Privacy-Preserving Sharing of Mobile Sensor Data

Yin Liu, Breno Dantas Cruz, and Eli Tilevich

Software Innovations Lab, Virginia Tech, USA
{yinliu, bdantasc, tilevich}@cs.vt.edu

Abstract. To personalize modern mobile services (e.g., advertisement, navigation, healthcare) for individual users, mobile apps continuously collect and analyze sensor data. By sharing their sensor data collections, app providers can improve the quality of mobile services. However, the data privacy of both app providers and users must be protected against data leakage attacks. To address this problem, we present *differentially privatized on-device sharing of sensor data*, a framework through which app providers can safely collaborate with each other to personalize their mobile services. As a trusted intermediary, the framework aggregates the sensor data contributed by individual apps, accepting statistical queries against the combined datasets. A novel adaptive privacy-preserving scheme: 1) balances utility and privacy by computing and adding the required amount of noise to the query results; 2) incentivizes app providers to keep contributing data; 3) secures all data processing by integrating a Trusted Execution Environment. Our evaluation demonstrates the framework's efficiency, utility, and safety: all queries complete in <10 ms; the data sharing collaborations satisfy participants' dissimilar privacy/utility requirements; mobile services are effectively personalized, while preserving the data privacy of both app providers and users.

1 Introduction

Mobile services have become a crucial part of the digital economy [2], generating large and growing revenues for application providers [42]. 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 [4]. 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 [8]. For example, numerous location-based apps (e.g., Google Map, Uber, and Yelp) 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.

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 [34,36]. 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 great need and potential benefit in providing holistic mechanisms for sharing mobile sensor data that preserve the privacy of both app providers and users.

To share the sensor data collected by their mobile apps, app providers can use cloud-based services. Each app uploads its collected data to the cloud, which aggregates and analyzes the results. The data privacy concerns of cloud-based processing of sensor data include: (a) cyber attackers can steal uploaded data by exploiting cloud server's vulnerabilities [1]; (b) insiders or careless employees can expose private data to the public [45]; (c) governments can legally force IT companies to reveal their cloud-stored data [43]. In fact, a growing number of privacy tips recommend disabling cloud-based storage and processing altogether with restrictive network access permissions [16] and network blocking apps [17]. Finally, network connectivity/congestion issues can render cloud-based processing infeasible.

Mobile apps can also share their sensor data locally on the same device. This on-device data sharing and processing—referred to as *data onloading*—has been studied widely in the research literature [18, 44, 47] 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" [5] 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) [20]. However, intersections alone are hardly ever sufficient to infer the profiles of mobile users.

In this paper, we present an on-device data-sharing framework, serving as a trusted intermediary that aggregates the sensor data contributed by the collaborating apps, which execute expressive statistical queries against the combined datasets. However, simple statistical queries can be exploited by executing exhaustive frequency queries over a complete finite set [47]. Take the body sensor data as an example: the reasonable range for systolic pressure is between 90 and 180 mm Hg. An app provider can execute queries "how many x mm Hg are there in the combined dataset." After executing 91 possible queries (i.e., different values of x in the range [90,180]), the app provider reveals the whole dataset.

A differential privacy mechanism can alleviate the risks above (e.g., PINQ [27], GUPT [32]). However, differential privacy cannot be applied directly due to the unique challenges of our problem domain: 1) it would be impossible to assign a fixed privacy level to all collaborators, as app providers may have dissimilar privacy/utility requirements; 2) some app providers can infer user profiles from the combined datasets, contributing only minimal data; and 3) attackers can illicitly access or tamper with differential privacy operations and data. To overcome the aforementioned challenges, our approach, 1) achieves the *privacy-utility tradeoffs* that satisfy given privacy/utility requirements, 2) incentivizes app providers to keep contributing data, and 3) secures the execution of statistical queries functions by placing them in a Trusted Execution Environment (TEE), whose trusted storage persists the shared data collections.

Our target is Android[1], on which apps commonly share data with each other [41]. We realize our approach as Go-Between, a system-level service that aggregates the sensor data contributed by 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. Specifically, Go-Between balances the utility and privacy needs of the collaborating apps. Privacy can be increased at the cost of decreasing utility and vice versa. Configured to prioritize privacy, Go-Between increases the amount of noise added to the query results, so the contributed data becomes hard to uncover. Besides, Go-Between incentivizes data contributions: the more data an app contributes, the more accurate and useful its inferred user profiles are. Moreover, Go-Between's TEE-based processing safeguards the predefined statistical queries (e.g., `Count`, `Mean`, and `Std`) and their data. Finally, the end-user remains in control of their data by explicitly restricting apps to share data via Go-Between and being informed of the data sharing events. The contributions of this paper are as follows:

1. An on-device framework for **differentially privatized** sharing of sensor data that is ***(a) usable:*** in dynamically adapting and balancing privacy/utility, as driven by the properties of the contributed data; ***(b) incentivizing:*** in rewarding active contributing app providers with higher utility; ***(c) secure:*** in protecting all shared data and relevant operations in TEE.
2. A general system design for privacy-preserving data sharing and processing, 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; the data sharing collaborations satisfy participants' dissimilar privacy/utility requirements; mobile services are effectively personalized, while preserving the privacy for both app providers and end users.

2 Go-Between Overview

To motivate our approach, we present two typical application scenarios and how Go-Between addresses their requirements. Then, we overview the differential privacy theory & technologies used by Go-Between.

[1] Android mobile platform takes ≈85% of the global mobile market [21].

2.1 Typical Application Scenarios

(1) Geolocation: one of the most common types of sensor data, enables app providers to infer location-based properties of a user profile (e.g., favorite areas). To optimize personalization, app providers frequently collect and share geolocations. An empirical study has revealed that within 14 days, geolocations were shared 5,398 times across 10 different apps, which included not only mapping/navigation apps, but also social media (e.g., Facebook) and shopping apps (e.g., Groupon) [3].

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.

(2) 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 [31, 35]. 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 [40].

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[2] specify their privacy and utility requirements (e.g., high privacy and medium utility), and then share their sensor data (e.g., geolocation/blood pressure datasets) with Go-Between, which aggregates the shared data into combined datasets for app providers to query. Go-Between differentially privatizes the query results in accordance to both the properties of the shared 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 share a certain type of sensor data. Go-Between maintains a trusted record of all apps the user has authorized to share data. A permitted app can query the combined dataset contributed by itself with a particular 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[3]; (2) upon each data deposit, Go-Between starts computing the noise scale for each built-in query operation; (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] 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, so we can use the terms "app provider" and "app" interchangeably.

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

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:

App Providers. When app providers share sensor data, the process 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* [38]: an adversary contributes valid data but tries to learn all possible information about the combined dataset. Specifically, we assume that neither party is malicious: store fake data to render Go-Between useless. However, competing with each other, any app provider can gain a business advantage by uncovering the data in possession of its collaborators. For example, through legitimate frequency queries for each discrete data item, an app can discover the data collections possessed by the other apps.

(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.

Mobile Users. Irrespective of how app providers share sensor data, mobile users deeply care about (a) which part of their data will be shared and (b) which app providers are involved in the sharing process.

Behavioral Model. As mentioned above, we assume that the app providers act as *honest-but-curious parties* that would not maliciously contribute fake data. To personalize mobile services in the absence of cloud-based processing, their apps can collaborate via Go-Between to improve the quality of their services and deliver a mutually beneficial outcome. Hence, rationally behaving apps would avoid any actions that would distort the subsequent statistical queries and cause all parties involved to lose out as a result. Further, by making use of machine learning or deep learning algorithms to identify outliers that deviate from the general data distribution, modern anomaly detection (i.e., outlier detection) can identify and exclude those apps that maliciously contribute fake data [19, 37]. In addition, we envision app providers forming a contractual obligation for using Go-Between that regulates resource contribution, profit distribution, and commercial credibility, explicitly proscribing the intentional depositing of fake sensor data. These mechanisms create a disincentive for apps to maliciously deposit fake data.

Countermeasures. To ensure the data privacy of app providers, our approach introduces 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 (§ 3.1 § 3.2).

(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 (§ 3.3)

(c) to prevent the combined datasets from being illicitly accessed, all shared data and relevant operations take place in a Trusted Execution Environment (TEE) (§ 4.1).

(d) 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

(1) Differential Privacy (DP) protects an individual's private information[4] from unauthorized discovery [10]. 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

[4] Hereafter, *individual* refers to *an app provider*, and *private information* refers to the sensor data collected by a provider.

neighboring databases if they differ by exactly one record. A *mechanism* M is a randomized function that maps D to output R.

Definition 1: ε-differentially private mechanism. Given $\varepsilon \geq 0$, M is ε-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 ε_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 $f : D \rightarrow R$; D,D' are *neighboring databases*, the global sensitivity of f is: $\Delta f = Max_{D,D'} |f(D) - f(D')|$ (3)

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

Definition 4: upper bound of ε [24]. 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., ρ), the number of records (n), the global sensitivity of query f (i.e., Δf), and the maximal difference between query results of each possible combination of D' (i.e., Δv): $\varepsilon \leq \frac{\Delta f}{\Delta v} ln \frac{(n-1)\rho}{1-\rho}$ (4)

(2) Laplace Mechanism [11] 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 μ. The *Laplace distribution* 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 μ and use $Lap(b)$ as the added noise.

Definition 5 — noise scale. To satisfy ε-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 μ with the actual result of query $f(D)$, we can get the privatized value: $f(D) + Lap(\Delta f/\varepsilon)$ that ensures the ε-differential privacy.

Definition 6 — noise scale for a query sequence. To satisfy ε-differential privacy for a query sequence $f_1, ..., f_n$, use scaled symmetric noise: $Lap(\sum_i \Delta f_i/\varepsilon)$ (6)

(3) Trusted Execution Environment (TEE) [13] 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* [26] 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 Applying & Complementing DP

We first explain by example how we apply differential privacy (DP) to defend against the aforementioned honest-but-curious attacks. Then we discuss how we complemented DP to meet the privacy requirements in our target domain.

3.1 From Theory to Practice

(1) Honest-but-curious attacks: We further develop the scenario in § 2.1 that shares body vitals. Consider the worst-case scenario: only two apps—H and M—share their collected blood pressure readings.

As shown in Figure 1, 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

D

App Provider	Systolic Pressure
H	150
H	140

D'

App Provider	Systolic Pressure
H	150
H	140
M	150

Fig. 1. The worst-case scenario of the attack.

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."

(2) 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. 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.

3.2 Privacy & Utility Tradeoffs.

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

(1) Privacy. Definition (4) determines the upper bound of ε, 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), ρ 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. Δv is the maximal difference between the query results of each possible combination of D'. Thus, n and Δv can be calculated based on the dataset's properties. ρ can be configured by apps in order to control the privacy level based on their specific requirements.

(2) 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)

The collaborating apps can set the required accuracy level. After adding noise, if the result of a query cannot meet the accuracy requirement 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 ε-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* (§ 3.3-3) below for details.

3.3 Data Contribution Incentives

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

1. The privacy level ρ 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.
2. The amount of contributed data should be commensurate with the obtained utility.
3. 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.

(1) Global Privacy Level: For each collaborating app, we define a *contribution rate* (c): $c_i = \frac{\omega_i}{\sum \omega_i}$ (9), where ω_i is the amount of data contributed by the ith 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 ρ_i is the *privacy level* configured by the ith app.

The *global privacy level* is used to calculate the noise scale (b) by using (7). 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 contribution from specifying the lowest privacy level with the goal of accurately inferring user profiles.

(2) 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)

(3) Noise Increase Scheme: Since every query invocation accumulates ε (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 ε-differential privacy.

4 System Design & Implementation

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

4.1 Architecture

Figure 2 and the code snippet below show Go-Between's architecture and programming interface, respectively:

`Mobile Apps` configure their privacy and utility requirements, persist their individual collections of sensor data, and perform reactive 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 (step 3) via the `Encoding Protocol` 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).

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. This feature runs on dedicated worker threads, synchronized by means of Android's `Handler` and `Message`. In addition,

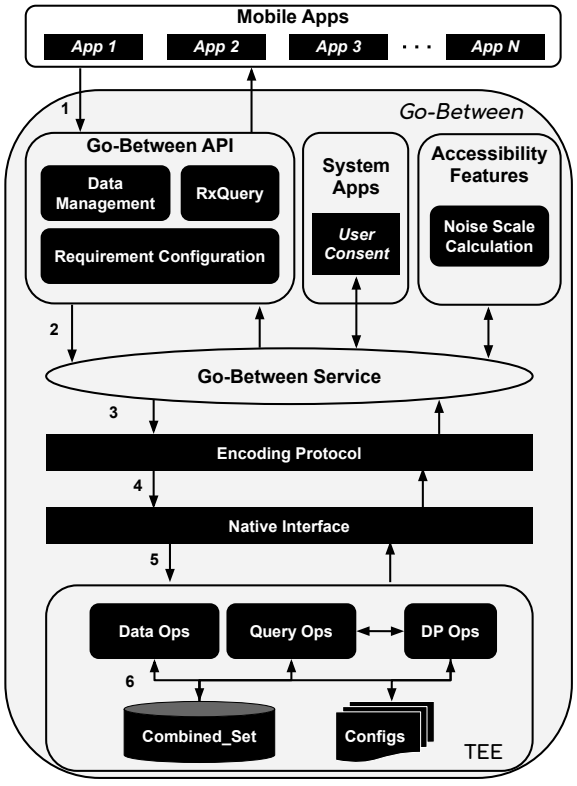


Fig. 2. System Architecture.

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/decline 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. 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.

4.2 Privacy Mechanism

Go-Between's adaptively parameterized privacy mechanism: 1) configures each app's privacy and utility requirements, 2) determines the required noise scale and pads the query results with the suitable amount of noise.

(1) 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*, ε-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[α]		Accuracy[β]		Usage Freq.[γ]	
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

[α] *Privacy level* is the probability of an adversary correctly discovering the combined dataset's raw data, used to calculate the noise scale.

[β] *Accuracy level* is the *% error* as per formula (8), a noisy result's utility for inferring user profiles.

[γ] *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.

(2) 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., ρ_{global}) for the combined dataset contributed by the collaborating apps, 2) use ρ_{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 § 3.3).

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., ρ_H), `Highest` (i.e., ρ_W), and `Default` (i.e., ρ_M), respectively. Go-Between first looks up the actual probability values for these human-readable levels as per Table 1:ρ_H=5%, ρ_W =1%, ρ_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 (as per *formula-10*): $\rho_{global}=c_H\rho_H+c_W\rho_W+c_M\rho_M$, where c is the data *contribution rate* of each app. Because Go-Between updates ρ_{global} whenever new apps join an existing data sharing collaboration, ρ_{global} always reflects the actual privacy requirement of the collaborating apps.

Having determined the global privacy level (ρ_{global}), Go-Between plugs the resulting value into the *formula-7* (§ 3.2) that calculates the noise scale of each predefined query: $b = \Delta v/ln\frac{(n-1)\rho}{1-\rho}$, with ρ becoming ρ_{global}, and n becoming the *size of the combined dataset*. To determine Δv, $n-1$ 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 Δv 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*). For example, each time app H performs `Mean` on the combined dataset, Go-Between ensures ε-differential privacy by adding the *Laplace noise* to the query result with the noise scale: $b_H = u_H \cdot (b_{mean} - c_H \cdot BONUS)$.

5 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?

5.1 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[5] for testing and debugging Android apps. Because its standard distribution excludes a TEE component, we integrated Open-TEE[6] 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 [15]) and has cumulatively covered 80.2% devices (cumulative distribution [14]).
(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. Without loss of generality, we assume a standard setup: device sensors are enabled, and the involved apps are granted permissions to collect sensor data.
(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. 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.

5.2 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

[5] a common practice for studying various performance trade-offs on the Android platform [6].
[6] a portable framework for code to run on any standardized TEEs.

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 `storeData`, `Mean`, `Std`, `CountOne`, and `CountFreq` Go-Between operations. Go-Between calculates noise scales 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[7], who according to Nepali folklore haunts the Himalayas [23]. As it turns out, the Yeti is a sophisticated and demanding mobile user. Due to the need 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, Yeti demands highly personalized services that customize the actively used apps to his 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 ρ_H), `Default` (i.e., accuracy level a_H), `Default` (i.e., usage frequency level u_H). Then, we simulate the attack scenario — *Revealing raw data:*

Because the reasonable range for SYS is between 90 and 180 mm Hg, H's competitor app C performs `CountOne`[8] 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 ρ_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[9], 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.

5.3 Results

Q1. Feasibility: As discussed in § 5.2, 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.3, neither of the predefined queries exceeds 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

[7] The Yeti is a metaphor that describes any real-world user with a similar behavioral profile in this application scenario.

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

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

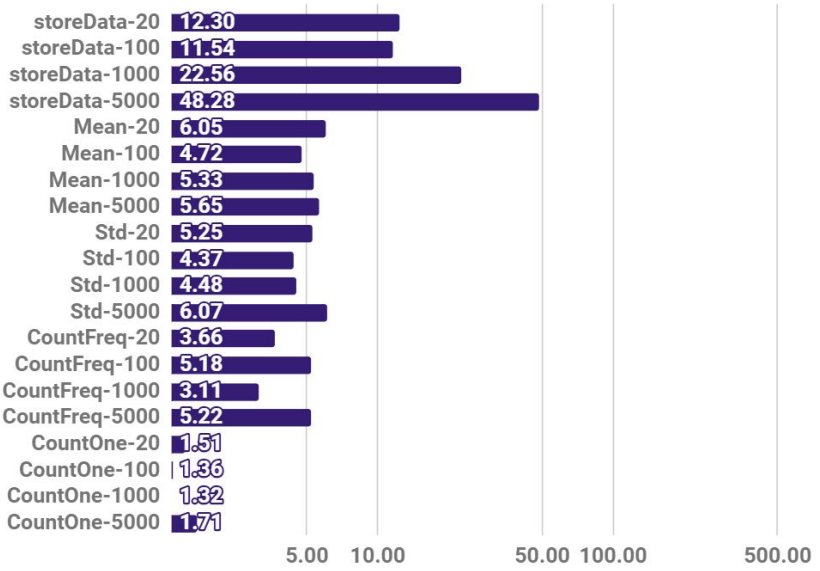


Fig. 3. Performance (log scale with millisecond).

(response time<5s) and expected by end-users (response<1s [29]); even the longest observed response time taken by `store-Data` (≈48 ms) is still within these boundaries.

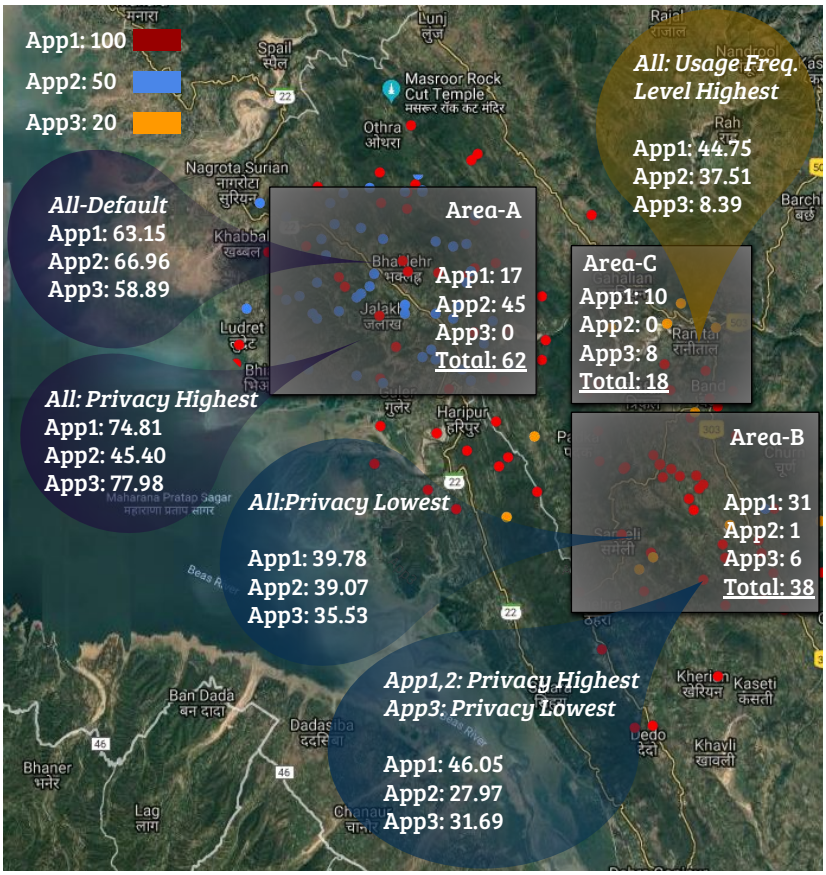
Q2. Utility: As shown in Fig. 4 (in square), without Go-Between, the query of "how many times the Yeti visited a given area", performed by App1, App2, and App3 returns 17, 45, and 0, respectively, for *Area-A*; 31, 1, and 6 for *Area-B*; and 10, 0, and 8 for *Area-C*. Hence, App1 would think *Area-B* 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. 4 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 ε-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, $n_1 ¡ n_2 ¡ n_3$. Go-Between adds the least noise to App1's result, making it most accurate (i.e., 63.15 is the closest to the actual result of 62). Although it cannot be guaranteed 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. 4, 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*.

As discussed in § 4.2, 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. 4—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 § 4.2, 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 it, we let all apps set their usage frequency to "Highest", meaning that the query can be repeated up to a 100 times. The results



(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. 4. Utility of Go-Between.

(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 perform continuous frequent invocations of queries whose results would not be as accurate.

Q3.Safety: *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 2 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).

Table 2. 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.

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.

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.

6 Discussion

(1) Data privacy of the app provider and the mobile user: Since all shared 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 framework enables app providers to accomplish their business objectives in a privacy-preserving fashion, thus implicitly improving end user privacy.
(2) Applicability: *(a) For other platforms:* although our reference implementation is Android-based, our approach's complemented differential privacy (§ 3) and system design (§ 4) are applicable to any Android release or other mobile platforms. *(b) For other scenarios:* our approach can benefit other data sharing scenarios. For example, sharing data at the edge, (e.g., smart home, autonomous driving), often involves mutually distrustful parties that can apply our approach to personalize their services by sharing data without compromising their data privacy. *(c) For other queries:* our functional and reactive query interfaces can be extended for additional queries, thus further increasing our approach's applicability.

7 Related Work

Data Privacy. Differential privacy [10] prevents attackers from discovering private information. Based on this concept, Airavat [39] provides differential privacy guarantees for cloud-based MapReduce computations. The PINQ [27] and GUPT [32] libraries add a unified amount of noise to raw data for privacy-preserving data analysis. Vu et al. [46] automatically detect the user's privacy requirements by determining the noise scale with a neural network model. Miller et al. [28] provide security protocols for clients to securely communicate with a non-trusted server. Gaboardi et al. [12] 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 ε-differential privacy augmented with *privacy & utility tradeoffs* 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. [22] 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. For example, CrowdShare [7] 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 [9,25,33]. In contrast, to share data, Go-Between relies neither on encryption nor on the SFE techniques. Distrustful parties can effectively collaborate via Go-Between

that balances their privacy & utility requirements by automatically determining the required noise scale.

8 Conclusion

We have presented on-device sharing of sensor data to personalize mobile services while protecting data privacy. Powered by an adaptive privacy mechanism, our approach: (1) satisfies the dissimilar privacy and utility requirements of the collaborating apps, (2) incentivizes the apps to keep contributing data, and (3) protects sensitive data and operations in a trusted execution environment. The evaluation demonstrates our approach's feasibility, utility, and safety.

Acknowledgements

The authors thank the anonymous reviewers, whose insightful comments helped improve this paper. NSF supported this research through the grant #1717065.

References

1. CVE-2016-6540. (2016), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6540
2. Acquisti, A., Taylor, C., Wagman, L.: The economics of privacy. Journal of economic Literature **54**(2), 442–92 (2016)
3. Almuhimedi, H., Schaub, F., Sadeh, N., Adjerid, I., Acquisti, A., Gluck, J., Cranor, L.F., Agarwal, Y.: 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. pp. 787–796 (2015)
4. Anderson, C., Andersson, M.P.: Long tail (2004)
5. Apple Inc.: App groups entitlement (2017), https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_security_application-groups
6. Armando, A., Merlo, A., Migliardi, M., Verderame, L.: Would you mind forking this process? A denial of service attack on Android (and some countermeasures). In: IFIP International Information Security Conference. pp. 13–24. Springer (2012)
7. Asokan, N., Dmitrienko, A., Nagy, M., Reshetova, E., Sadeghi, A.R., Schneider, T., Stelle, S.: Crowdshare: Secure mobile resource sharing. In: International Conference on Applied Cryptography and Network Security. pp. 432–440. Springer (2013)
8. Binns, R., Lyngs, U., Van Kleek, M., Zhao, J., Libert, T., Shadbolt, N.: Third party tracking in the mobile ecosystem. In: Proceedings of the 10th ACM Conference on Web Science. pp. 23–31 (2018)
9. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of CRYPTOLOGY **13**(1), 143–202 (2000)
10. Dwork, C.: Differential privacy: A survey of results. In: International Conference on Theory and Applications of Models of Computation. pp. 1–19. Springer (2008)
11. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography conference. pp. 265–284. Springer (2006)
12. Gaboardi, M., Honaker, J., King, G., Nissim, K., Ullman, J., Vadhan, S.: PSI: a private data sharing interface. arXiv:1609.04340 (2016)
13. GlobalPlatform: GlobalPlatform, TEE system architecture, technical report. www.globalplatform.org/specificationsdevice.asp (2011)
14. Google: Android studio – select the minimum api level (2018), https://developer.android.com/studio/projects/create-project
15. Google: Distribution dashboard (2018), https://developer.android.com/about/dashboards
16. Google: Connect to the network (2019), https://developer.android.com/training/basics/network-ops/connecting
17. Google Play Store: Noroot firewall (2019), https://play.google.com/store/apps/details?id=app.greyshirts.firewall&hl=en
18. Han, S., Philipose, M.: 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)
19. Hendrycks, D., Mazeika, M., Dietterich, T.G.: Deep anomaly detection with outlier exposure. arXiv preprint arXiv:1812.04606 (2018)
20. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM conference on Electronic commerce. pp. 78–86. ACM (1999)

21. IDC: Smartphone OS market share (2017), https://www.idc.com/promo/smartphone-market-share/os
22. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. Proceedings on Privacy Enhancing Technologies **2017**(4), 177–197 (2017)
23. KQED Science: Scientists looked at DNA supposedly from a Yeti and here's what they found (Dec 2017), https://goo.gl/uDvypP
24. Lee, J., Clifton, C.: How much is enough? choosing ε for differential privacy. In: International Conference on Information Security. pp. 325–340. Springer (2011)
25. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al.: Fairplay-secure two-party computation system. In: USENIX Security Symposium (2004)
26. McGillion, B., Dettenborn, T., Nyman, T., Asokan, N.: Open-TEE–an open virtual trusted execution environment. In: Trustcom/BigDataSE/ISPA, 2015 IEEE. vol. 1, pp. 400–407. IEEE (2015)
27. McSherry, F.D.: Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. pp. 19–30. ACM (2009)
28. Miller, A., Hicks, M., Katz, J., Shi, E.: Authenticated data structures, generically. In: ACM SIGPLAN Notices. vol. 49, pp. 411–423. ACM (2014)
29. Miller, R.B.: Response time in man-computer conversational transactions. In: Proceedings of the December 9-11, 1968, fall joint computer conference, part I. pp. 267–277. ACM (1968)
30. 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/
31. MOCACARE: Blood pressure monitor. (2020), https://www.mocacare.com/mocacuff/
32. Mohan, P., Thakurta, A., Shi, E., Song, D., Culler, D.: GUPT: privacy preserving data analysis made easy. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 349–360. ACM (2012)
33. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: International Workshop on Public Key Cryptography. pp. 458–473. Springer (2006)
34. Nguyen, N.: 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
35. Omron: Omron wearable blood pressure monitor (2020), https://omronhealthcare.com/products/heartguide-wearable-blood-pressure-monitor-bp8000m/
36. O'Sullivan, D.: Cloud leak: How a Verizon partner exposed millions of customer accounts (2017), https://www.upguard.com/breaches/verizon-cloud-leak
37. Pang, G., Cao, L., Chen, L., Liu, H.: Unsupervised feature selection for outlier detection by modelling hierarchical value-feature couplings. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 410–419. IEEE (2016)
38. Paverd, A., Martin, A., Brown, I.: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Tech. Rep. (2014)
39. Roy, I., Setty, S.T.V., Kilzer, A., Shmatikov, V., Witchel, E.: Airavat: Security and privacy for mapreduce. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. pp. 20–20. NSDI'10, USENIX Association, Berkeley, CA, USA (2010), http://dl.acm.org/citation.cfm?id=1855711.1855731
40. Singer, N.: 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
41. Statt, N.: 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
42. marketing team, F.: How much money can you earn with an app in 2019 (2019), https://fueled.com/blog/much-money-can-earn-app/
43. 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/
44. Vallina-Rodriguez, N., Erramilli, V., Grunenberger, Y., Gyarmati, L., Laoutaris, N., Stanojevic, R., Papagiannaki, K.: When david helps goliath: the case for 3g onloading. In: Proceedings of the 11th ACM Workshop on Hot Topics in Networks. pp. 85–90. ACM (2012)
45. VARONIS: 2018 varonis global data risk report (2018), https://www.varonis.com/2018-data-risk-report/
46. Vu, X.S., Jiang, L.: Self-adaptive privacy concern detection for user-generated content. arXiv preprint arXiv:1806.07221 (2018)
47. Wendt, N., Julien, C.: Paco: A system-level abstraction for on-loading contextual data to mobile devices. IEEE Transactions on Mobile Computing **17**(9), 2127–2140 (2018)