A Privacy-Preserving Personal Sensor Data Ecosystem

by

Brian M. Sweatt

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2014

Author . . . .
Department of Electrical Engineering and Computer Science
May 22, 2014

Certified by . . . .
Alex "Sandy" Pentland
Professor
Thesis Supervisor

Accepted by . . . .
Prof. Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

A Privacy-Preserving Personal Sensor Data Ecosystem

by

Brian M. Sweatt

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2014, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Despite the ubiquity of passively-collected sensor data (primarily attained via smartphones), there does not currently exist a comprehensive system for authorizing the collection of such data, collecting, storing, analyzing, and visualizing it in a manner that preserves the privacy of the user generating the data. This thesis shows the design and implementation of such a system, named openPDS, from both the client and server perspectives. Two server-side components are implemented: a centralized registry server for authentication and authorization of all entities in the system, and a distributed Personal Data Store that allows analysis to be run against the stored sensor data and aggregated across multiple Personal Data Stores in a privacy-preserving fashion. The client, implemented for the Android mobile phone operating system, makes use of the Funf Open Sensing framework to collect data and adds the ability for users to authenticate against the registry server, authorize third-party applications to analyze data once it reaches their Personal Data Store, and finally, visualize the result of such analysis within a mobile phone or web browser. A number of example quantified-self and social applications are built on top of this framework to demonstrate feasibility of the system from both development and user perspectives.

Thesis Supervisor: Alex "Sandy" Pentland
Title: Professor

Acknowledgments

There are honestly too many people to acknowledge, so I'll preface this section by stating up-front that I'm likely missing at least one person (probably many) who deserve mention and brought me to where I am today. So, they get the first acknowledgment - if you know me, and you're reading this, odds are good that I owe a great deal of who I am today to you and I love you for it.

Sandy Pentland, my thesis advisor and academic / research / business / life mentor over the last 2 years deserves more recognition than I could possibly provide here. His considerate understanding of my unorthodox path back to and through school, and the lengths that he's gone to accommodate it have both surprised and humbled me many times. I can't count the number of times he's "gone to bat" for me over my time here, whether it was finding new opportunities for sharing and collaborating in my work, or finding new opportunities to keep my bills paid. Combined with the fact that he's literally creating a new field of science, in addition to already being a pioneer in a number of areas in computer science, yet still makes time to meet with and guide me and his other students, I don't think I could imagine a better advisor if I tried.

My girlfriend, Ellen Wong, has kept me balanced over the course of my degree. She's been very understanding and encouraging every step of the way - since before I even made up my mind about coming back to school right up until today. She's also seen and talked to me just about every day in between, so the fact that she still even wants to see my face after the countless rants, vents, and my trademark obsession with work, makes me love and appreciate her all the more.

My family, both adopted and blood-related, have been my own personal cheerleading squad. The Rivas family - mom, pop, Danny, and Katie (and Pepper!) - have been beyond supportive since the day they took me into their family. My brother, Tim Sweatt, has been one of my biggest role models (in addition to being my biggest fan) since before I can even remember, and he and his wife, Amanda, have have made me feel at home every time I visit. My sister, Erica Cassidy, has been a supportive voice of reason every time I get the chance to see her, and has, most importantly, been incredibly easy-going and understanding when I can't. Despite not being with me anymore, my parents helped make me the man I am today, and live on in the encouragement, memories, and pride they provide every time I think of them.

My close friends, whom I also consider family: Eddie Urteaga, Louis Avila, Danny Rivas, Guy Cross, Juan Grande, Ubaldo Cruz, Mike Cross, Anthony Roldan, and John Hufnagel are all credited with providing necessary down-time, whether it's via Halo, or via a trip to Jersey. Eddie has been especially supportive, and my call logs can attest to that, whether it's for fun or to check in on how far I am on my thesis.

Finally, my MIT and Media Lab cohorts deserve mention: Anne Hunter, Nicole Freedman, Elizabeth Bruce, and Sam Madden for their help and working with my unique circumstances. Jeff Schmitz, Yves-Alexandre de Montjoye, Arek Stopczynski, Vivek Singh, Oren Lederman, Erez Shmueli, Laura Freeman, Ankur Mani, Chazz Sims, Ali Kamil, Layla Shaikley, Joost Bonsen, Sharon Paradesi, Ilaria Liccardi, Dazza Greenwood, and Denise Cheng for their discussions, collaborations, and distractions, which helped me a great deal during my time here.

Contents

List of Figures

List of Tables

Listings

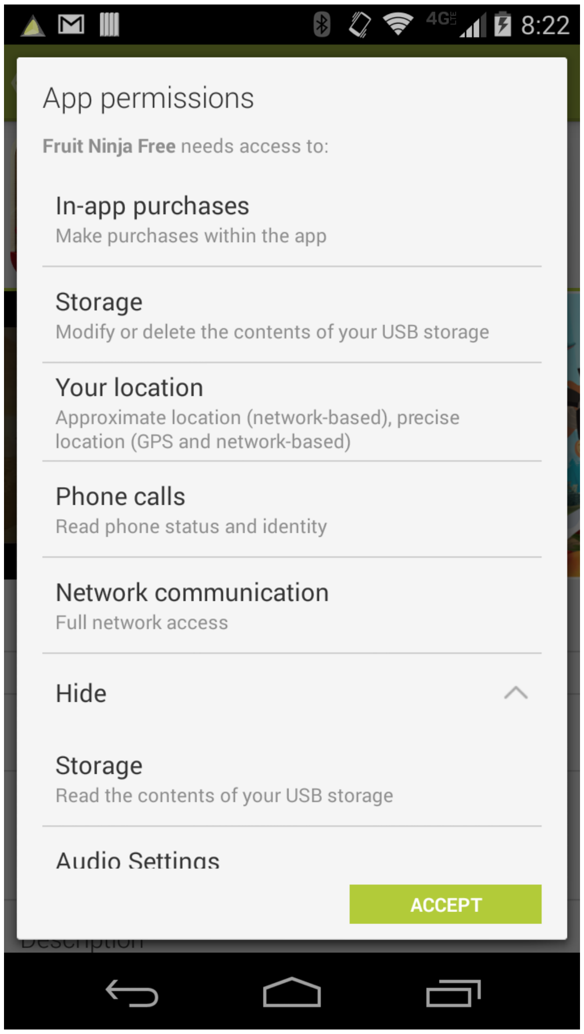
Chapter 1

Introduction

1.1 Motivation

With the advent of smartphones and the multitude of sensors they contain, humans now have an unprecedented ability to observe our own behavior - be it places we frequently visit, people we frequently contact, or even how physically active we are. Computational social science, the study of such data to discern insights about social behavior, has the potential to increase our own (or a third party's) knowledge of individuals and communities, with an unprecedented breadth, depth, and scale [33]. Additionally, this data collection can (and should, for the purpose of observing behavior) all be done without requiring any explicit interaction from the users generating the data. Such passive data collection provides the basis for quantifying how we go about our daily routines, and its collection is also ubiquitous - the vast majority of applications running on mobile devices today have access to sensitive personal data [47], users typically are unaware of the extent of the collection [42], and furthermore, the number of application whose livelihood (via ad-serving) depends on such data collection is increasing [7].

Despite the ubiquitous collection of sensor data by a vast number of mobile applications running on literally billions of devices worldwide, users currently have little to no insight into what data is actually being collected on them - the permissions system for granting access to personal data on the current leading smartphone OS in terms of market share (Android), follows a ”take it (everything) or leave it” model, whereby users must grant an application all permissions it requests or opt out of using the application entirely. This applies to all permissions an app requests, regardless of if the permission is of crucial importance to the app's function, or if the service the app provides contributes the user's data to aggregate computations. Figure 1-1 shows the permissions screen for a popular Android game that requests access to sensitive personal data, with no apparent use for it in the game itself.


App permissions

Fruit Ninja Free needs access to:

In-app purchases
Make purchases within the app

Storage
Modify or delete the contents of your USB storage

Your location
Approximate location (network-based), precise location (GPS and network-based)

Phone calls
Read phone status and identity

Network communication
Full network access

Hide

Storage
Read the contents of your USB storage

Audio Settings

Figure 1-1: Android permissions screen - Users must opt-in to all permissions in order to use an app, regardless of if the data is essential to the app's function

After granting this blanket access to their personal data, users have no insight into how often an application is querying their data and storing it on third party machines outside of their control. Even in the presence of this information on data collection, the user still would have little insight, aside from conjecture, on how their data is being used. This status quo provides all value that can be gleaned from the data back to the service the individuals used to generate the data. A more transparent means of data collection; allowing individuals to collect data for themselves and participate in group computation on an opt-in, privacy-preserving basis, provides ownership of the data and any value it provides back to the individuals first. This would not inherently preclude the individual from providing insights derived from the data (or the data itself) to services or researchers, but puts the individual in control of their sensitive personal data and identity.

1.2 Relevant Research

1.2.1 Living Labs and Personal Sensor Data

Gathering of information about a community of users in an attempt to learn about them and humans in general, as well as attempting to improve the participants' lives is not a new practice. In fact, a leading traditional dataset from one of the most ubiquitous studies, in terms of duration and number of participants, is the Framingham Heart Study, which ran for decades across thousands of participants [13]. However, traditionally, data has been costly to collect in terms of time and effort, so the throughput of data in such sets is typically low - on the order of single-digit samples per user, per year. Due to the relatively small or low-resolution data gathered from these traditional studies, the opportunity to test new interventions in a controlled and timely manner was not possible.

Recent advances in technology have allowed the study of higher through-put datasets - even in the absence of ubiquitous sensor-laden smartphones, mobile phone record datasets donated by mobile carriers can provide data on millions of users with a very high sampling rate. Though this data is typically not on an individual basis, and provides little to no additional contextual information about the users, they can nonetheless tell us interesting things about human behavior. For example, by using carrier mobility traces to characterize human mobility, Gonzalez et al. showed that humans follow simple, reproducible mobility patterns [27]. Call detail records (CDR) can likewise be used to discern trends in communication patterns, such as the work by Eagle et al., which combined these records with national census data to find a strong correlation between diversity in individual relationships and economic development [19]. Figure 1-2, compiled by Aharony et al. [1], shows an overview of a number of social science datasets with respect to sample size, duration, and data throughput. Within this figure, The Framingham Heart Study [13] occupies the far-end of the horizontal axis (7), with thousands of participants over decades of study, but low data throughput, while the Social fMRI dataset (aka "Friends and Family") [1] occupies the upper-right of the graph (3), with a shorter duration and fewer participants, but much higher data throughput.

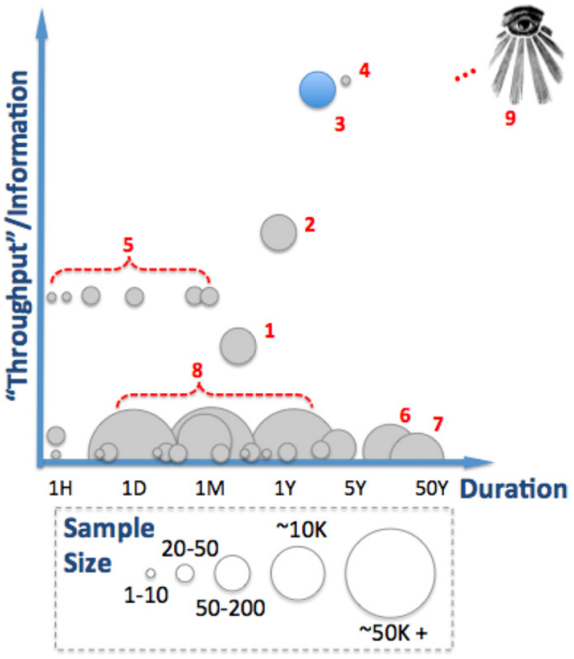


Figure 1-2: Social science dataset throughput - data collected in traditional studies typically have low throughput, despite having potentially many participants and long durations. Recent studies leveraging mobile sensor data have much higher throughput, but have been limited in duration and number of participants. [1]

Since mobile carrier companies continuously generate user data as part of their standard bookkeeping practices, there is the potential to move discoveries about the data from the traditional post-hoc domain into the realm of realtime analysis and interventions. A Living Lab seeks to do just that - turning a community of individuals opting into data collection, experiments, and research into a testbed for experiments and interventions, with the ability to see resulting changes in data in realtime. There are a number of impediments to opening this data entirely, and they include privacy concerns, legal liability for use of the data, and the ability to re-identify users even in sets of data that have been very carefully anonymized [33]. While concerns on the part of corporations limit the data that they release, smartphone applications have access to even higher-resolution sensor data and operate via an agreement between the user running the application and the application provider themselves, removing the requirement that data be gathered by a third party corporation, such as a mobile operator.

There are a number of recent examples of successful living labs built on top of sensor data collected directly from mobile phones. Aharony et al. collected high-resolution sensor data for 150 individuals over the course of 15 months. Their results agree with the aforementioned result by Eagle et al. in regards to a correlation between social diversity and economic development [1,19], in addition to finding a relationship between the number of apps individuals share and their face-to-face interaction time, implying that app adoption may spread via such interactions. Furthermore, the authors performed an intervention within the community of participants to incentivize physical activity and found that peer-reward social influences, which can be measured via sensor data, are an effective means of influencing behavior change. The dataset from this work has been opened as part of the Reality Commons project [39], providing an avenue for post-hoc research to be done. Recently, this dataset was utilized to find that social features measured via mobile phones can be better predictors of spending behavior than personality traits in couples [45], and that similar mobile-sensed features can reliably predict daily happiness for individuals when combined with information about the weather and personality traits [6].

The Trentino Mobile Territorial Lab project is a living lab consisting of over 150 participants in Trento, Italy that is ongoing and being continuously studied [38].

Participants in this living lab have opted in to passive data collection directly from their mobile phones, as well as through their carrier's traditional data collection practices. The project is of particular relevance due to how the users' personal data is managed and the tools provided to each of the participants surrounding their personal data and its use. As part of the living lab, users have access to a web interface that allows them to view the personal data collected about them, as well as opt-in to studies and research. Additionally, data in the study is stored in a user-centric manner - data about an individual is stored alongside all other data about that individual, regardless of the source. The ongoing nature and, as a consequence, the growing corpus of personal data for study provides the opportunity to cut out the typical months-long process of gathering participants and data about them for study.

1.2.2 Privacy-Preserving Data Handling

Higher resolution and higher throughput datasets naturally expose more information about the underlying participants contributing to the datasets. Under the strictest definition of privacy-preservation, access to a dataset should never enable an attacker to learn more about a target than they could without access to the dataset [11]. While this criterion has been proven impossible [18], the trade-off between information provided in a dataset and privacy afforded to the participants contributing to the dataset has been well-studied.

There are 2 models for privacy-preserving data publishing: that of the untrusted and the trusted data publisher [25]. In the untrusted case, the entity publishing the data is assumed to have malicious intent and thus, privacy-preservation necessarily relies on never exposing the data in an unencrypted fashion to even the data publisher(s). To this end, much of the work surrounding the untrusted case has been on encryption solutions for data collection [51] resulting in homomorphic encryption work [26] for mining the data, anonymous communications [9, 29], and statistical methods for anonymized data collection [50] [23]. A typical issue with encryption schemes is a loss of generality - solutions for mining data must be discovered on a case-by-case basis for each algorithm and insight, while anonymous communications remove metadata that provides the receiver context needed to understand the information, and statistical methods attempt to reconstruct representative data from the dataset based on perturbed samples, resulting in a loss of accuracy in the final solution.

Given the low cost of collecting and publishing data today, self-publishing of personal data is not only possible, but may be a better means of assuring data is being handled in a manner the individual contributing the data is comfortable with. Self-published data sets would naturally follow the trusted data publisher model and, as they are the sole source of data for this thesis, the remainder of this review will focus on the trusted model.

A standard practice when publishing a dataset containing personal information is to anonymize the dataset [10, 12]. For this, it is not sufficient to simply remove all explicitly identifying information from records within the dataset, as individuals could potentially still be re-identified from quasi-identifier attributes within a record - data that narrows down the likely owners for a particular record, potentially to a single person within the dataset [49]. To this end, a prolific property for assessing the privacy of a published dataset is k-anonymity, under which each person's data within a published dataset cannot be distinguished from at least k-1 individuals also in the dataset [41, 49]. To achieve k-anonymity, publishers typically either performed suppression - removing offending attributes from the table - and/or generalization - sufficiently lowering the resolution of the attribute to the point that at least k records alias to the same value for that attribute.

While k-anonymity is a necessary property for an anonymized dataset to have, it is not necessarily sufficient for privacy preservation. For example, in the absence of sufficient diversity amongst records within a dataset (the case of a very rare disease in a medical records database), individuals could still be re-identified from the values of these attributes. As such, Machanavajjhala et al. [35] identified l-diversity, the requirement that each set of quasi-identifier attributes must contain at least l distinct values, as a means to further anonymize datasets. Likewise, l-diversity is flawed in the face of a skewness attack on the data, in which an attacker can narrow down the set of likely individuals with a given set of quasi-identifiers based on the distribution of values for sensitive attributes within the subset. To combat this, Li et al. [34] proposed t-closeness - the requirement that the distribution of all all sensitive attributes within any subset of quasi-identified data must be close to the distribution for that attribute in the entire dataset (”close” meaning within t, calculated via the Earth Mover Distance).

A trend starts to appear in the work for anonymizing datasets. Namely, for each property that must be met prior to publishing a dataset, a new type of attack may be constructed to re-identify individuals within the dataset, or the data will be degraded to the point that useful insights about the data cannot be discerned (a common complaint with t-closeness [23]). These shortcomings are further exacerbated when dealing with fine-grained, high-resolution sensor data, such as location. In fact, de Montjoye et al. [14] showed that only 4 spatio-temporal points are necessary to uniquely identify 95% of 1.5M people in a mobility database that exposed locations at the cell-tower level (low accuracy). Furthermore, their study showed that lowering either time or space resolution within the dataset had little effect on the number of points needed to identify individuals. Traditional means of preventing location data leakage, such as frequently changing pseudonyms [5] and cloaking [28] have likewise proven to be inadequate [32, 44].

When viewed under the lens of failed attempts at anonymizing datasets, one may begin to consider alternative means of handling sensitive personal data. Self ownership and publishing, along with tools to manage one’s own privacy, appear to be a necessary step towards providing awareness to users about the nature of what their data says about them, allowing them to control its dissemination, and removing liability from organizations that would be held accountable for sensitive data leaks in the world of centralized personal data publishing. This thesis, and indeed the openPDS framework that it implements [16], provides a further step: disallowing the outright publishing of the raw sensor data, and instead providing compute space within an individual’s trusted personal data store for approved analysis to run.

Chapter 2

Personal Data Stores

Personal data stores seek to solve multiple issues present in the current state of online personal data. In this section, these issues are described, along with potential high-level solutions.

2.1 Principles

2.1.1 Personal data ownership

An individual's personal data are currently spread across the multitude of services that individual uses.Their social graph, calendar, and location history are typically stored by, and under the control of, the service the individual used to generate the data. For this reason, any entity seeking a more holistic view of an individual's online life (as well as offline via sensor readings) must obtain authorization by the user to access each service's data in its raw form, and perform analysis across the different data in a separate location outside of the individual's control. This process can be complicated; in the best case, each service has its own API, and the individual or third-party must learn each of them before pulling the data together for analysis. In the worst case, a service will not provide a publicly-accessible API, and the user must resign to either not use that specific service's data, or resort to screen-scraping, which is error-prone, and can even violate the terms of use for a given service. Additionally, for the case of a third-party aggregation service, the user must certify that the new entity is trustworthy and will not share their data or use it for other nefarious purposes once it has been provided. Figure 2-1 shows the current state of online personal data collection and use.

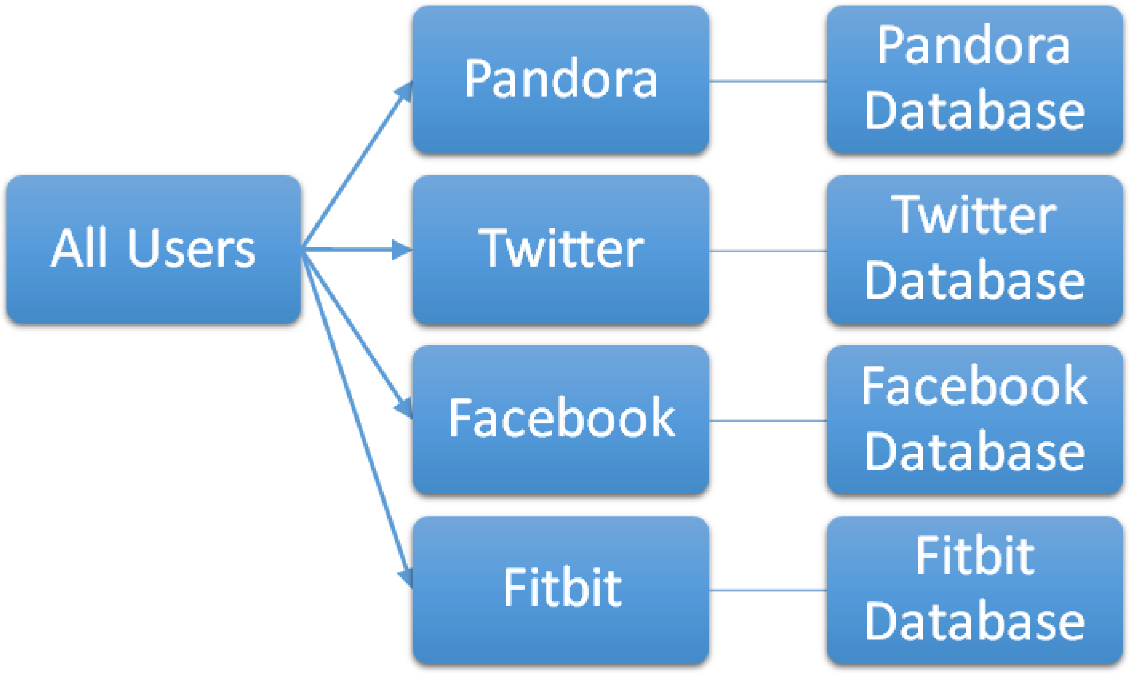


Figure 2-1: Service-centric personal data storage - data from all users are stored by, and under control of the services the individual used while generating the data

Personal Data Stores seek to solve the problem of providing personalized services based on a complete view of the individual's data by providing an user-centric storage model for such data as opposed to the service-centric storage model that is ubiquitous today. In this manner, openPDS provides a unified location and interface for an individual to store their personal data, and run analysis against it. Additionally, it provides a framework for authorizing third parties to access their data. Figure 2-2 shows the state of personal data storage in the presence of ubiquitous personal data stores.

In addition to physical storage of individuals' data, personal data services today typically stipulate that the data remains under control of the company providing the service, rather than the individual generating the data. In this manner, the company

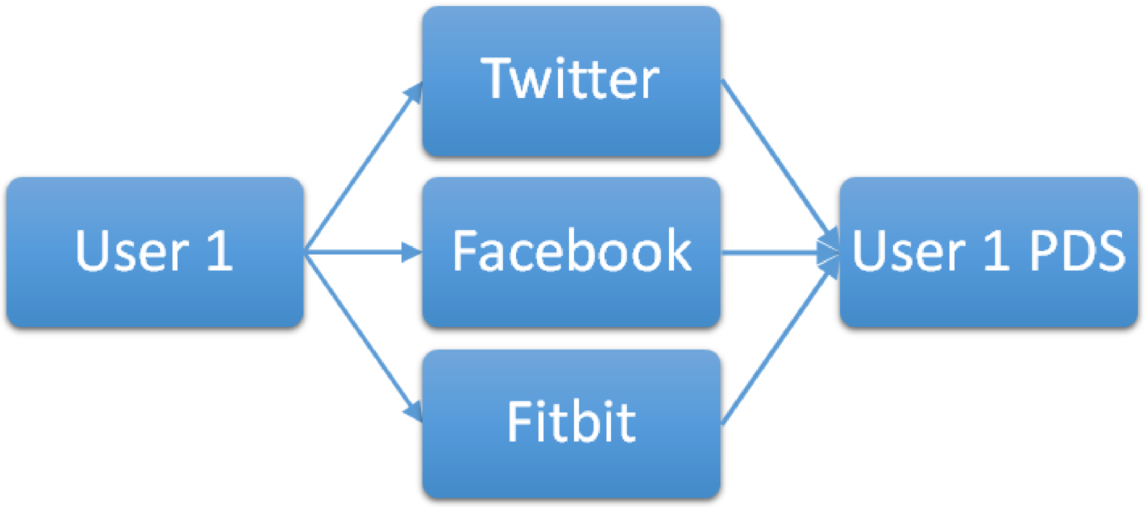


Figure 2-2: User-centric personal data storage - data from all services an individual uses are stored by, and under control of, the user generating the data

has the right to retain data after a user has requested deletion, use the identified data internally, and share the data in an anonymous aggregate (or even an identified fashion, if the terms of use stipulate it), all without the user's express consent. Such egregious stipulations exist in Terms of Service across the web today that an entire movement and website thrives around the notion of providing a grade and simple, human-readable evaluations of what a service's terms actually mean [40].

2.1.2 Informed consent based on purpose for data use

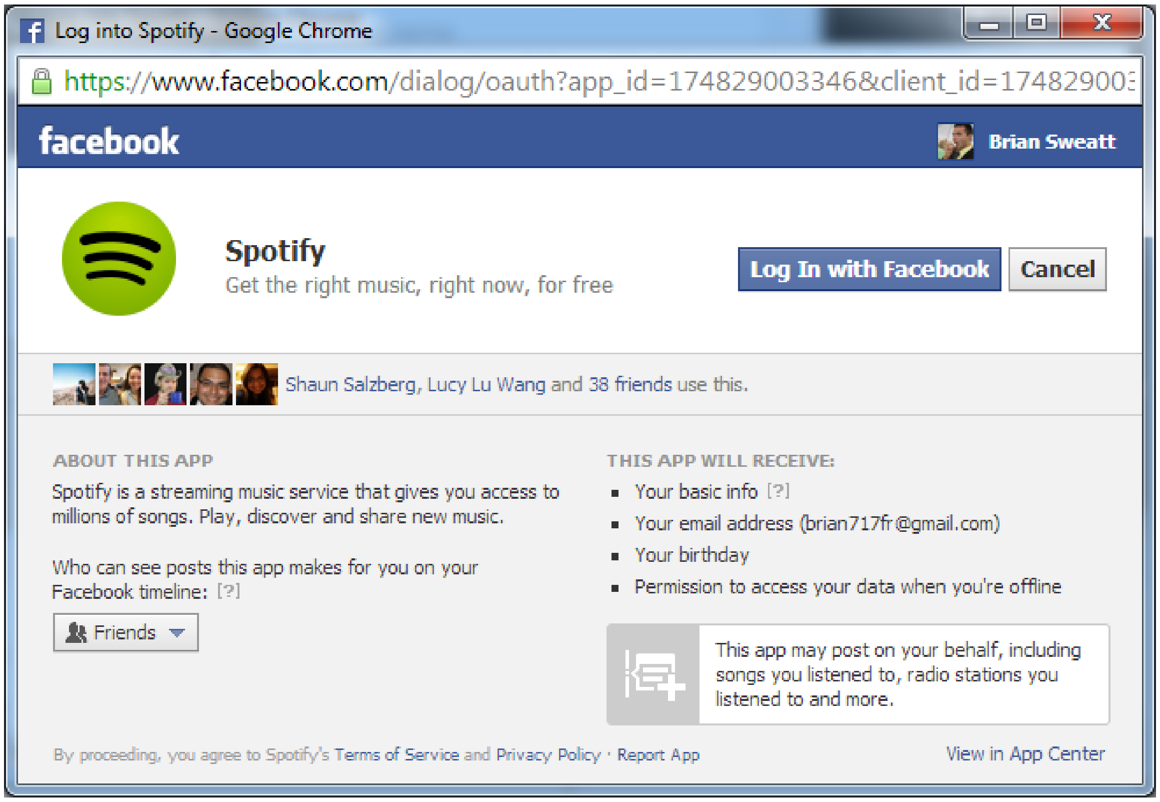
Informed consent is a cornerstone of ethical data collection in both clinical and research settings [21]. In order to properly collect data on and about individuals, it is a researcher's duty to:

1. Obtain voluntary agreement from participants for enrollment into services that collect their data
2. Provide adequate information about the nature of the data collection and the service that will be provided prior to seeking participants' agreement

The second of the two requirements above implies that there is a burden on the part of the party wishing to use the data to adequately educate participants before opting them in to data collection. It is not enough to simply provide this information to the participants - the service seeking to use the data must assure to the best of their abilities that the user actually understands the terms of their agreement [46]. As previously described, terms for such services are typically very lengthy and written by legal entities with little to no regard for actual comprehension of the text. The critical leap towards actual informed consent that these terms of service fall short of is comprehension. Even if an explanation of all data being collected is provided, users may not know the implications of allowing this data to be collected - they lack (understandably so) comprehension of the purpose for which this data can and will be used by the collecting party.

The vast majority of personal data services on the web use role and scope-based authorization to provide a means for users to authorize third parties to use their data. In typical authorization, individuals authorize roles (typically a third party app acting on the user's behalf) to access specific scopes, which correspond to the source and type of data, with little to no insight or control over how that raw data is used once the service surrenders it. For example, Facebook, a popular personal data service, provides scope-based authorization via OAuth that allows third-party applications to access a user's personal data generated on the site. Controls are provided via scopes - each corresponding to a different type of data from the user's Facebook profile. In the Facebook case, a user's first and last name have their own scope, as well as the user's birthday and hometown. Figure 2-3 provides an example of the user experience for an application requesting access to a user's personal data.

In a typical scope-based authorization flow, the user authenticates against a service's identity provider, and is presented a user interface the delineates all types of data a given third party is requesting access to. However, besides describing the data type and source they control, scopes are limited in their ability to control actual operations on the data - scopes can be generated corresponding to typical CRUD (create, read, update, delete) operations, but little can be done to support more complex or semantic purposes. In order for a user to fully understand the consent they are giving to a third party when they agree to the terms of service, a user must have a concept


Log into Spotify - Google Chrome

https://www.facebook.com/dialog/oauth?app_id=174829003346&client_id=17482900

facebook

Brian Sweatt

Spotify

Get the right music, right now, for free

Log In with Facebook

Cancel

Shaun Salzberg, Lucy Lu Wang and 38 friends use this.

ABOUT THIS APP

Spotify is a streaming music service that gives you access to millions of songs. Play, discover and share new music.

Who can see posts this app makes for you on your Facebook timeline: [?]

Friends

THIS APP WILL RECEIVE:

- Your basic info [?]
- Your email address (brian717fr@gmail.com)
- Your birthday
- Permission to access your data when you're offline

This app may post on your behalf, including songs you listened to, radio stations you listened to and more.

By proceeding, you agree to Spotify's Terms of Service and Privacy Policy · Report App

View in App Center

Figure 2-3: Typical authorization flow - blanket access to specific types of data are provided to third-parties

of the purpose for which the third party intends to use the data. Without a full understanding of how the data can be used, the user cannot conceivably give their informed consent for use of their data.

2.1.3 Privacy-preserving computation

Self-ownership of personal data, along with complete control over how it is used, including fine-grained access control on the data type and purpose level, provides a means for users to help manage their personal data privacy. However, these tools alone do not provide any guarantee that code run against an individual's data adheres to the stated purpose, or that their privacy is being preserved in an aggregate computation across other users' data.

In a service-centric architecture, there are no technical guarantees that an individual is unidentifiable in the dataset that each service exposes either internally to employees at the company providing the service, or externally to third parties that interact with the service. This is due to a lack of control over the dataset(s) being exposed. User-centric storage helps remedy that problem by providing a complete view of the dataset being exposed. By adding a trusted compute space within an individual's personal data store, the individual or an outside party can effectively audit all code being run against their data and determine if they are comfortable both with the data that the computation requests in raw form, as well as the dimensionality-reduced resulting dataset that the third party submitting the code is requesting access to.

2.2 Recent Work

User-centric personal data storage and management has a long history that starts in 1945 with Memex [8], which sought to manage personal data in a pre-digital world via a mechanical framework. As this pre-dates massive aggregation, the focus was on efficient means for capturing (via a head-mounted camera), storing, and retrieving personal data, rather than managing the role of the individual in mass data aggregation. Research began exploring collective intelligence based on individual personal data in the 60's [20, 36], still well before the age of ubiquitous data collection, and modern privacy research.

More recent work has focused on the personal data store as a means of maintaining user privacy, ownership, and control over personal data, brought on by the modern age of mass data collection, and the emergent service-centric storage for storing the sensitive data. Allard et al. described a vision of decentralized personal data stores residing on embedded devices physically owned by individuals, with supporting servers for asynchronous communication, durability, and global processing [2]. An extension of this work also proposes implementations of the embedded physical devices within smartphones and set-top boxes [3]. A personal data store for an "Internet of Subjects" was described in [30], and a small-scale test to determine willingness amongst users to adopt the described PDS in a job-application scenario was explored in [31].

Direct pre-cursors to the work presented in this thesis were discussed in [16].

In particular, the authors introduce the openPDS architecture and the concept of a question answering framework running within a user's personal data store. The architecture and its implications for privacy, as well as use cases for developers and end-users are further described in [15]. The work contained in this thesis represents a direct extension to this work in terms of scalability, performance, and experience for both end-users and developers.

Chapter 3

Implementing a Personal Data Ecosystem

This section covers the core openPDS architecture, beginning with definition of terms, and proceeding to a description of the functional components necessary to build a cohesive personal data store solution. The functional components are organized into distinct sections: authentication service and Registry, individual decentralized Personal Data Stores, and REST service interfaces for both.

3.1 Nomenclature

Authorization Server A server issuing OAuth 2.0 access tokens to the Client after successfully authenticating the user and obtaining authorization.

Client In the OAuth context, an application or system making protected requests on behalf of the ser and with his or her authorization.

OAuth 2.0 An open protocol to allow secure API authorization in a simple and standard method from desktop and web applications. It enables users to grant third-party access to their web Resources without sharing their passwords.

Personal Data Store A protected resource owned and controlled by an individual to hold their personal data. The user controls access to use, modification, copying, derivative works, and redaction or deletion of the data they enter into the Personal Data Store, including data collected from their smartphone via either passive sensor collection or surveys.

Registry Account management and database of registered users. This server stores only data that is necessary to authenticate a user (email and password hash) and locate their PDS, and has an internal as well as an external identifier for each user. The Registry authenticates login requests, as part of the OAuth authorization flow.

REST Service A type of web service that is stateless, cacheable, and provides a uniform interface to resources. Such services use URLs to uniquely identify resources and standard HTTP methods of GET, POST, PUT, and DELETE to specify the operation to perform on a resource.

Scope When an individual authorizes access to data on their personal data store, the access token also includes one or multiple named Scopes, each designating a type of data access that has been authorized.

Symbolic User ID Indirect reference to a registered users OAuth and Registry identifier key. Also referred to as a participants UUID.

User ID A participants OAuth and Registry identifier key. It is accessed only internally by openPDS and the Registry server. When Entities other than the Participant or System Entity need to reference it, they are given a Symbolic User ID.

3.2 High-Level System Diagram

Figure 3-1 provides a high-level overview of all stakeholders in a general personal data ecosystem. This includes third parties providing data and services, users in the ecosystem (shown as participants), and applications acting on behalf of users and services.

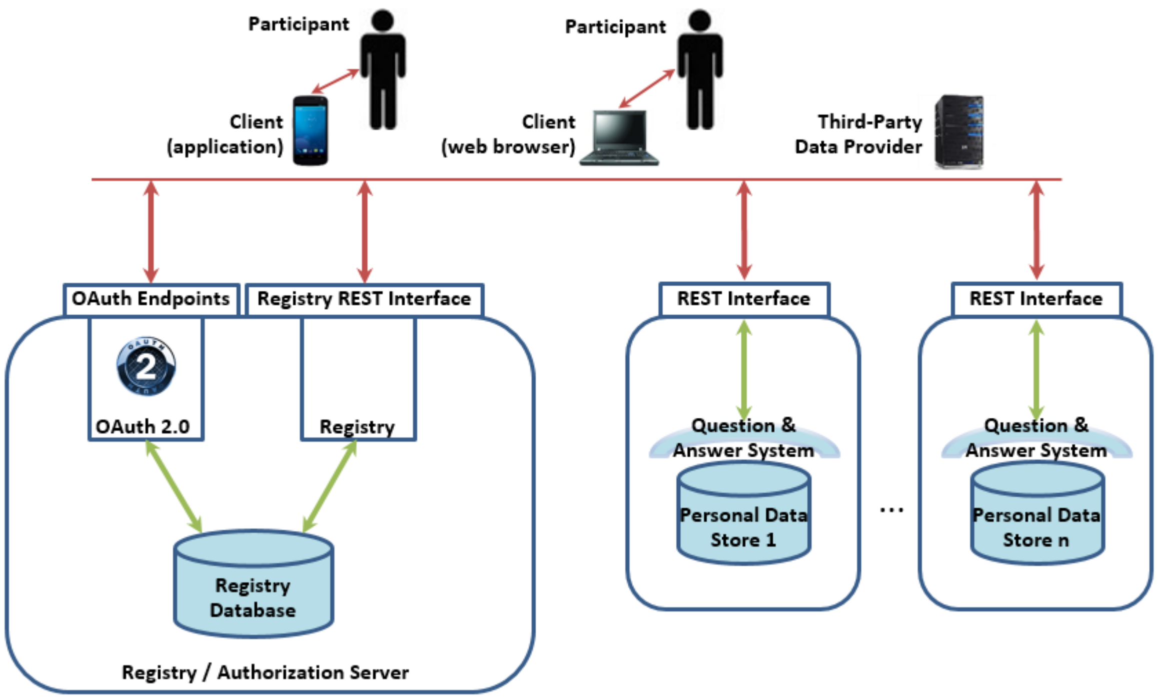


Figure 3-1: High-Level personal data ecosystem - a set of decentralized personal data stores constitute a trust framework with a centralized registry of users and authorization service.

Subsequent sections explain each of the blocks called out in figure 3-1, including Registry and authorization, personal data stores and their associated question and answer system, as well as clients and data providers built on top of the ecosystem. APIs are described to facilitate communication between physical servers in a personal data ecosystem, as well as how data is organized in personal data stores, and the means by which one may define new applications or data connectors to run on personal data stores.

3.3 Registry and Authorization

An authorization server, supporting the OAuth 2.0 protocol, provides secure user authentication and authorization of access to personal data stores. The OAuth component is tightly coupled with a registry providing account management services. As part of these account management services, the registry includes a database of registered users, holding login credentials for authentication, and profile data.

Account management services include profile edits, password changes, and recovery procedures for forgotten passwords. Administrators have full access to these services, while other participants may have access only a restricted subset.

At the point of user registration, the registry creates a profile for the user, and a personal data store is lazily initialized for them. The user profile contains the following information:

Username used to login to the registry server and personal data ecosystem. Must be unique.

Email used for account verification purposes. Must be unique.

Password hash

First name

Last name

UUID anonymous symbolic identifier used to address this user across the personal data ecosystem. Is unique by definition.

PDS Location URL identifying where the user's distributed personal data store resides

Additionally, to facilitate authorization, the registry server also holds information about the groups a user belongs to, as well as the roles the user can take on within the system. After registration, authorization for collecting and utilizing data by an app on the user's behalf follows the flow described in figure 3-2. The registry server also holds all information pertaining to the flow described therein - including all access tokens and authorization codes for all users in the ecosystem. These users can revoke authorization tokens and, in turn, revoke the access they had provided to their personal data. Each request against any server in the personal data ecosystem must provide an Authorization header of the form: `Authorization: Bearer <token>`, where `<token>` is a placeholder for a valid authorization token.

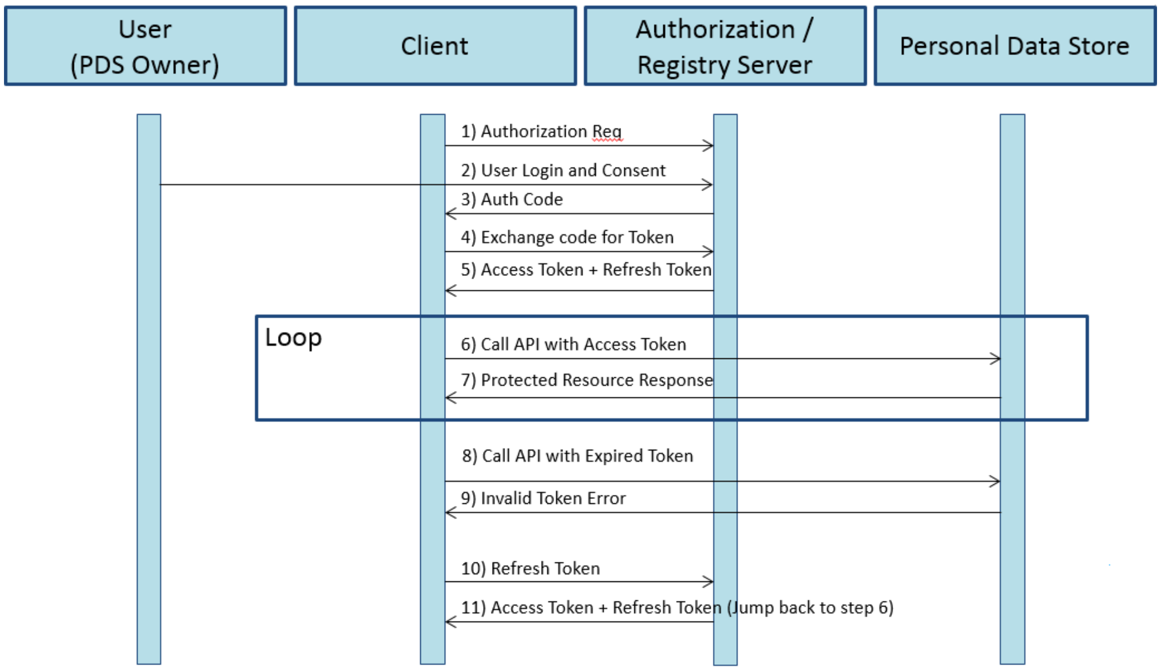

User (PDS Owner)
Client
Authorization / Registry Server
Personal Data Store
1) Authorization Req
2) User Login and Consent
3) Auth Code
4) Exchange code for Token
5) Access Token + Refresh Token
Loop
6) Call API with Access Token
7) Protected Resource Response
8) Call API with Expired Token
9) Invalid Token Error
10) Refresh Token
11) Access Token + Refresh Token (Jump back to step 6)

Figure 3-2: General Authorization Flow - a registered user authorizes an application to act on their behalf. The application then presents an authorization token, which can be refreshed as needed, when accessing the user's data.

In addition to the traditional scopes associated with authorization tokens, the registry server stores an additional parameter for each authorization token: purpose. From the registry, this is a unique human-readable string that identifies logic run within users' personal data stores. This logic is further described in the question and answer framework section.

For accessing data, the system supports scopes at both the resource and key level. For resource types (such as Funf or Facebook data), a scope named `funf_write` may denote access to write funf data to a user's personal data store. Additionally, to control access to the results of analysis on the raw data within a personal data store, the PDS supports scopes at the key level as well. In this manner, a one-to-one correspondence exists between answers to questions on the PDS and scopes for accessing those questions (a "socialhealth" answer would have a corresponding "socialhealth" scope).

3.4 Distributed Data Stores

After a user authorizes a third party to store or access their personal data, all subsequent communication is done directly with that user's personal data store using the granted authorization token, as shown in figure 3-2. Storage for personal data within this location is done in a user-centric manner; while a single openPDS server supports storage and analysis on multiple user's data, each user has a separate backend database, and separate, user-specified encryption keys for all personal data residing in the store, regardless of the data source. In this manner, personal data stores can either have a one-to-one correspondence between physical hosting machines and users, or an arbitrary number of physical machines may provide a logical one-to-one correspondence by sharing physical resources between users. Data from a given user hosted on the same physical machine as other user's may be accessed with a combination of the user's UUID and a valid OAuth token. Figure 3-3 provides an overview for reference by the subsequent design and implementation sections.

3.4.1 Storage

In order to support data with an arbitrary schema, MongoDB is the primary backend storage technology. MongoDB is a non-relational ("No-SQL") data store consisting of JSON-formatted documents identified with a unique ID, provided by the MongoDB server running within the PDS. In order to support queries against this data based on time and type of data, each piece of raw data stored in this system has a key denoting the type of data, and a timestamp denoting when the particular sample of data was taken. Given these constraints, relational backends with fixed schemas can also be supported.

To this effect, a SQLInternalDataStore base class has been implemented with corresponding PostgresInternalDataStore and SQLiteInternalDataStore subclasses. Schemas for relational backends are specified in a backend-agnostic manner in order to support the multitude of different SQL dialects, or other query languages, such as SPARQL. A backend-agnostic schema for a table is represented as a Python dictio-

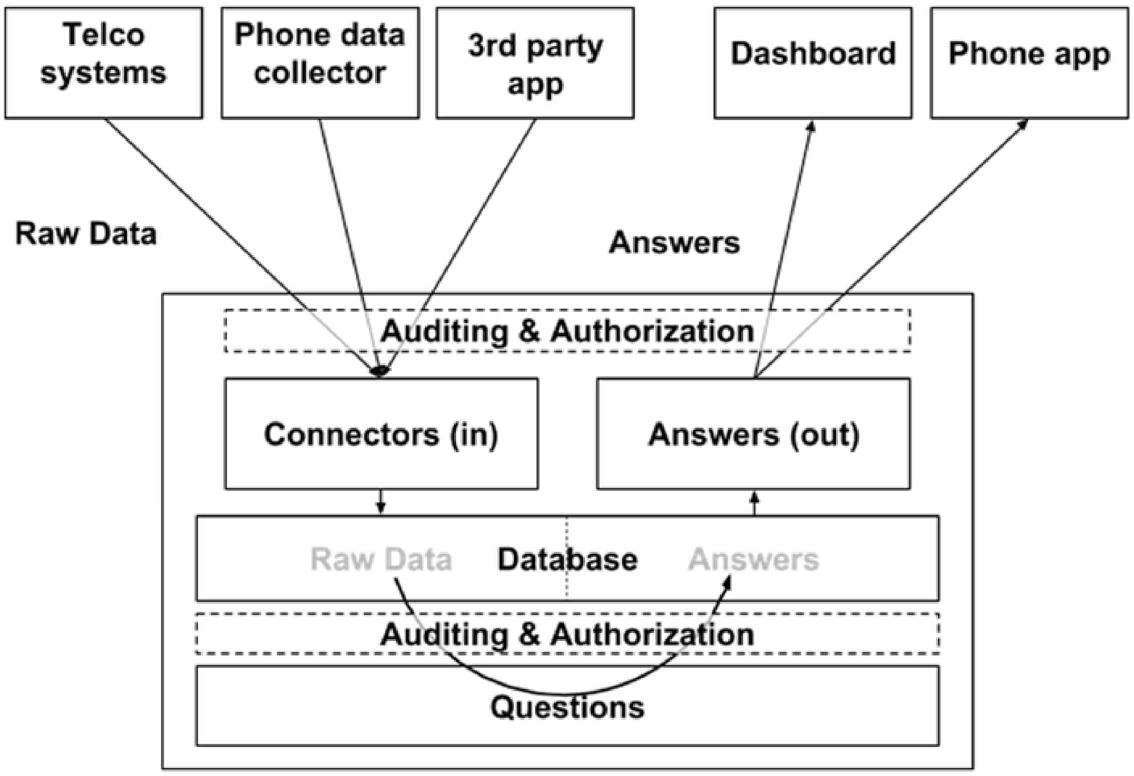


Figure 3-3: Detailed PDS architecture - Connectors handle data input from external sources, Answers handle output of processed data, Questions handle populating these answers, each step checks for authorization and records an entry in an audit log.

nary with a name, a list of column tuples containing the name and data type for the column, as well as an optional mapping list for retrieving data from a field that does not match the name provided for the column. An example backend-agnostic schema definition for the CallLogProbe table is provided below:

Listing 3.1: Example backend-agnostic schema definition.

```
CALL_LOG_TABLE = {
    "name": "CallLogProbe",
    "columns": [
        ("_id", "INTEGER"),
        ("name", "TEXT"),
        ("number", "TEXT"),
        ("number_type", "TEXT"),
        ("date", "BIGINT"),
        ("type", "INTEGER"),
        ("duration", "INTEGER")
    ],
    "mapping": {
        "funf": {
            "number_type": lambda d: d["numbertype"]
        }
    }
}
```

3.4.2 Connecting a Data Source

The personal data store currently supports storing data obtained from the Funf framework running on Android phones via a connector. In order to extend the system to support a new type, a connector for that data source must be written, and a user must authorize the new application to access their PDS. These steps are described below.

Authorization to write to the PDS

As all requests to the PDS must contain a bearer token and the UUID of the data store owner as query string parameters (`bearer_token` and `datastore_owner__uuid`, respectively). The endpoint written for the new connector must check for the presence of these querystring parameters and verify them against the registry server that provided them. The endpoint must specify an OAuth scope corresponding to the type of data it is collecting (Funf uses `funf_write`, for example), and the token provided must have access to that scope. Scopes can be created by signed-in users on the registry server at `/admin/oauth2app/accessrange/`.

Extending InternalDataStore

To handle a new data source, the internal API for storing and retrieving data, InternalDataStore, must be updated to handle the new type of data. In most cases, this entails specifying either a new collection in Mongo to hold the data or, for relational backend support, a new SQL table. For relational backends, the schema for the new data type must be provided as described in section 3.4.1. Authorization against the created scope, connections to the database, and queries against the new collection or table are then provided automatically by the parent InternalDataStore implementation.

Exposing an endpoint for the new connector

After getting authorization to access a PDS, a means of transferring data to it must be provided. This is typically done by exposing a REST endpoint on the PDS for the data source device to POST to. The endpoint will typically accept a POST request of raw data, along with PDS credentials (`datastore_owner__uuid` and `bearer_token`), authenticate the request using the `PDSAuthorization` and `PDSAuthentication` modules, and write the data if the request is valid. For the case of encrypted files, this will also involve a decryption step that can either be done synchronously while the PDS processes the request, or queued up for asynchronous decryption by a separate process.

3.4.3 A Two-Level API for Accessing Personal Data

As figure 3-3 shows, accessing data within a personal data store follows a two-step process.

Internal API

As a means of preserving the privacy of the owner, openPDS prohibits data from leaving the store in its raw form. Instead, in order to enforce data use for a specific purpose, the system provides a trusted compute space that a third party may submit code and queries to as a means of computing answers to questions about the data. This compute space is shown within figure 3-3 as the “Questions” block. Each time raw data is accessed within this space, the data store checks with the registry server to assure the question is allowed access to the raw data it is requesting, and stores an audit log of the request and the code's corresponding purpose. Access to raw data within the trust compute space is provided via the InternalDataStore interface:

class InternalDataStore	
Method	Description
`__init__(profile, token)`	Initializes an InternalDataStore for the user associated with profile, using authorization provided via token.
`getAnswer(key)`	Retrieves a dictionary representing the answer associated with the given answer key, if it exists, or None if it doesn't.
`getAnswerList(key)`	Retrieves a list representing the answer associated with the given answer key, if it exists, or None if it doesn't.
`saveAnswer(key, answer)`	Stores the given answer under the provided answer key.
`getData(key, start, end)`	Retrieves data of the type specified by key, recorded between the start and end times.
`saveData(data)`	Saves data to this InternalDataStore. Data must specify a key and time in order to be accessed later via getData.
`notify(title, content, uri)`	Sends a notification to all devices registered for this user.

Table 3.1: InternalDataStore provides an interface for accessing raw data and answers within the trusted compute space

External API

Upon accessing the raw data within the question-answering framework, third party code has the ability to store the result of the computation as an answer. These answers provide the second layer of data access on the PDS. While question answering occurs privately within the data store's trusted compute space, answers to questions populated within that space are available via the answer and answerlist REST APIs on the data store. In order to gain access to the answers stored using an InternalDataStore, a client must list that answer's key as a scope when requesting a bearer token to access the user's PDS. This token, as with all requests against a user's PDS, is included on the request to the answer or answerlist endpoints, where the PDS checks for the requisite scope and stores an audit log of the request prior to returning the pre-computed data. Table 3.2 documents the full external REST APIs available to clients via standard HTTP requests.

External PDS REST endpoints		
Endpoint	Methods	Description
`/api/personal_data/answer`	GET POST PUT DELETE	Retrieves or stores the answer for a given key as a JSON-formatted dictionary.
`/api/personal_data/answerlist`	GET POST PUT DELETE	Retrieves or stores the answer for a given key as a JSON-formatted array.
`/api/personal_data/funfconfig`	GET POST PUT DELETE	Retrieves or stores Funf configuration objects.
`/api/personal_data/device`	POST DELETE	Registers the cloud-messaging identifier (`gcm_reg_id`) for a client device.
`/api/personal_data/notification`	GET DELETE	Retrieves all outstanding notifications from the PDS.

Table 3.2: The PDS provides a number of external endpoints for client applications to interface with.

Each REST endpoint in the external API takes `datastore_owner__uuid` and the client's OAuth token as `bearer_token` as querystring parameters. For the case of answer endpoints, the key for the desired answer must also be provided for GET requests. For all endpoints, the authorization layer within the PDS assures that the given bearer token has access to the necessary scopes to complete the request, and an audit log is stored to keep track of the request and the data provided. For the device endpoint, clients with the approved scope can register devices or delete their registration, but cannot perform GET or PUT requests to retrieve or update pre-existing devices. Likewise, the notification endpoint only provides support for retrieving and deleting notifications; notifications may only be added via the internal

API.

3.4.4 Writing Questions

Questions are structured as asynchronous tasks run on a pre-defined schedule within the user's personal data store. The system makes use of Celery, an asynchronous job queue and processing service running continuously within the personal data store's trusted compute space. As such, a question within the personal data store is a Python method that takes an InternalDataStore as parameters and has a `@task()` attribute attached to it. An example task for computing the number of probe entries for a given user is provided below:

Listing 3.2: Example question for pre-computing an answer from a single user's data.

```python
@task()
def recentProbeCounts(internalDataStore):
    start = getStartTime(2, False) #Timestamp from 2 days ago
    probes = [
        "ActivityProbe",
        "SimpleLocationProbe",
        "CallLogProbe",
        "SmsProbe",
        "WifiProbe",
        "BluetoothProbe"]
    answer = {}
    for probe in probes:
        #Look up all data for the given probe (no end time)
        data = internalDataStore.getData(probe, start, None)
        answer[probe] = data.count()
    internalDataStore.saveAnswer("RecentProbeCounts", answer)
```

A file containing each such question is submitted to the personal data store, along with a schedule corresponding to each task that specifies the frequency with which it runs and updates the answer it populates. The personal data store then schedules each task to run and allows each to only update answers with keys that the provided token has specified as scopes. For the case of a single personal data store hosting a number of user's data, the data store iterates over all users hosted on the given PDS install and runs each question installed for that user before moving on to the next user. In this manner, the backend storage can take advantage of time locality between references to similar data in order to maximize the amount of data present in the backend storage's cache, if it exists.

3.4.5 Group Computation

Data and answers about that data, are often more useful when aggregated across different users. However, the ability to perform such aggregation is notably absent from the aforementioned question answering code. In order to perform an aggregation across different users' personal data stores, a second question answering format is specified, along with a number of endpoints within the PDS to facilitate group computation across a distributed set of machines.

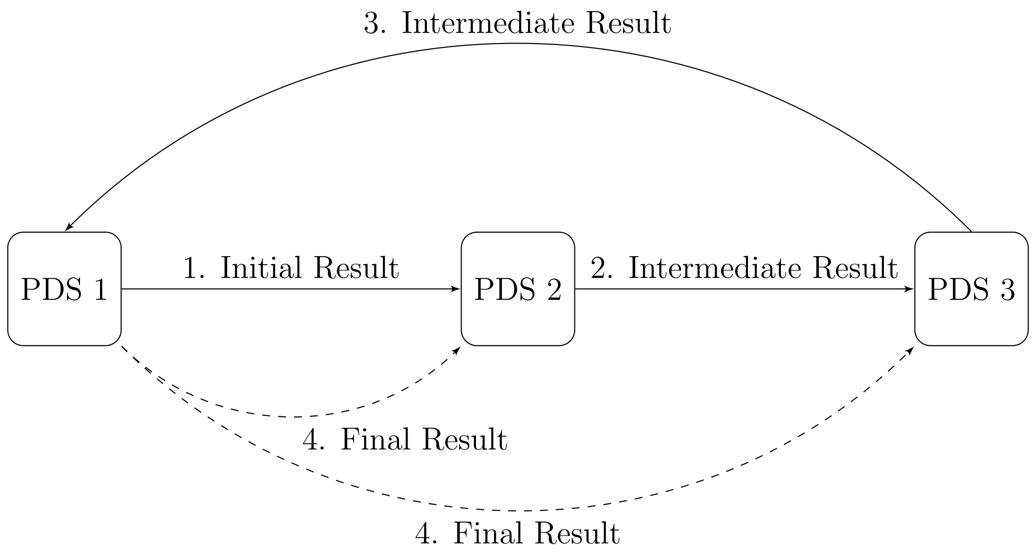


Figure 3-4: Group computation flow: 1) Initiating PDS begins aggregate computation 2-3) PDSes contribute to intermediate results passed along the chain of contributers 4) The initiating PDS completes the computation and POSTs the result back to all contributing data stores

As with the single-user case, questions that contribute to group answers are asynchronous celery tasks. However, instead of being run within the PDS continuously on a set schedule, these tasks are invoked via a group contribution URL on the user's PDS, located at `/aggregate/contribute`. Personal data stores POST to this endpoint in a ring fashion to build up a running total for the aggregate computation. This endpoint takes the UUID of the user contributing to the computation as a querystring parameter, and the body of the POST request contains both the current intermediate result of the running computation as well as the remaining list of personal data stores to visit in order to complete the computation. Each personal data store updates the running total, taking into account data within that PDS, and then posts the updated intermediate result to the next personal data store in the chain. Figure 3-4 describes the flow of data amongst a small set of contributing personal data stores.

To take the running total into account when contributing to a group computation, the format for such questions modifies the original question format to take the current running total in addition to an InternalDataStore. Below is an example of such a question contribution that leverages the probe counts answer calculated by the previous example question:

Listing 3.3: Example question for contributing to an aggregate computation.

```
@task()
def contributeToAverageProbeCounts(ids, runningTotal):
    myCounts = ids.getAnswer("RecentProbeCounts")
    count = runningTotal["count"]
    for probe in myCounts:
        runningTotal[probe] = runningTotal[probe]*count
        runningTotal[probe] += myCounts[probe]
        runningTotal[probe] = runningTotal[probe]/(count+1)
    runningTotal["count"] += 1
    return runningTotal
```

After calling the method associated with a given aggregate question, the PDS takes the return value and POSTs it, along with the list of remaining contributers to the next PDS in the chain. When the running computation reaches the initiating

PDS, a separate completion method is called that pushes the final result out to each contributing PDS, where a copy of the result is stored locally.

3.4.6 Aggregate Performance and Approximations

Aggregate computations incur a latency cost over traditional methods that would pull raw data for all contributing parties to a centralized place and perform computation therein. For a small sample population of hundreds of users, this latency can be acceptable - the set of users a registry server returns for a given computation can be ordered in a way that clusters personal data stores in similar geographic locations as a best-effort means of minimizing latency of the total computation. Assuming average latency between data stores can be kept below 500ms, every 100 separate data stores will add at most 50 seconds of latency to the computation. This is deemed acceptable for the typical group computation workload, where the value of the aggregate changes smoothly and slowly over time and thus, computing the aggregate in an offline manner and storing the result is acceptable with minimal loss of accuracy over time.

However, in the presence of even thousands of separate personal data stores, latency can begin to affect the accuracy of the computation. Table 3.3 shows computation times and overhead for a typical group aggregate that computes the average and standard deviation of a given answer as the number of personal data stores grows. These numbers were taken from a simulated workload between 2 personal data stores with identical system configurations (1.6 GHZ dual-core processor, 2GB RAM), and 40ms of network latency between them, as measured by `ping`.

For the purposes of analysis, a result that takes less than 5 seconds to compute will be considered feasible for real-time computation at the time of the request, and any result that takes less than 5 minutes to compute will be considered “real-time” if the pre-computed result is stored in a manner that can be quickly retrieved. This method of retrieving an up-to-date, but pre-computed result at the time of the request for an answer is known as a “real-time batch”, where the result is returned in real-time (the time it takes to look up an answer in a PDS is less than 100ms), and the data has been calculated based on the most up-to-date raw data available. As the number

Data stores	Computation Time (s)	Overhead (s)	Total Time (s)
1	0.011	0	0.011
2	0.022	0.27	0.292
5	0.05	1.43	1.48
10	0.16	2.66	2.82
100	1.64	28.4	30.2
1000	15.1	302.27	317.37
10000	156.2	3166.215	3316.415

Table 3.3: Aggregate performance and overhead - Overhead quickly overtakes computation time when computing an aggregate.

of data stores contributing to a computation grows, time to complete one rotation around all contributing data stores makes real-time results intractable for anything more than 10 contributing data stores. Furthermore, when 1000 or more data stores contribute to a result, even maintaining a real-time batch result becomes unfeasible.

To circumvent this limitation, personal data stores have the ability to short-circuit aggregate computation by returning an intermediate result to the data store that initiated the computation, allowing it to push out the partial result of the computation to contributing data stores prior to completing the computation. The number of data stores to visit prior to storing an intermediate result is configurable on a per-question basis. For the provided workload, a good time to store an intermediate result might be after visiting 100 data stores. Visiting the contributing personal data stores in an unbiased order allows the system to store the intermediate result of the 5-minute long computation in 30 seconds with 10% confidence. For each 100 contributions after the approximate result is computed, the resulting approximate answer is updated with an additional 10% confidence, until the computation is complete and the result represents the true value.

The ability to return approximate results means that each contribution to a running calculation must be representative of the final result - if the final result of an aggregate computation depends on post-processing within the initiating data store, prior to pushing to the contributing stores, this post-processing must apply to an intermediate (approximate) result as well. To facilitate such post-processing, the personal data store keeps track of the number of contributers along with the run-

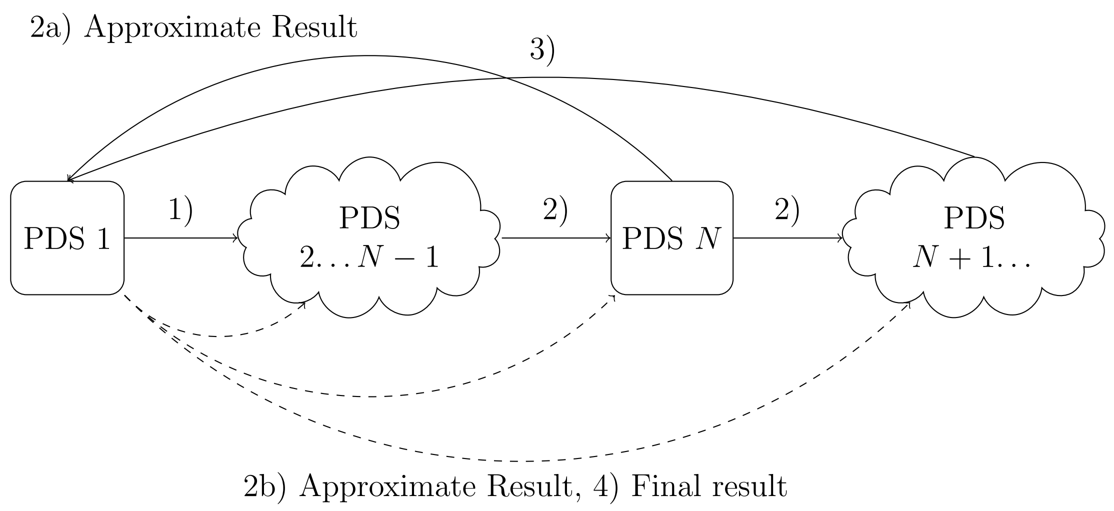


Figure 3-5: Group approximation flow: 1-2) PDS's contribute to intermediate results passed along the chain of contributers 2a) After N contributions have been made, an approximate result is returned to the initiating PDS 2b) The initiating PDS pushes this result to all contributors 2-3) Aggregation continues, returning approximate answers ever N contributions, until 4) the final result is returned.

ning total passed between contributers. Under the approximate flow, contributers to the example calculation would receive an intermediate result approximately every 30 seconds with increasing confidence until the 5 minute computation has completed.

3.4.7 Clustered Hosting

Using the group approximation framework, there is the potential to further enhance performance by clustering a number of logical personal data stores within a single physical machine, and contributing to group computations on all clustered data stores at once. It is important to note that this is different than hosting all personal data stores in an ecosystem in the same place - in an ecosystem of thousands or millions of personal data stores, such an approach would provide a single point of attack and furthermore, is prone to leaking everyone's data in the event of a security breach. Rather, from the aggregate performance numbers above, a good initial guess at the number of personal data stores to host within a single machine would be 100, resulting in reasonable performance, and little to no increase in the likelihood of discerning which machine to attack for a given user's data.

For the case of a clustered personal data store that represents the data from a number of different users, the personal data store framework can re-order the list of contributers for a given aggregate computation to pull out all users that are hosted within the given physical machine, and perform a portion of the aggregate computation entirely within the machine before passing the intermediate result along to the next contributer. In this manner, figure 3-5 stands as an accurate representation of the flow, however the clouds of PDSes in the figure could represent either N physically separate personal data stores, or N clustered personal data stores within a single physical machine. Table 3.4 shows the results of the same computation as table 3.3, with clusters consisting of 100 logical personal data stores. While overhead is decreased by a factor of 100, computation sees a commensurate decrease, likely due to better resource utilization within the cluster (code can stay in memory, personal data can stay in cache).

Data stores	Computation Time (s)	Overhead (s)	Total Time (s)
1	0.01	0	0.01
10	0.1	0	0.16
100	0.55	0	0.55
200	3.22	0.27	3.49
1000	4.17	2.66	6.83
10000	40.37	28.4	68.77

Table 3.4: Clustered aggregate performance - Overhead is reduced by a factor of N for clusters of N personal data stores. Furthermore, computation time decreases as cache and memory within a cluster is better utilized.

As aggregation within a clustered personal data store occurs without even intermediate aggregate results leaving the store, clustered hosting also has the potential to further preserve the privacy of individuals participating in group aggregate computations; an individual's personal data store within a cluster becomes more resilient to attacks from outside the cluster that would attempt to utilize the group computation framework to re-identify the individual based on their contributions to group computations.

3.5 Client Applications

To facilitate data collection and user interaction, a Java client library is implemented on top of the Android platform. In addition to extending Funf data collection to incorporate access control and syncing both data and configuration with a personal data store, classes are provided to interface directly with both the registry and distributed data store components of the openPDS personal data ecosystem.

Applications must follow the authentication flow described in figure 3-2, which, from the client library perspective, involves the use of two library classes: RegistryClient and PersonalDataStore. These classes wrap corresponding interactions with the Registry Server for authentication and authorization, and the PersonalDataStore for retrieving data and visualizations. The RegistryClient and PersonalDataStore interfaces are provided below in tables 3.5 and 3.6.

class RegistryClient		
Method	Arguments	Description
RegistryClient	String url String clientKey String clientSecret String basicAuth	Constructs a RegistryClient connecting to the given url for the given client credentials.
authorize	String username String password	Attempts to authorize the client associated with this RegistryClient to access the given user's account.
getUserInfo	String token	Attempts to retrieve information, including PDS location and UUID for the user that issued the provided token.
createProfile	String email String password String firstName String lastName	Registers a new user with the given credentials on the Registry server.

Table 3.5: RegistryClient provides a means of interfacing with a Registry server from an Android client application.

A typical client application will first prompt the user to either register or login, and will then retrieve authorization and a link to the user's personal data store. To this end, the openPDS client library provides a number of convenience asynchronous tasks: `UserLoginTask`, `UserRegistrationTask`, and `UserInfoTask` that wrap each

class PersonalDataStore		
Method	Arguments	Description
PersonalDataStore	Context context	Initializes a PersonalDataStore given an Android Context with the requisite information (PDS location, user UUID, bearer token).
getAnswer	String key String password	Retrieves a JSONObject for the answer corresponding to the given key from the PDS if it exists and the client has access to it. Null otherwise.
getAnswerList	String key String password	Retrieves a JSONArray for the answer corresponding to the given key from the PDS if it exists and the client has access to it. Null otherwise.
registerGCMDevice	String regId	Registers the current device with the provided GCM registration ID on the PDS.
getNotifications		Retrieves all notifications on the PDS
buildAbsoluteUrl	String relativeUrl	Builds the absolute URL for the resource located at relativeUrl on this PDS.

Table 3.6: PersonalDataStore provides a means of interfacing with a user's Personal Data Store from an Android client application.

of the corresponding methods on RegistryClient, and update the application's SharedPreferences to store the necessary information used to construct a PersonalDataStore.

After the user has registered or logged in via the RegistryClient, and has authorized the client application to access their personal data store, all subsequent communication is done directly via PersonalDataStore objects (described in table 3.6). A client application can request answers from the PersonalDataStore object and present native user interface widgets, or request a fully-qualified URL from a PersonalDataStore object and display a Webview containing the corresponding page on the user's personal data store.

3.5.1 Configuring Data Collection

The openPDS client extends Funf data collection by providing an implementation of the standard Funf Pipeline interface named OpenPDSPipeline. This is used in place of Funf's BasicPipeline in the configuration file for Funf and, as with standard Funf configurations, the configuration is a JSON-formatted dictionary containing a `@type` field for specifying the fully qualified pipeline class, followed by a dictionary for declaring scheduled actions, and an array for declaring the data the pipeline instance will collect. An example configuration file for configuring data collection from call logs and location are provided below:

Listing 3.4: Example openPDS Funf pipeline definition.

```
{
  "@type": "edu.mit.media.openpds.client.funf.OpenPDSPipeline",
  "schedules": {
    "upload": {"strict": false, "interval": 900 }
  },
  "data": [
    {
      "@type": "edu.mit.media.funf.probe.builtin.CallLogProbe",
      "afterDate": 1365822705,
      "@schedule": {"strict": false,"interval": 3600 }
    },
    {
      "@type": "edu.mit.media.funf.probe.builtin.LocationProbe",
      "maxWaitTime": 30, "goodEnoughAccuracy": 10,
      "@schedule": {"strict": true,"interval": 900 }
    }
  ]
}
```

This configuration is provided as metadata within the service tag for the FunfManager service in the application's AndroidManifest.xml file. Additionally, permissions for each probe must be requested within this file as well, including fine and coarse grain location for any location probe, the requisite admin permissions for wifi and bluetooth scanning probes, and external storage and internet permissions for accessing the user's personal data store and writing data.

3.5.2 Defining Client Data Use and Visualizations

The full definition for a client application built on top of the openPDS ecosystem consists of three parts:

1. A client manifest file
2. A file containing all questions the app needs
3. A set of HTML visualization files

Each of the above requirements are combined into a zip archive to be installed on a user's personal datas tore. This section describes how each of these are defined and used by the openPDS client library and servers to create a cohesive client application.

Client Manifest

An client manifest file format is specified as a means of defining how a client application may present answers to questions about data, as well as what data contributes to a particular answer. Specifying a manifest for an app built on top of the openPDS ecosystem allows the client library to automatically generate user interfaces for notifying and providing control to the user over how their data is collected at the client level. The app manifest file is a JSON-formatted dictionary containing fields for the app name, client credentials, visualizations, and any answers the app will request access to, along with the data required to compute each answer. An example is provided below:

Listing 3.5: Example client manifest definition.

```
{
  'name': 'App Name',
  'credentials': {
    'key': 'registered_client_key',
    'secret': 'registered_client_secret',
    'auth': 'BASIC auth_string_here',
    'scopes': [ 'funf_write', 'answer1', 'answer2', 'answer3']
  },
  'visualizations' : [
    { 'title': 'Visualization 1 Title', 'key': 'vis1Key',
      'answers': ['answer1'] },
    { 'title': 'Visualization 2 Title', 'key': 'vis2Key',
      'answers': ['answer2', 'answer1'] }
  ],
  'answers': [
    { 'key': 'answer1',
      'data': ['answer3', {'key':'WifiProbe','required': false]}},
    { 'key': 'answer2',
      'data': ['ActivityProbe', 'LocationProbe'] },
    { 'key': 'answer3',
      'data': ['SmsProbe','CallLogProbe','BluetoothProbe'] }
  ]
}
```

The client library constructs access control screens for each answer based on the data it requires. For compound answers - those that rely on the result of another answer - the client library traverses the resulting data dependency tree to map back to the necessary probes required for computing the answer. Additionally, the client library provides a suite of user interface elements for displaying server-generated visualizations from the visualization keys provided in the client manifest. A well-defined client manifest file, combined with the pipeline definition metadata, allows a developer building on top of the personal data ecosystem to quickly develop applications with minimal client-side code, if they so choose, and have the client library automate the process of authorization, access control, and UI generation for visualizations.

Questions File

Each client must define any new questions they would like to assure will be populated on the user's personal data store. Individual questions are to be written as described in section 3.4.4, and included in a single `questions.py` file. If the client application requires questions from another application, the unique identifier for that question's key must be included in the client manifest file, but the definition of the question need not be included in the questions file - the given client will simply have a hard dependency on the related client being installed on the user's PDS.

Visualizations

As defined in the client manifest file, an openPDS client application can specify a number of visualization keys to automatically generate visualizations that have been approved and installed on a user's personal data store. These visualizations are available on the personal data store at `/visualization/key`, and take `bearer_token` and `datastore_owner` as querystring parameters.

As visualizations are generated on the user's personal data store, standard web languages are used to write them - HTML, Javascript, and CSS. To facilitate writing visualizations in web languages, the personal data store provides a set of Javascript libraries for interfacing with the endpoints described in 3.4.3. These includes answer and answerlist libraries, which provide Backbone.js collections and models for the corresponding objects in the external PDS API. Developers then write corresponding Backbone.js Views to pull in data from the provided collections and models and present the user with a visualization layer. Standard supporting libraries, such as jQuery and jQuery Mobile are provided by default on all visualization files, and externally-hosted Javascript and CSS files may also be imported.

Visualizations are built on top of Django's template language for HTML and consist of three template blocks a developer must fill in: `title`, `scripts`, and `content`. The personal data store uses these three sections to fill in a complete HTML page for the visualization. An example visualization file is provided below:

Listing 3.6: Example visualization file

```
{% block title %}
Recent Places
{% endblock %}
{% block scripts %}
```

```
<script src="//openlayers.org/dev/OpenLayers.mobile.js"></script>
<script src="{{ STATIC_URL }}js/answerList.js"></script>
<script src="{{ STATIC_URL }}js/answerListMap.js"></script>
<script>
$(function () {
    window.answerListMap = new AnswerListMap("RecentPlaces","
       answerListMapContainer");
    $(window).bind("orientationchange resize pageshow",
       answerListMap.updateSize);
    $("#plus").live('click', answerListMap.zoomIn);
    $("#minus").live('click', answerListMap.zoomOut);
});
</script>
{% endblock %}
{% block content %}
<div id="answerListMapContainer">
</div>
<div id="footer" data-role="footer">
</div>
{% endblock %}
```

In the above example, HTML is used only for structure on the page. Resulting information is filled in as a map using `answerListMap.js` - a pre-defined View provided on openPDS for displaying answers with latitude and longitude components on a map. When the key for this visualization is included in a client manifest, the client application will load a Webview with the corresponding fully-qualified URL to reach this visualization in the app, allowing an authorized app to display the visualization without supporting Java code.

Chapter 4

Designing Privacy-Preserving Apps

A personal data ecosystem designed with privacy-preservation as a first-order concept presents a number of unique challenges in app design. Namely, a number of traditional algorithms and design patterns must be rethought to operate within the constraints of user-centric personal data storage. This chapter details a number of example applications that seek to provide useful services built on top of personal data in a privacy-preserving manner using the techniques described in the previous chapter. In each case data has been collected continuously and passively via mobile phones and persisted to individuals' personal data stores.

4.1 Location-Based Apps

Social location based apps provide a useful service in allowing users to localize themselves with respect to their peers and the world around them. The raw location data these apps take into account when providing their services has been shown to be accurate in determining general activity patterns [37] and are also highly unique [14]. Due to the ability to re-identify location data with relative ease, various approaches have been taken to preserve the privacy of location data. These approaches include: quality degradation [17], false data injection [43], adding uncertainty [24], providing frequently changing pseudonyms [5], and enforcing k-anonymity via cloaking [28].

However, these approaches have proven inadequate for preserving privacy of the individuals' location data [32], [44]. Researchers [44] suggest that cloaking cannot preserve k-anonymity in a location dataset - if the k users were present in a small region, intersections among the cloaked regions of nearby users could be used to infer the locations of those users. Krumm [32] showed that, even in the presence of frequently changing pseudonyms, subjects' home addresses could be accurately determined and, using reverse geocoders, this approach allowed 13% of subjects to be re-identified by name. In each case, the presence of a unified set of raw location dataset for a multitude of users proves to be a liability in preserving the users' privacy.

4.1.1 Mining Frequently-Visited Locations

Using location data continuously collected every 15 minutes with a minimum accuracy of 100 meters, a record of an individual's most visited locations across time can be calculated. As this takes into account only a single individual's data, and provides the resulting set of most-visited places only to approved applications on behalf of the data owner, it is inherently privacy-preserving. Additionally, since this computation takes into account no aggregate data from other users, it can be written without using the personal data store's group computation framework.

Mining frequently visited locations can provide a semantic understanding of an individual's location data if further information about the individual's routine is taken into account. For example, knowledge of the times of day when an individual works would allow automatic tagging of the most frequent location during these hours as work. Similar analysis can be performed for other locations: home is where individuals most frequently sleep, the gym is where they are typically most active, etc. A visualization of a "My Places" application, installed on a user's PDS is displayed in figure 4-1.

Given a set of time ranges, a simple approach to finding frequented locations would be to look up all location traces during that time and cluster them according to their proximity to each other. For a set of time ranges that each represent the same time of day across different calendar days, such a clustering approach would provide the set of regions where the individual typically resides during that time of

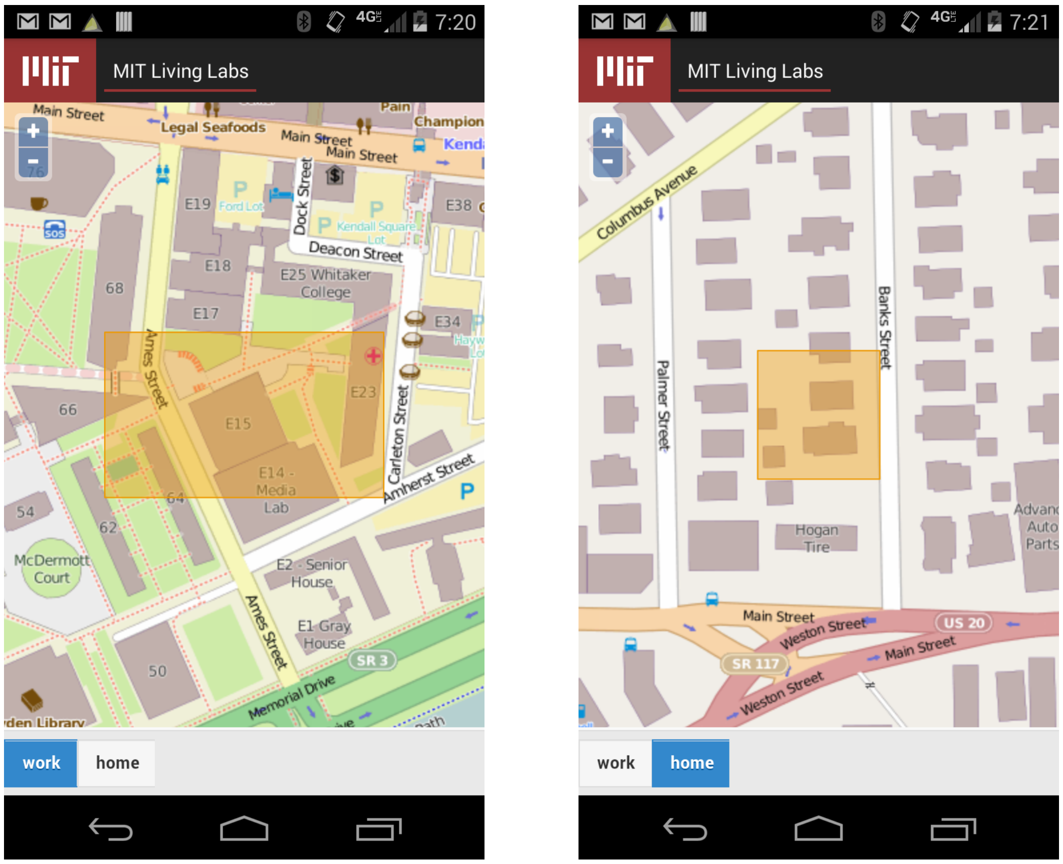


Figure 4-1: Home / work visualization - Home and work locations are accurately predicted with 2 days of location samples. A visualization is automatically generated on the client.

day, regardless of the day. These time ranges could be hours of the day; the set of times between 1pm and 2pm every day, for the last 2 weeks, or they could be a more semantic concept; "business hours", for example. For finding the most-visited region within a set of location points pulled from a given set of time ranges, a number of approaches could be taken:

1. Naive clustering - take all location samples from all time ranges, find clusters within them, choose the cluster with the most samples and compute its bounding box. This provides the most accurate regions, however, it can be very slow.

2. Naive clustering with binning pre-process - Apply a binning pre-process to location samples, collapsing multiple samples into single points with associated weights. Perform naive clustering and choose the cluster with the highest

weighted sum. This provides lower-resolution versions of the naive approaches' regions, and can trade accuracy for speed.

3. Multi-level clustering - apply naive clustering for each time range individually to get candidate regions. Loop over these candidate regions, collapsing and "voting" for those that overlap. Take the region with the most votes.

Of the above approaches, approach 1 provides the most accurate regions as it takes all location samples into account while clustering. However, as hierarchical clustering is very costly (O(n^3)) in computation time, finding the region can be prohibitively expensive in terms of processing time. Approach 2 is an improvement as it still takes all points into account while clustering and thus, essentially provides lower-resolution versions of the regions from approach 1, but provides a means of trading accuracy for speed. However, in the presence of very sparse location data, binning may not prove effective in decreasing the number of points to cluster and, as such, there is no guarantee that the process will run in a reasonable amount of time.

Approach 3 provides a low upper-bound on the time to perform the initial clustering; rather than depending on the total number of samples from all time ranges, it is dependent on the number of samples in an individual time range. For the case of a single, very long time range, this is not ideal and performance regresses to that of the naive approach. However, for the typical workload of discerning frequently visited locations for specific times of the day it is well-suited and fast; by definition, time ranges will always be less than 24 hours, and accuracy is gained by growing the number of time ranges, rather than the length of individual ranges. This decrease in runtime and increase in predictability of runtime for the algorithm comes at the expense of the quality of the resulting bounding boxes - some regions that should have been clustered according to standard hierarchical clustering may not count as the same cluster simply because they don't overlap. This shortcoming is deemed acceptable in this case - two regions that should have been clustered within 100m but do not overlap are likely to be considered the same region by a user, so selection between the two is arbitrary.

An implementation of the multi-level approach using the PDS internal API is as follows:

Listing 4.1: Multi-level location clustering algorithm

```
def findTopPlaces(ids, placeKey, timeRanges, numPlaces):
  regions = []
  for times in timeRanges:
    samples = ids.getData("LocationProbe", times[0], times[1])
    # Use all locations with at least 100m accuracy
    samples = [s for s in samples if s.accuracy < 100]
    # Hierarchical clustering of samples within 100 meters
    clusters = clusterFunfLocations(samples, 100)
    if (len(clusters) > 0):
      # Take locations from the biggest cluster
      clusterLocations = max(clusters, key=lambda c:len(c))
      regions.append(getBoundingBox(clusterLocations))

  if len(regions) > 0:
    overlaps = [{
      "region": r1,
      "overlaps": [r2 for r2 in regions if boundsOverlap(r1, r2)]}
      for r1 in regions]
    reduced = [{
      "region": reduce(lambda r1, r2: merge(r1,r2), r["overlaps"],
          r["region"]),
      "votes": len(r["overlaps"])}
      for r in overlaps]
    reduced.sort(key = lambda r: -r["votes"])
    mostOverlap = reduced[:min(len(reduced),numPlaces)]
    mostVoted = [r["region"] for r in mostOverlap]
    if numPlaces == 1:
      mostVoted = mostVoted[0]
    answer = ids.getAnswerList("RecentPlaces")
    answer = answer["value"] if answer is not None else []
    answer = [datum for datum in answer if datum["key"] !=
        placeKey]
    answer.append({ "key": placeKey, "bounds": mostVoted})
    ids.saveAnswer("RecentPlaces", answer)
```

With the above clustering algorithm, a good guess as to where an individual lives would select time ranges at night, when the user is likely to be at home and possibly asleep. Likewise, assuming average working hours of 9am to 5pm, a reasonable estimate of a user's work location could be computed by selecting time ranges between 10am and 4pm on weekdays to account for noise in the time the individual gets to work. The following question, structured in accordance with the question-answering specification in section 3.4.4, calls the above clustering method to provide one such answer on a schedule that keeps it continually up-to-date on a user's personal data store:

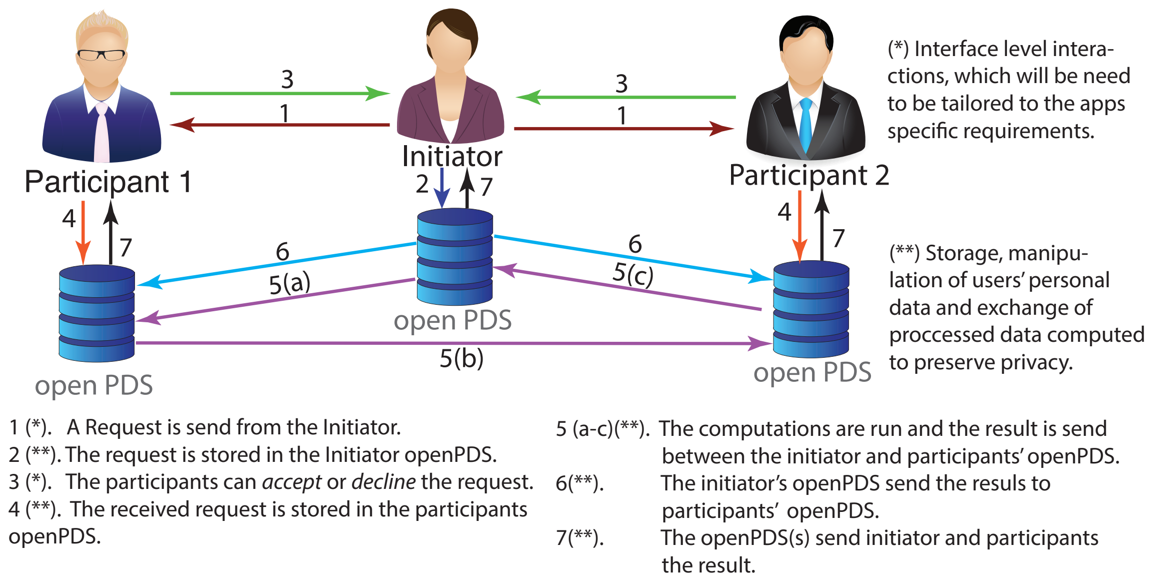
Listing 4.2: Question for finding an individual's work and home locations

```
@task()
def findRecentPlaces(internalDataStore):
  now = time.time()
  today = date.fromtimestamp(now)
  #Look at the last 2 weeks
  start = time.mktime((today - timedelta(days=14)).timetuple())
  #Find work location
  nineToFives = [(n, n + 3600*8) for n in range(start + 3600*9,
     now, 3600*24)]
  findTopPlaces("work", nineToFives, 1)
  #Find home location
  midnightToSixes = [(m, m + 3600*6) for m in range(start, now,
     3600* 24)]
  findTopPlaces(internalDataStore, "home", midnightToSixes, 1)
```

An individual with the "My Places" app installed on their personal data store would have the answer "RecentPlaces" populated as an answer on their personal data store's external API, and a provided visualization with a key of "places" would be available as well. The client library for an Android application would then be able to directly construct a user interface for an application to calculate and display work and home addresses for a user without explicit user interaction, as displayed in figure 4-1.

4.1.2 ScheduleME: Automated Meeting Scheduling

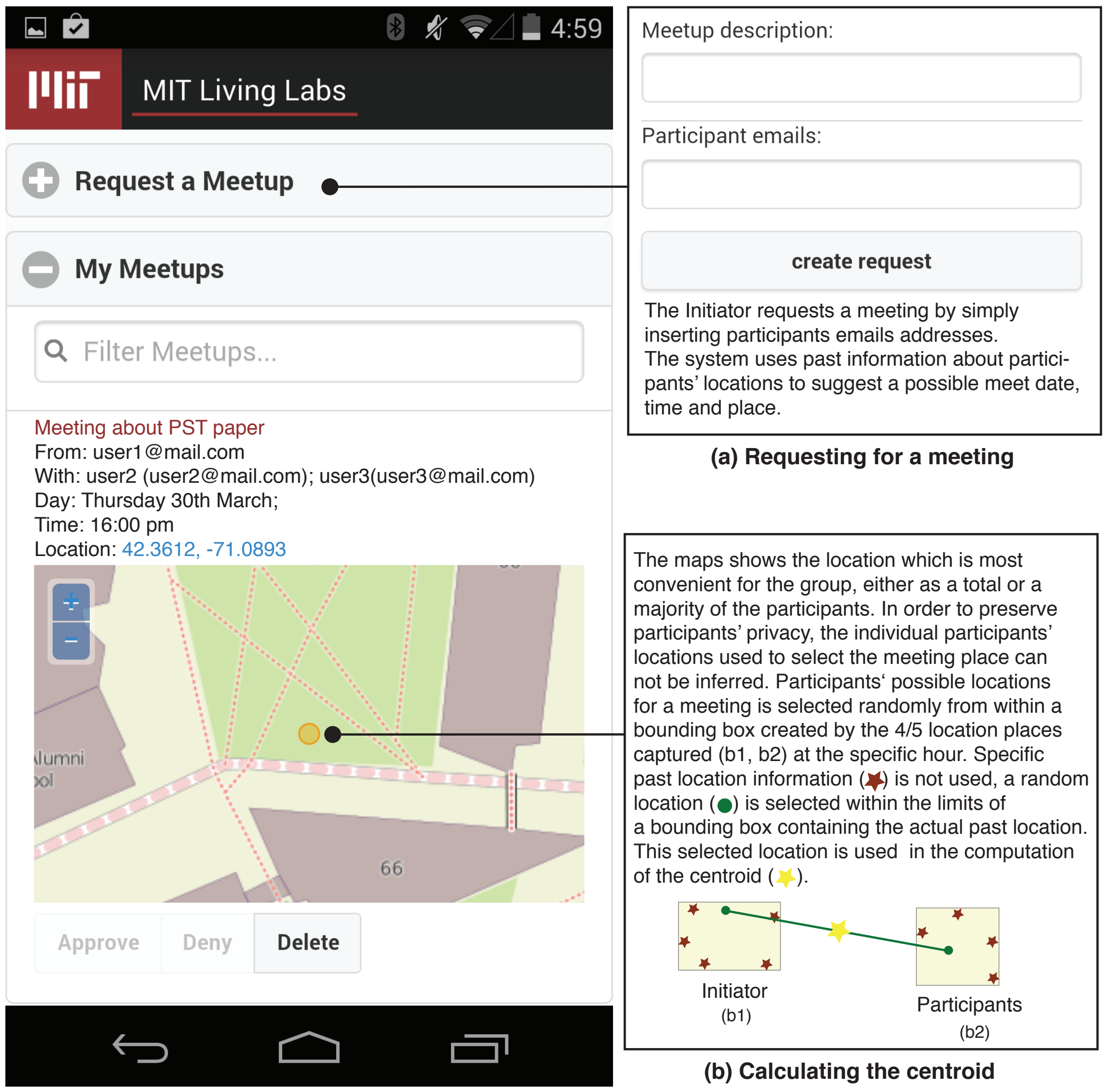
Assuming even loose adherence to a routine, a reasonable guess as to where an individual will be at any given time can be constructed by finding the place where the individual most frequently resides at the given time. Given the aforementioned question for discovering frequently visited regions, a set of time ranges can be constructed for each hour of the day over the last N weeks or months in order to find where the individual is likely to be for each hour of the day. Extending the "RecentPlaces" answer to provide a location key for each hour of the day makes reasonable real-time predictions about a user's whereabouts possible. This updated answer, providing guesses about an individual's location at any time of the day, can be provided in a group computation question to determine a reasonable time and place for a number of individuals to meet without exposing the raw data for each participant to the personal data stores engaging in the computation, or a centralized server [48].



(*) Interface level interactions, which will be need to be tailored to the apps specific requirements.
Participant 1
Initiator
Participant 2
3
1
3
1
2
7
4
7
4
7
6
5(a)
6
5(c)
5(b)
open PDS
open PDS
open PDS
(**) Storage, manipulation of users' personal data and exchange of proccessed data computed to preserve privacy.
1 (*). A Request is send from the Initiator.
2 (**). The request is stored in the Initiator openPDS.
3 (*). The participants can *accept* or *decline* the request.
4 (**). The received request is stored in the participants openPDS.
5 (a-c)(**). The computations are run and the result is send between the initiator and participants' openPDS.
6(**). The initiator's openPDS send the resuls to participants' openPDS.
7(**). The openPDS(s) send initiator and participants the result.

Figure 4-2: Modified group computation workflow with per-request approval [48]

In order to support aggregate computation for only users that have agreed to meet with a given user, the group computation framework detailed in 3.4.5 must be modified as shown in figure 4-2 to request consent of the user prior to participating in the group computation on a per-request basis.



Meeting about PST paper
From: user1@mail.com
With: user2 (user2@mail.com); user3(user3@mail.com)
Day: Thursday 30th March;
Time: 16:00 pm
Location: 42.3612, -71.0893

The Initiator requests a meeting by simply inserting participants emails addresses.
The system uses past information about participants' locations to suggest a possible meet date, time and place.

(a) Requesting for a meeting

The maps shows the location which is most convenient for the group, either as a total or a majority of the participants. In order to preserve participants' privacy, the individual participants' locations used to select the meeting place can not be inferred. Participants' possible locations for a meeting is selected randomly from within a bounding box created by the 4/5 location places captured (b1, b2) at the specific hour. Specific past location information (★) is not used, a random location (●) is selected within the limits of a bounding box containing the actual past location. This selected location is used in the computation of the centroid (★).

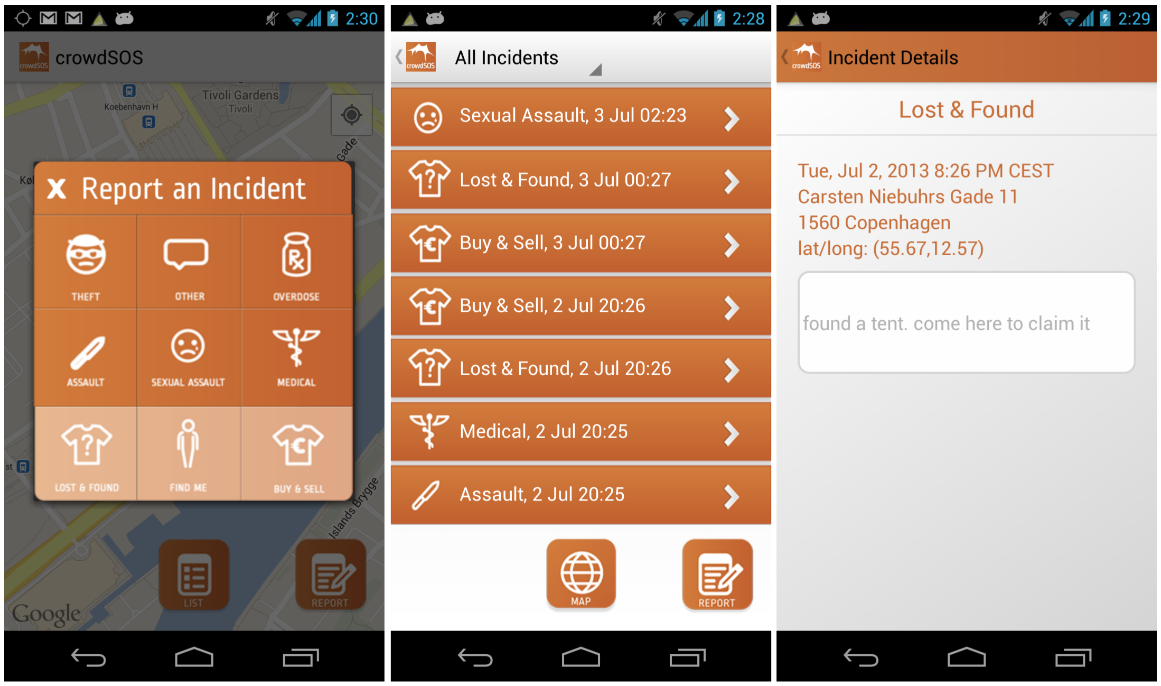
(b) Calculating the centroid

Figure 4-3: ScheduleME app: showing (a) the interface to request a meeting; (b) the possible results of a group computation, explaining how the actual personal location information is preserved and the computed answer is shared among users' PDS to maintain participants' privacy. [48]

Figure 4-3 shows a visualization to create, manage, and opt-in to requested meetings, as well as the process for automatic determination of a meeting time and location. A guess at a good meeting location for a number of participants is obtained by taking the centroid of their predicted locations for each hour of the day, then choosing the centroid (and corresponding time) that minimizes the total distance all participants must travel. The contribution step for each member of this group computation consists of first selecting a point at random from the hourly predicted regions and updating a running approximation of the centroid for all users at the given hour, along with a running sum of the distance the participant must travel to reach each of the current running centroids.

4.1.3 crowdSOS: Location-Based Crime Reporting

The crowdSOS Android application allows users to submit time and location-stamped incident reports for notifying contacts and displaying aggregate incident reports on a map. The application is built on top of Funf for passive data collection and openPDS for storage of both passively collected user data and incidents, as well as to aggregate incidents in a privacy-preserving manner. Each user within the crowdSOS app logs



Sexual Assault, 3 Jul 02:23
Lost & Found, 3 Jul 00:27
Buy & Sell, 3 Jul 00:27
Buy & Sell, 2 Jul 20:26
Lost & Found, 2 Jul 20:26
Medical, 2 Jul 20:25
Assault, 2 Jul 20:25

Lost & Found

Tue, Jul 2, 2013 8:26 PM CEST
Carsten Niebuhrs Gade 11
1560 Copenhagen
lat/long: (55.67,12.57)

found a tent. come here to claim it

Figure 4-4: crowdSOS crime reporting flow - users specify the type of incident, as well as a description. The incident is stamped with the current time and an approximate location, and a version with reduced spatio-temporal resolution is aggregated by users within 1km of the crime.

into the registry and receives a hosted personal data store for submitting crime reports to. Upon witnessing a crime, the user reports the crime via the flow in figure 4-4. This flow submits an incident entry to the user's personal data store, where it is stored in its raw form, with identifying information and raw location data. The user's personal data store then initiates a group computation as described in section 3.4.5 that provides a version of the incident with reduced spatio-temporal resolution and identifying information removed as the intermediate result. This intermediate result is thus POSTed in a chain to each personal data store returned by a query to the registry server.

Listing 4.3: Group contribution for storing an incident if it is relevant.

```
@task()
def storeIncidentIfRelevant(ids, incident):
  #Check for proximity to incident within the last day
  now = time.time()
  start = now - 24 * 3600
  locations = ids.getData("LocationProbe", start, None)
  nearby = ids.getAnswerList("NearbyIncidents")
  nearby = nearby if nearby is not None else []
  for location in locations:
    # Check if incident occurred within 1km of my recent locations
    if distanceBetween(location, incident.location) < 1000:
      nearby.append(incident)
      ids.saveAnswer("NearbyIncidents", nearby)
      return incident
  return incident #Unchanged, just to pass it along the chain
```

As each personal data store participating in this group computation has no knowledge of the origin for a given incident, the anonymity of the reporting party is preserved. Likewise, since each participating personal data store cannot see the result of the group contribution or the total list of nearby incidents for the others, privacy amongst the receiving parties is also preserved. Each personal data store contains only the corresponding list of incidents that are relevant for the given user and, as such, the personal data store is also the only machine the client application must interface with in order to present the map or list of incidents for the given user.

4.2 Quantified "Stuff": Self, Places, Relationships

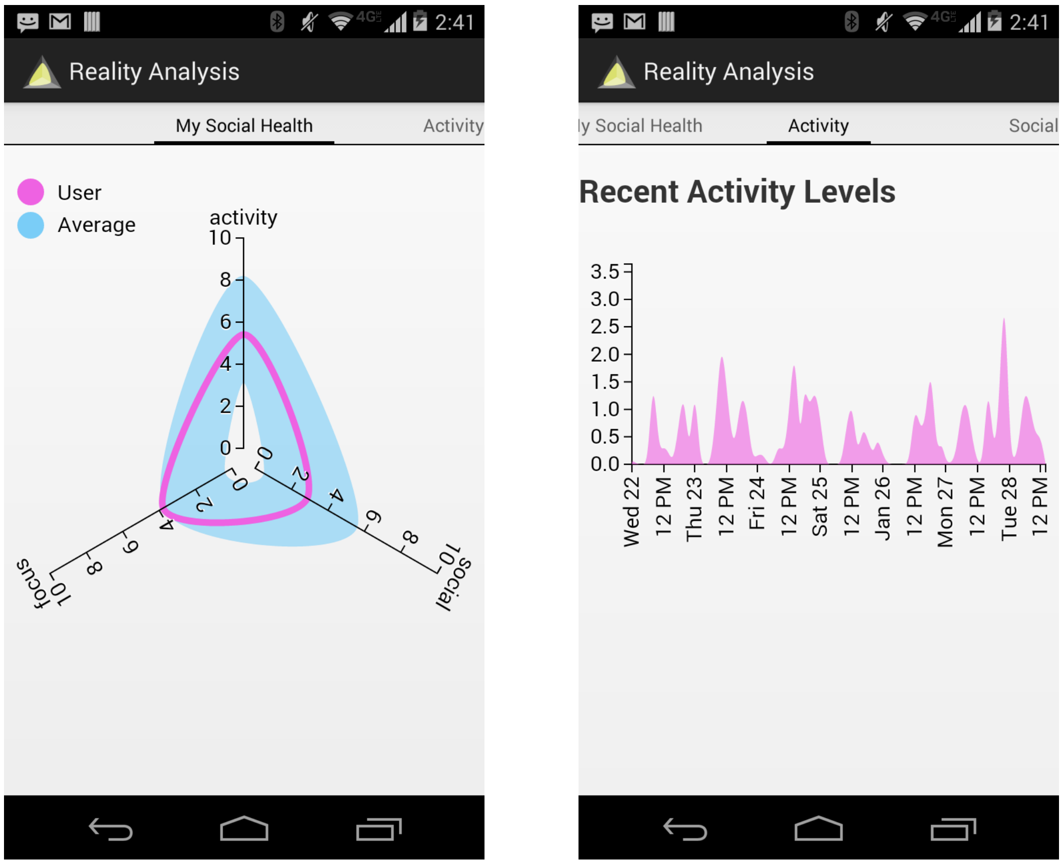


Figure 4-5: Social Health Tracker provides a view of 3 metrics: activity, focus, and social on a personal (pink) and average (blue) level. These metrics are calculated from sensor dat, aggregated in a privacy-preserving fashion, and presented as a current score as well as historical levels over time.

The Social Health Tracker application, shown in figure 4-5 is an application to construct 3 metrics of an individual's "social health": activity, social, and focus. Each of these metrics are computed based on input from a multitude of sensors that the app is configured to collect:

activity Uses the ActivityProbe in Funf - this is a compound probe that aggregates readings from the AccelerometerProbe for each second and classifies readings as having either no activity, low activity, or high activity. The app is configured to sample this probe for 15 seconds every 2 minutes.

social Uses CallLogProbe, SmsProbe, and BluetoothProbe as a means of measuring interactions via calls and text messages, as well as proximity to others via nearby Bluetooth devices. Bluetooth probes are taken once every 5 minutes.

focus Uses ScreenProbe and WifiProbe as a means of measuring regularity in the user's behavior. ScreenProbe records every screen on and off event on the phone, while WifiProbe is run every 5 minutes and records every wifi access point seen by the phone during a scan.

Both the US Center for Disease Control and Prevention and The American Heart Association recommend 150 minutes of moderate to vigorous exercise per week as a means of maintaining a healthy lifestyle [4, 22].This type of exercise includes brisk walking, which can be modeled using accelerometer data from a mobile phone, if it is carried on an individual for the majority of the day. Assuming 16 active hours per day, 150 minutes accounts for 2.2% of the week. As such, an initial guess for an activity score can be calculated by taking the percentage of ActivityProbe readings that have either low or high activity levels and applying a CDF with a mean of 2.2% and a standard deviation of 1% - an outlier in this case will be those individuals who get either less than 2 minutes or more than 40 minutes of exercise per day, on average.

A social score can be calculated by taking the weighted sum of all social interactions on the phone. This includes incoming and outgoing calls and texts, as well as unique devices seen in a Bluetooth proximity scan, as a proxy for face-to-face interaction. Summing the number of such social interactions and applying a similar CDF can yield an individual's social score. Likewise, taking the average number of screen-on events from a group of users, and applying a CDF to each individual user's ScreenProbe data can then yield a portion of the score for the focus metric. This portion of the focus score can be combined with a measure of regularity in the wifi access points the individual's phone senses, which is calculated by taking the set of access points sensed for each combination of hour of the day and day of the week over the past month and computing the Jaccard index of the resulting sets. The focus score will thus take into account both long-term adherence to a routine, as well as short-term attention.

Finally, the group aggregate for each score can be computed in a privacy-preserving manner by having individual personal data stores engage in a group computation to contribute their social health score results to the average. The high and low end of the average scores are computed as +/- 1 standard deviation from the mean, and are shown in figure 4-5 as the light blue band on the radial graph. The group contribution question for this aggregate is as follows:

Listing 4.4: Group contribution to average social health scores.

```
@task()
def contributeToAverageScores(ids, runningAvg):
    mine = ids.getAnswer("SocialHealth")
    count = runningAvg["count"]
    for metric in mine:
        runningAvg[metric] = runningAvg[metric]*count
        runningAvg[metric] += mine[metric]
        runningAvg[metric] = runningAvg[metric]/(count+1)
    runningAvg["count"] += 1
    return runningAvg
```

After computing the average using the above group computation, the initiating data store then runs the following aggregate to find the deviation. When this result is obtained, the answer is pushed out to all contributing data stores and the group high/low levels are stored.

Listing 4.5: Group contribution to social health score standard deviation.

```
@task()
def contributeToStdDev(ids, running):
    mine = ids.getAnswer("SocialHealth")
    runningDev = running["dev"]
    avg = running["average"]
    count = running["count"]
    for metric in mine:
        runningDev[metric] = runningDev[metric]**2*count
        runningDev[metric] += (mine[metric] - avg[metric])**2
        runningDev[metric] = sqrt(runningDev[metric]/(count+1))
    running["count"] += 1
    running["dev"] = runningDev
    return running
```

Quantifying one's own behavior is interesting in itself, but given the diversity of data located within a personal data store, including data about who an individual interacts with and where they spend their time, the opportunity to quantify more than just individuals presents itself. With a rich enough history, one could quantify relationships by observing the effect of interacting with someone on the rest of their sensor data ("I tend to be more active when I talk to Bob a lot"). Likewise, enough individuals contributing to an aggregate computation about a certain place or thing can help quantify the place or thing, in addition to the individuals contributing to the calculation ("Everyone seems to be more active when they talk to Bob a lot"). For example, a coffee shop might have a high incidence of visitors that like a certain type of music, and a library on a college campus will likely be quantified as the most focused, but least social and physically active building on campus.

4.3 Determining Uniqueness in Individuals' Data

"Unique in the Crowd" is an application intended to run as a coordinated demo amongst a large group of participants in the same location (a conference, for instance). The client is configured to passively collect wifi access point data every 5 minutes and persist the raw data to personal data stores provisioned for users at the start of the demo. As these data stores contain no data to begin with and recording begins at the same time for a group of individuals in the same location, no individual's data can uniquely identify them in the crowd of users running the application. The purpose of the application is to notify each of the participants when their data uniquely identifies them.

Determining uniqueness in a set of individuals' data is a more complex group computation than demonstrated thus far. In a standard aggregate computation, the result of a computation is intended to be distributed to all participants, and the initiating store plays the same role in the computation as the rest of the contributers. For the case of determining uniqueness, some or all of an individual's data must be compared to potentially everyone else's data as well. This requirement implies that the data store initiating the group computation plays a different role than the other contributors - they provide the seed data for one iteration of the computation.

Listing 4.6: Generating a "fingerprint" from wifi data.

```
@task()
def generateWifiPrint(ids):
  #Look at up to a week's worth of data
  start = getStartTime(7)
  now = getCurrentTime()
  wifiPrint = []
  for time in range(start, now, 3600):
    wifi = ids.getData("WifiProbe", time, time+3600)
    # Sort them by descending signal strength
    wifi = sorted(wifi, key=lambda x: -x["level"])
    #Record the top 10
    top = { "time": time, "wifi": wifi[:5] }
    wifiPrint.append(top)
  ids.saveAnswer("WifiPrint", wifiPrint)
```

Each data store continually updates a private answer containing the top 10 (in terms of signal strength) access points seen by the individual for each hour since the demo began (listing 4.6). This generates a sort of "fingerprint" for each user that is not unique at the start of the demo, and can be compared for uniqueness by the group computation framework. An initiating data store provides this list, along with a unique identifier, as the running result parameter of a group computation amongst all data stores. These contributing data stores each compare the provided set of timestamped access points to their own set internally. If the sets are different, the data is still unique. However, if the provided set is a subset of the contributor's set, the result is marked as not unique. Listing 4.7 provides an implementation of this group computation.

Listing 4.7: Group contribution to determine uniqueness.

```
@task()
def contributeToUniqueness(ids, intermediate):
  uuid = getStoredUUID()
  if intermediate["uuid"] == uuid:
    ids.saveAnswer("unique", intermediate["unique"])
    notifyCompletion()
  else:
    mine = ids.getAnswer("WifiPrint")
    theirs = intermediate["WifiPrint"]
    #If their's is contained in mine, it's not unique
    if len(theirs.intersection(mine)) == len(theirs):
      intermediate["unique"] = False
    return intermediate
```

Despite the computation being complete at this point, a link to the initiating store is intentionally not provided in order to reduce the risk of associating personal data stores with individuals. Upon receiving the result, the initiating data store notifies the registry server that their computation is complete, but does not reveal the result of the computation. The registry server then picks another data store at random from the set of remaining data stores to initiate the same group computation with their own data as the seed. Randomly selecting the order of data stores to run the uniqueness computation prevents a compromised contributing data store from correlating a fingerprint with the data store that provided it.

The above described serial computation around the data stores is admittedly much less efficient than a parallel approach, where each data store provides their fingerprint to every other data store at the same time. In fact, the parallel approach is $O(1)$ (with respect to the number of data stores) if all data stores perform the request at the same time, or $O(n)$ if a more practical approach (in terms of system resources) is taken whereby each data store performs the request one at a time. Compared to the $O(n^2)$ complexity of the aforementioned algorithm, this result may seem attractive. However, a compromised or malicious data store could potentially record the location of a given request for comparison and correlate fingerprints received from the same data store over time as a means of potentially re-identifying and linking a user to the given data store. For this reason, the less-efficient implementation is considered superior in cases where privacy is of utmost importance.

Chapter 5

Conclusions and Future Work

The personal data ecosystem described in the previous chapters shows promise as a means of giving control and ownership over personal data back to those who generated it. A move from the current world of service-centric storage and scope-based authorization towards a user-centric storage paradigm with a more semantic, purpose-based authorization model for data use represents a huge departure from the status quo, but also one that puts the individual at the forefront of their online life. In the presence of the ever-more ubiquitous sensor data collection prevalent in society, this level of control will become more and more essential for assuring the safety of sensitive data and the individuals it describes.

Research in the area of privacy-preserving data handling shows that such safety is hard-fought, and absolutely essential. An over-arching trend in this field of study leads to the inevitable conclusion that there may be no way to effectively "release" a raw data set that provides meaningful information about the individuals contained therein, without risking re-identification of the underlying individuals. As such, strong ownership, security, and access control are necessary but insufficient criteria for a framework that intends to support a large personal data ecosystem. In order to protect against risks that individuals and even researchers may not currently be aware of, a framework that truly hopes to protect the privacy of individuals' data within it must control all computation against the data as well.

While such a framework incurs some overhead in terms of computation time and resource usage, especially during aggregate computation, this overhead can mitigated in large part through approximate results and clustered hosting, to the point that a world with ubiquitous user-centric personal data stores becomes feasible in the near-term. Additionally, Moore's law is fighting squarely on the side of safer, albeit less efficient means of aggregating data - with each generation, hardware becomes exponentially faster and cheaper, allowing personal data ecosystems to "throw hardware at it" when dealing with inefficiencies. Indeed, the distributed and parallel nature of personal data stores allows arbitrary numbers of machines to engage in computation that can effectively be made more efficient by simply adding more machines with which to distribute the task.

Looking forward, openPDS and the resulting personal data ecosystem it powers has absolutely necessary steps to take in terms of scalability, adoption, and ease of development. To this end future work will seek to pick apart each component of the framework in the interest of discerning bottlenecks to scalability. Preliminary results from an outside corporation have identified the web framework of choice - Django within Python - as one potential bottleneck for both the registry server and data connectors on the personal data stores. An obvious next step is to provide an alternate implementation in a more vetted web language, such as Java.

Adoption must be spurred from both the industry / developer perspective, as well as the consumer perspective. To this end, making a turn-key personal data ecosystem that is more attractive to a corporation that building a competitor in-house will be a necessary step. This will involve improvements to the ease of deployment for the system, as well as the ease of development on the system. To this end, a full software development kit (SDK) should be developed that provides a means of writing questions against test data sets deploying resulting question and visualization / client bundles to a trusted "app store" for openPDS applications. Such a store would also ease deployment and adoption of the applications built on top of the framework.

Perhaps most importantly, a next and ongoing step is advocacy and education. Currently individuals are one of the biggest impediments improving the state of affairs in personal data management; studies cited within these chapters show little attention being paid to the use of sensitive personal data. However, this is not the fault of the individual - the onus must be on the party intending to use the data to properly inform individuals about the use of their data. The fact that this step is often ignored, or not given the proper thought and time it deserves is possibly the most important next step for the future of personal data as a whole.

Bibliography

[1] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing*, 7(6):643–659, 2011.

[2] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray, and Shaoyi Yin. Secure personal data servers: a vision paper. *Proceedings of the VLDB Endowment*, 3(1-2):25–35, 2010.

[3] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Iulian Sandu Popa, and Philippe Pucheral. Trusted cells: A sea change for personal data services. In *CIDR*, 2013.

[4] American Heart Association. American heart association recommendations for physical activity in adults, May 2014.

[5] Alastair R Beresford and Frank Stajano. Location privacy in pervasive computing. *Pervasive Computing, IEEE*, 2(1):46–55, 2003.

[6] Andrey Bogomolov, Bruno Lepri, and Fabio Pianesi. Happiness recognition from mobile phone data. In *Social Computing (SocialCom), 2013 International Conference on*, pages 790–795. IEEE, 2013.

[7] Theodore Book, Adam Pridgen, and Dan S Wallach. Longitudinal analysis of android ad library permissions. *arXiv preprint arXiv:1303.0857*, 2013.

[8] Vannevar Bush. As we may think. 1945.

[9] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[10] Lawrence H Cox. Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association*, 75(370):377–385, 1980.

[11] Tore Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15(429-444):2–1, 1977.

[12] Tore Dalenius. Finding a needle in a haystack-or identifying anonymous census record. *Journal of official statistics*, 2(3):329–336, 1986.

[13] Thomas Royle Dawber. *The Framingham Study: the epidemiology of atherosclerotic disease*, volume 84. Harvard university press Cambridge, MA:, 1980.

[14] Yves-Alexandre de Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3, 2013.

[15] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Pentland. openpds: Protecting the privacy of metadata through safeanswers. *PloS one*, 2014. (to appear).

[16] Yves-Alexandre de Montjoye, Samuel S Wang, Alex Pentland, Dinh Tien Tuan Anh, Anwitaman Datta, et al. On the trusted use of large-scale personal data. *IEEE Data Eng. Bull.*, 35(4):5–8, 2012.

[17] Matt Duckham and Lars Kulik. A formal model of obfuscation and negotiation for location privacy. In *Pervasive computing*, pages 152–170. Springer, 2005.

[18] Cynthia Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

[19] Nathan Eagle, Michael Macy, and Rob Claxton. Network diversity and economic development. *Science*, 328(5981):1029–1031, 2010.

[20] Douglas C Engelbart and William K English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 395–410. ACM, 1968.

[21] Ruth R Faden, Tom L Beauchamp, and NM King. A history and theory of informed consent. 1986.

[22] Center for Disease Control and Prevention. How much physical activity do adults need?, May 2014.

[23] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[24] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Show me how you move and i will tell you who you are. In *Proc. of ACM SIGSPATIAL Workshop on Security and Privacy in GIS and LBS*, pages 34–41. ACM, 2010.

[25] Johannes Gehrke. Models and methods for privacy-preserving data publishing and analysis. *Tutorial at*, 2006.

[26] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009.

[27] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[28] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42. ACM, 2003.

[29] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX security symposium*, pages 339–353. San Francisco, USA, 2002.

[30] Tom Kirkham, S Winfield, S Ravet, and S Kellomaki. A personal data store for an internet of subjects. In *Information Society (i-Society), 2011 International Conference on*, pages 92–97. IEEE, 2011.

[31] Tom Kirkham, Sandra Winfield, Serge Ravet, and Sampo Kellomaki. The personal data store approach to personal data security. *Security & Privacy, IEEE*, 11(5):12–19, 2013.

[32] John Krumm. Inference attacks on location tracks. In *Pervasive Computing*, pages 127–143. Springer, 2007.

[33] David Lazer, Alex Sandy Pentland, Lada Adamic, Sinan Aral, Albert Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009.

[34] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, volume 7, pages 106–115, 2007.

[35] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[36] Theodor H Nelson. Literary machines: The report on, and of, project xanadu concerning word processing, 1981.

[37] Anastasios Noulas, Salvatore Scellato, Cecilia Mascolo, and Massimiliano Pontil. An empirical study of geographic user activity patterns in foursquare. *ICWSM*, 11:70–573, 2011.

[38] Alex Pentland, Fabrizio Antonelli, Fabio Pianesi, and Nuria Oliver. Trentino mobile territorial lab, October 2012.

[39] Alex Pentland, Wen Dong, and Todd Reid. Reality commons, October 2012.

[40] Hugo Roy. Terms of service; didn't read, June 2012.

[41] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, Technical report, SRI International, 1998.

[42] Fuming Shih and Julia Boortz. Understanding peoples preferences for disclosing contextual information to smartphone apps. In *Human Aspects of Information Security, Privacy, and Trust*, pages 186–196. Springer, 2013.

[43] Reza Shokri, George Theodorakopoulos, George Danezis, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Quantifying location privacy: the case of sporadic location exposure. In *Privacy Enhancing Technologies*, pages 57–76. Springer, 2011.

[44] Reza Shokri, Carmela Troncoso, Claudia Diaz, Julien Freudiger, and Jean-Pierre Hubaux. Unraveling an old cloak: k-anonymity for location privacy. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 115–118. ACM, 2010.

[45] Vivek K Singh, Laura Freeman, Bruno Lepri, and Alex Sandy Pentland. Predicting spending behavior using socio-mobile features. In *Social Computing (SocialCom), 2013 International Conference on*, pages 174–179. IEEE, 2013.

[46] Gopal Sreenivasan. Does informed consent to research require comprehension? *Health Care Ethics in Canada*, page 335, 2011.

[47] Harry Sverdlove and Jon Cilley. Pausing google play: More than 100,000 android apps may pose security risks. Technical report, Bit9, Inc, 2012.

[48] Brian Sweatt, Sharon Paradesi, Ilaria Liccardi, Lalana Kagal, and Alex (Sandy) Pentland. Building privacy-preserving location-based apps. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*. IEEE, 2014. (to appear).

[49] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[50] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[51] Zhiqiang Yang, Sheng Zhong, and Rebecca N Wright. Anonymity-preserving data collection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 334–343. ACM, 2005.