backup service, 2018. `https://developer.android.com/guide/topics/data/keyvaluebackup.html`.
38. Google. Distribution dashboard, 2018. `https://developer.android.com/about/dashboards`.
39. Google. Getting the last known location, 2018. `https://developer.android.com/training/location/retrieve-current.html`.

40. Google. Google play, 2018. `play.google.com/store/apps?hl=en`.
41. Google. Hardware abstraction layer (HAL), 2018. `https://source.android.com/devices/architecture/hal`.
42. Google. Receive location updates, 2018. `https://developer.android.com/training/location/receive-location-updates#java`.
43. Google. Connect to the network, 2019. `https://developer.android.com/training/basics/network-ops/connecting`.
44. Google. Keeping your app responsive., 2022. `https://developer.android.com/training/articles/perf-anr`.
45. Google Play Store. Afwall+, 2019. `https://play.google.com/store/apps/details?id=dev.ukanth.ufirewall`.
46. Google Play Store. Internet blocker, 2019. `https://play.google.com/store/apps/details?id=com.superappsdev.internetblocker`.
47. Google Play Store. Noroot firewall, 2019. `https://play.google.com/store/apps/details?id=app.greyshirts.firewall&hl=en`.
48. C. Guitton. Foiling cyber attacks. In *Cyber Security And Protection Of Digital Services (Cyber Security), 2017 International Conference on*, pages 1–7. IEEE, 2017.
49. S. Han and M. Philipose. The case for onloading continuous high-datarate perception to the phone. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, 2013.
50. D. Hendrycks, M. Mazeika, and T. G. Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.
51. Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *HotSec*, 2011.
52. B. A. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86. ACM, 1999.
53. IDC. Smartphone OS market share, 2017. `https://www.idc.com/promo/smartphone-market-share/os`.
54. J. Jiang, G. Han, L. Shu, M. Guizani, et al. Outlier detection approaches based on machine learning in the internet-of-things. *IEEE Wireless Communications*, 27(3):53–59, 2020.
55. Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings on Privacy Enhancing Technologies*, 2017(4):177–197, 2017.
56. S. Kotz, T. Kozubowski, and K. Podgorski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer Science & Business Media, 2012.
57. KQED Science. Scientists looked at DNA supposedly from a Yeti and here's what they found, Dec 2017. `https://goo.gl/uDvypP`.
58. J. Lee and C. Clifton. How much is enough? choosing $\varepsilon$ for differential privacy. In *International Conference on Information Security*, pages 325–340. Springer, 2011.
59. J. Li, Y. Zhang, X. Chen, and Y. Xiang. Secure attribute-based data sharing for resource-limited users in cloud computing. *Computers & Security*, 72:1–12, 2018.
60. Y. Li and N. Vasconcelos. Background data resampling for outlier-aware classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13218–13227, 2020.
61. Y. Liu, B. D. Cruz, and E. Tilevich. HTPD: Secure and flexible message-based communication for mobile apps. In *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17*, pages 273–294. Springer, 2021.
62. Y. Liu, B. D. Cruz, and E. Tilevich. Privacy-preserving sharing of mobile sensor data. In *Mobile Computing, Applications, and Services: 12th EAI International Conference, MobiCASE 2021, Virtual Event, November 13–14, 2021, Proceedings*, pages 19–41. Springer, 2022.
63. Y. Liu, B. D. Cruz, and E. Tilevich. Secure and flexible message-based communication for mobile apps within and across devices. *Journal of Systems and Software*, 193:111460, 2022.
64. V. Madani and R. King. Strategies to meet grid challenges for safety and reliability. *International Journal of Reliability and Safety*, 2(1-2):146–165, 2008.
65. D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, 2004.
66. McAfee. Introducing app reputation for Android apps, 2013. `https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/introducing-app-reputation-for-android-apps/`.
67. B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan. Open-TEE–an open virtual trusted execution environment. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 400–407. IEEE, 2015.
68. T. McKay, K. Miller, and J. Tritz. What to do with actionable intelligence: E 2 coach as an intervention engine. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, pages 88–91. ACM, 2012.

69. F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.
70. A. Miller, M. Hicks, J. Katz, and E. Shi. Authenticated data structures, generically. In *ACM SIGPLAN Notices*, volume 49, pages 411–423. ACM, 2014.
71. R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM, 1968.
72. Minitab. Proceed with the analysis if the sample is large enough., 2020. `https://support.minitab.com/en-us/minitab/19/help-and-how-to/statistics/basic-statistics/supporting-topics/normality/what-to-do-with-nonnormal-data/`.
73. S. Mittal, P. K. Das, V. Mulwad, A. Joshi, and T. Finin. Cybertwitter: Using Twitter to generate alerts for cybersecurity threats and vulnerabilities. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 860–867. IEEE Press, 2016.
74. MOCACARE. Blood pressure monitor., 2020. `https://www.mocacare.com/mocacuff/`.
75. P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2012.
76. P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *International Workshop on Public Key Cryptography*, pages 458–473. Springer, 2006.
77. W. Morningstar, C. Ham, A. Gallagher, B. Lakshminarayanan, A. Alemi, and J. Dillon. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics*, pages 3232–3240. PMLR, 2021.
78. G. S. Na, D. Kim, and H. Yu. Dilof: Effective and memory efficient local outlier detection in data streams. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1993–2002, 2018.
79. N. Nguyen. A lot of apps sell your data. here's what you can do about it., 2018. `https://www.buzzfeednews.com/article/nicolenguyen/how-apps-take-your-data-and-sell-it- without-you-even`.
80. Omron. Omron wearable blood pressure monitor, 2020. `https://omronhealthcare.com/products/heartguide-wearable-blood-pressure-monitor-bp8000m/`.
81. D. O'Sullivan. Cloud leak: How a Verizon partner exposed millions of customer accounts, 2017. `https://www.upguard.com/breaches/verizon-cloud-leak`.
82. G. Pang, L. Cao, L. Chen, and H. Liu. Unsupervised feature selection for outlier detection by modelling hierarchical value-feature couplings. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 410–419. IEEE, 2016.
83. A. Paverd, A. Martin, and I. Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.*, 2014.
84. J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.
85. I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association. `http://dl.acm.org/citation.cfm?id=1855711.1855731`.
86. RxJava. RxJava: Reactive extensions for the JVM, 2018. `https://github.com/ReactiveX/RxJava/tree/1.x`.
87. M. Shen, J. Duan, L. Zhu, J. Zhang, X. Du, and M. Guizani. Blockchain-based incentives for secure and collaborative data sharing in multiple clouds. *IEEE Journal on Selected Areas in Communications*, 38(6):1229–1241, 2020.
88. K. Siegrist. Random: probability, mathematical statistics, stochastic processes, 2017.
89. N. Singer. New data rules could empower patients but undermine their privacy, 2020. `https://www.nytimes.com/2020/03/09/technology/medical-app-patients-data-privacy.html`.
90. Sophos Mobile Security. How does app reputation work?, 2016. `https://community.sophos.com/kb/en-us/120915`.
91. N. Statt. Some major android apps are still sending data directly to facebook, 2019. `https://www.theverge.com/2019/3/5/18252397/facebook-android-apps-sending-data-user-privacy-developer-tools-violation`.
92. The New Daily. Federal government to force tech giants to reveal user data, 2018. `https://thenewdaily.com.au/news/national/2018/08/14/tech-surveillance-laws/`.
93. Trend Micro Inc. Using the security scan feature in mobile security for Android, 2018. `https://esupport.trendmicro.com/en-us/home/pages/technical-support/mobile-security-for-android/1111853.aspx`.

94. N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, and K. Papagiannaki. When david helps goliath: the case for 3g onloading. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 85–90. ACM, 2012.
95. VARONIS. 2018 varonis global data risk report, 2018. `https://www.varonis.com/2018-data-risk-report/`.
96. X.-S. Vu and L. Jiang. Self-adaptive privacy concern detection for user-generated content. *arXiv preprint arXiv:1806.07221*, 2018.
97. N. Wendt and C. Julien. Paco: A system-level abstraction for on-loading contextual data to mobile devices. *IEEE Transactions on Mobile Computing*, 17(9):2127–2140, 2018.
98. S. Yang, D. Yan, and A. Rountev. Testing for poor responsiveness in Android applications. In *Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the*, pages 1–6. IEEE, 2013.