Reducing Mobile Experimentation Risks with Sensibility Testbed: Balancing Data Acquisition with Privacy Protection

Yanyan Zhuang[†,‡] Albert Rafetseder[†] Jill Jermyn[*] Justin Cappos[†]
[†]New York University [‡]University of British Columbia [*]Columbia University

ABSTRACT

Due to their omnipresence, mobile devices such as smartphones and tablets could be of tremendous value to research. However, since research projects can use these devices to collect data that reveal personal information, there are substantial privacy concerns with mobile device use. Therefore, researchers must go through a detailed IRB process before recruiting participants. As such, many research studies either do not use data from real participants or use data collected from a tightly controlled and non-representative subset of participants.

In this paper we present Sensibility Testbed, an experiment platform that lowers the policy and technical barriers to performing experiments on end users' mobile phones. Data can be gathered through Sensibility Testbed without rendering mobile devices vulnerable to privacy invasion or potentially buggy experiment code. Access to sensor data on a mobile device is mediated according to the policies set by the institutional review board (IRB) of the researcher's institution. These policies are enforced by Sensibility Testbed via technical means, not just as stated policies, by a set of *blurring layers*. Different policies can be created by customizing individual blurring layers and loading them in order. This allows Sensibility Testbed to cater to a wider range of participants than prior mobile testbeds. Furthermore, Sensibility Testbed's platform design saves researchers' effort in recruiting participants and managing a deployment. Our evaluation shows that Sensibility Testbed enables useful experiments to be run while limiting the impact that malicious code could have on user privacy.

1. INTRODUCTION

End-user mobile devices, such as smartphones and tablets, have become indispensable gadgets in people's everyday life. Research has shown that nearly two-thirds of Americans own a smartphone, and 19% of them use the phone as their only means of staying connected [**?**]. For many people, these devices have become the dominant way they interact with others, and with the physical world.

Given the sheer numbers of these devices and the increasing sophistication of their features, the value of smart devices as data collection vehicles for government, university, or corporate studies continues to grow. Since these devices have embedded features like GPS, accelerometers, cameras, and microphones, they can generate valuable data for such studies as determining noise levels within an urban neighborhood, detecting approaching earthquakes, or studying traffic patterns at particular intersections. Accessing devices in a home network can also help providers improve the types of services they offer [**?**]. Testing applications on remote devices allows developers to better understand how applications perform in diverse environments, enabling improvements in performance [**?**]. For instance, some platform APIs change their behavior depending on the battery level of the device [**?**]. Without being able to remotely access battery life, these APIs can not guarantee basic function efficiency.

There have been initiatives within the network community to study mobile devices(e.g., Mobilyzer [**?**]), and in the systems community to deploy new services and test research prototypes (e.g., Phonelab [**?**, **?**]). However, personal devices remain largely underused because of two interrelated challenges:

- The risk research studies pose to the privacy of **device owners** and the performance of applications, and
- The difficulties **researchers** have securing access to devices, and using any data gathered in a responsible and ethical manner.

For device owners, privacy and security threats to mobile devices have increased dramatically over the years as potential attackers seek to take advantage of the rich functionality that sensors[1] on mobile devices can provide. Data acquired through a smartphone's GPS,WiFi connections, or Bluetooth pairing history can be highly personal, exposing sensitive information, such as where a person lives or shops [**?**]. Even seemingly benign applications, such as popular online games downloaded to mobile devices, can leak data, such as the model number of the device, or the age, gender, or location of its owners. [Lois: the citation was deleted somewhere along the lines. Its Angry Birds, I think] Furthermore, running experiment code poses a risk to the operation of the device itself, through potential exposure to bugs and other vulnerabilities.

For potential testbed users, the challenges are equally formidable. These experimenters are under the governance of Institutional Review Boards (IRBs) [**?**] that review all experimental protocols involving human subjects, and set a strict set of procedures that any researcher working under the aegis of the institution is required to follow. These include careful

[1] In this work, we broadly define sensors as the hardware components that can record phenomena about the physical world, such as the WiFi/cellular network, GPS location, movement acceleration, etc.

control over the collection and storage of data to ensure the privacy of subjects is preserved. It falls on the researcher to enforce these policies, and the process must be repeated every time he or she starts a new project. The experimenter must also recruit device owners willing to volunteer their devices for testing, a process that is time-consuming. Moreover, experimental setup and results cannot easily be shared with other researchers. As a result, each research group has to repeat the process of infrastructure setup, and policy implementation for each experimental deployment.

To address some of these concerns, experimental testbeds such as PhoneLab, have been established to provide a platform for running apps on smartphones. PhoneLab recruits participants by giving them free smartphones and reduced data plans in exchange for their commitment to use the phones as their personal devices. Other testbeds deal with both the recruitment and privacy issues by choosing to select participants from an internal group, such as faculty working with their students and colleagues [**?**, **?**, **?**]. However, such controlled candidate selections fall short for a few reasons. First, it does not relieve the burden on the researchers to ensure the privacy of the device owner and to enforce IRB policies, since the present network testbeds do not yet have a systematic way to provide these protections. As a result, there is limited protection for the device owners. Research has shown that most people usually do not understand the basics of privacy, or the implication of granting device permissions [**?**]. Therefore, if the testbed does not safeguard the security of devices, no matter what advantages participants may receive, their devices are still at risk.

In this work, we introduce Sensibility Testbed [**?**, **?**], an Internet-wide mobile testbed that lowers the technical barriers to research on personal mobile devices without lowering the ethical standards of the research institution [**?**]. The new testbed makes experiment prototyping faster, the remote control and management of devices easier, and the running of experiment code more secure in a number of ways. First, it provides better protection against invasion of privacy by carefully controlling access to device sensors. The testbed employs a stringent set of policies as to which sensors can be accessed, and these policies are customizable to each researchers' IRB policies. Therefore, this serves as a template for researchers to parameterize their experiment. The testbed infrastructure automatically implements the IRB policies on end-user devices, through the use of *blurring layers* in a secure sandbox. Each blurring layer mediates the access to a sensor by limiting the precision of data generated by it, and regulating the frequency that the sensor can be accessed. In addition, the testbed's secure sandbox provides both security and performance isolation, and ensures experiment code can not harm the devices of volunteers. Due to these privacy and security mechanisms, the enrollment process for volunteer device owners is as simple as a one-time download and install of an app. The testbed thus builds and maintains a pool of willing volunteers for researchers to choose from, eliminating the time-consuming process of recruitment.

The contributions of this work are as follows:

1. We identify the issues that have prevented successful implementation of experiments on remote user devices, including potential damage to devices from experiment code, the risk of privacy invasion, and the administrative challenges faced potential researchers. [Yanyan: I fell this may not be significant enough as a contribution.]

2. We introduce Sensibility Testbed as a flexible and secure platform for conducting experiments on Android devices that addresses many of the concerns above, including automatically obtaining and enforcing institution-mandated IRB policies.

3. We describe the unique features of Sensibility Testbed, which include controlled sensor access through the development of blurring layers.

4. We evaluate Sensibility Testbed's effectiveness in protecting device owners' privacy and find that nearly 86% of security and privacy issues present in previous projects using mobile devices were addressed by the new system.

The rest of this paper is organized as follows. First, in Section **??** we present background information about several key concepts that are critical to understanding how Sensibility Testbed works and how its use can benefit both researchers and device owners. Section **??** offers an overview of the design principles that guided development of the testbed, including a description of its key components and a simplified look at the operation of the program. The architecture of the Sensibility Testbed is reviewed in Section **??**. Section **??** presents a detailed look at the implementation of Sensibility Testbed , while Section offers a detailed walkthough of the program in operation. Section Seven provides experimental results to prove the viability of Sensibility Testbed in enforcing privacy policies, while Section **??** examines challenges and current limitations for implementation of the testbed. In Section **??** we review related work in protecting the privacy of data on mobile devices, and we share some concluding thoughts in Section **??**.

2. MOTIVATION AND BACKGROUND

In this section, we present some background information on factors that shaped the design and deployment of Sensibility Testbed. First, we offer the motivations behind its development. Next, we look at one solution to the problem: collecting proximate data in place of real data. Third, we talk about the traditional role of IRB in research involving human subjects and to data accessed remotely. Lastly, we look at Sensibility Testbed's default policies, which automatically closes off access to sensors which have the highest risk for violating the privacy of device owners.

Motivation: the potential and the risks of accessing sensor data. Having access to data from the enormous number of smartphones in use today could be tremendously valuable to researchers in a number of fields, ranging from health and fitness to the social sciences. It is also a research method that does not incur high maintenance costs. Unfortunately, it is a research method that is "expensive" in other ways, primarily time. Recruitment of volunteers is time-consuming and often frustrating. First, there is no easy method to build a pool of willing device owners, so it needs to be re-done every time a new experiment is proposed. Second, there are complications due to institutional policies on work with human subjects (discussed later in this section) that effects who can be recruited and under what conditions. An individual

recruiting on her own is likely to end up with a more homogenous set of subjects, drawn either from one locality or interest group. Last but not least, there is a very real threat of damage to a participant's device, and invasion of the participant's privacy. Participants face a significant risk if they offer their device to a researcher.

Solutions to the privacy and security problem have been difficult to identify, in part because the threat is not broadly recognized by device owners. Device owners may not fully understand the implications of granting access to a particular type of sensor or resource [**?**]. It is therefore challenging to conduct research on end-user devices in compliance with ethical standards, without an organized approach to enforcing those standards [**?**]. Another reason for the prevalent privacy breaches on many mobile systems is that only a subset of sensors like GPS and bluetooth are considered risky, and have their access mediated [**?**]. Other sensors such as accelerometer, gyroscope, etc., require no permission to access, thus leaving them open to attack. A first step to expanding the use of sensors on mobile devices is to design solutions that address the security needs of device owners and the practical needs of potential researchers.

Restricted data access as privacy protection. How specific does data need to be in order to be useful? Studies have indicated that sensor data can be accessed without compromising device owners' privacy or sacrificing service functioning if that data is generalized. Such data, that substitutes approximate location does not directly violate the privacy of the device owner, but still provides valuable information to the researcher conducting the study. For this reason, researchers have proposed substituting mocked [**?**] or anonymized [**?**] data in place of real data. For example, in location-based services such as maps, restaurant guides, and bus schedules, end users can still use the service even if a device only provides a discretized location [**?**, **?**]. The use of generalized data could also encourage more volunteer participation. Recent research shows that more than half of the surveyed individuals had no problem in supplying imprecise sensor data from their personal devices [**?**]. Most participants could accommodate some reduced application functionality, as long as their privacy was protected. Those applications included in the survey ranged from location-based search (e.g., Yelp), social network apps, to gaming and weather forecasting apps.

Based on these facts, restricting the amount of data accessible, such as reducing the precision or access frequency, offer an effective privacy protection mechanism to provide to end users. In this work, we coin the term *data blurring* as one privacy protection mechanism. However, to achieve data blurring easily and reliably to ensure experiments do not collect more data than needed to provide their functionalities requires a way to automatically substitute approximate data in place of explicit raw data.

IRB policies: guiding ethical behavior. As mentioned earlier, every researcher's work will be guided by an Institutional Review Board[2], an internal group that serves as the ethical watch dogs for colleges, universities, government agencies, and other research institutions. It is the job of these boards, to approve, monitor, and review research involving human subjects [**?**]. While a commitment to ethical treatment of humans who submit to experiments has always been part of the professional codes of most scientists, IRBs did not become ubiquitous in research facilities until the latter part of the 20th century. Partly provoked by atrocities committed by the Nazis in the name of scientific experiments in the Second World War II, and partly inspired by directives from the medical community, including Declaration of Helsinki established In 1964 by the World Medical Association, research institutions formally acknowledged the need to protect human subjects in any research setting. Today, IRBs require all researchers working under their aegis to submit the protocols of their studies in advance, with an aim to protect not only the physical and mental well-being of subjects, but also to protect any information about these individuals generated over the course of the study. However, enforcement of these policies becomes significantly harder when dealing with remote subjects. Although many current network testbeds require that researchers obtain IRB approval before conducting an experiment on the testbed, these platforms do not provide a guarantee for IRB policy compliance [**?**, **?**]. Therefore, there is no guarantee that an experiment will be compliant with a researcher's IRB policies.

Sensor	**Default policy**		
	LR	**MR**	**HR**
Battery (plug-in type, level, technology, etc.)	✓		
Bluetooth (local name, scan mode, etc.)		✓	
Cellular network (cell ID, area code, country code, operator name, etc.)		✓	
Location (latitude, longitude, altitude, speed, etc.)		✓	
Settings (screen brightness, ringer volume, etc.)		✓	
Motion sensors (accelerometer, gyroscope, magnetometer, orientation , etc.)		✓	
WiFi network (information about the currently active access point, and WiFi scan result)		✓	
Camera (take pictures, record videos)			✗
Intent (scan barcode, search, etc.)			✗
Address book			✗
Microphone (voice record)			✗
SMS (send/receive messages, delete messages)			✗

Table 1: Sensibility Testbed's default policies for sensors. LR/MR/HR stands for low/moderate/high risk, respectively. Access is only allowed to sensors that have low to moderate risks (marked by ✓). Sensors that are highly risky are disabled by default (marked by ✗).

Sensibility Testbed's default policies. [Yanyan: this may fit well with design principles.] The last unique concept Sensibility Testbed leveraged to ensure protection of end-user security and privacy protection is that all sensors are not alike. As mentioned earlier, failure to recognize the vulnerability of certain sensors was a key reason for privacy breaches. In designing Sensibility Testbed, default policies were set as to what types of sensors could be accessed. Even if an IRB happened to approve such a policy, there are certain sensors that the testbed's own IRB designates as off-limits due to the high risk associated with potential breaches. Only those sensors listed on our project wiki page [**?**] are accessible to a researcher. A summary of these sensors is listed in Table **??**, with each one categorized as low, moderate or high privacy risk. The list of sensors that Sensibility Testbed provides are all of moderate to low privacy risks (marked by ✓), and the testbed further provides policy enforcement (Section **??**) to protect all the sensor data. Sensors such as cameras and microphones that are deemed sensitive

[2] Also known as an independent ethics committee (IEC), ethical review board (ERB), or research ethics board (REB).

are not exposed to experiment code by default (marked by ✗). Such classification is motivated by the Android system, where permissions are categorized into different protection levels [**?**]: *normal* permissions are automatically granted to the apps, *dangerous* permissions are given based upon the user's consent, and so on. In our case, we divide sensors into different risk levels by the consequences and difficulties of a potential attack. If a microphone is controlled by a malicious party, it can be used to intelligently choose data of a higher value (e.g., credit card number, password) to record [**?**]. On the other hand, in order to infer a credit card number or password typed on a smartphone using motion sensors, the attack requires the installation of a sophisticated algorithm on the device that constantly learns about the patterns of data generated by accelerometer or gyroscope. In contrast, using battery information alone is not sufficient to create a fingerprint for each device. Different information and mutiple occurrences need to be pieced together to extract this data [**?**]. Therefore, compared to motion sensors, a microphone is considered a higher risk, and a battery is a significantly lower risk.

Although high-risk sensors are disabled, if such access is critical to the study, access can be requested using a different IRB procedure. In this case, the research project has to go through the Sensibility Testbed's IRB, in addition to the researcher's IRB. [Yanyan: if we think this is ok, then we provide specially designed interface and policy?] [Lois: following up on Yanyan's comment–If the Testbed's IRB says this expanded access is permissable, are the device owner's notified and can they opt out of this study? Otherwise, that would be a direct violation of the privacy protection you claim to give them] [Lois: I did not touch these last two paragraphs because I still don't know about the opt-out policy for individuals if this permission is given]

As a result, Sensibility Testbed does not provide unfettered access to all sensors. The default policies serve as a common denominator to all researchers' IRB policies. In most cases, we expect that researchers need only go through their local IRB to get the sensor access they need for their experiment.

3. OVERVIEW OF SENSIBILITY TESTBED

The design of Sensibility Testbed is guided by three design principles. In this section, we describe these principles and the resulting testbed components. We also show the testbed operation via two usage scenarios.

3.1 Design Principles

The operation of Sensibility Testbed involves three types of interacting parties, as shown in Figure **??**: mobile *devices* owned by ordinary people, with our app installed; a *clearinghouse* server that discovers and configures participating devices; and *experimenters* who want to run experiments on mobile devices. The interaction between these three components, as illustrated in Figure **??**, is designed to support three fundamental design principles.

Shared access to device resources. Since hardware resources are limited on mobile devices, it is not possible to dedicate an entire device to an experiment. Therefore, Sensibility Testbed allows several experiments to share one device. It uses a sandbox to isolate one researcher's code from another, through its performance isolation technique. The sandbox also prevents experiment code from inadvertently harming the device via security isolation. Other software within the device controls a researcher's access to a sandbox, allows the device owner to opt in or out of the testbed, and allows the device to communicate with the rest of the testbed. Together, they form the *device software* (Section **??**, though only the sandbox is shown in Figure **??**).

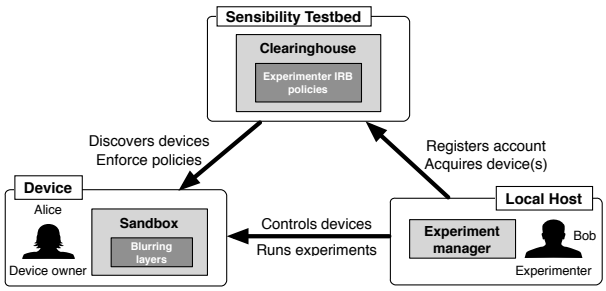


Figure 1: Sensibility Testbed architecture.

A centralized way of allocating devices and mediating data access. Sensibility Testbed is designed to provide both enhanced security and more efficient management of experimental setup, so another principle for the design was a central component that could serve as a a trusted intermediary for acquiring and managing device resources. Both tasks are delegated to a server called *clearinghouse* [**?**]. The clearinghouse allows researchers to register accounts for each experiment, and share access to a common pool of devices, freeing them from the need to individually recruit participants. The clearinghouse can also allocate resources and mediate data access according to policies defined by a researcher's IRB. The clearinghouse thus facilitates device sharing and policy enforcement (Section **??**).

Local support for remote experimentation. The last design principle was to allow researchers to manage their experiments on remote devices from a local machine, as is offered by other testbeds [**?**, **?**]. Sensibility Testbed addresses this need by using a tool called *experiment manager*. The experiment manager provides access to hardware resources on mobile devices using the researcher's testbed credentials (assigned by the clearinghouse). This tool will be introduced in Section **??**.

The following section describes how these parties interact to enable safe experimentation on mobile devices.

3.2 Testbed Operation

The basic operation of Sensibility Testbed involves two separate parties: a device owner interested in participating in experiments and a researcher seeking to run tests on remote devices. The device owner starts by installing the Sensibility Testbed app from the Android app store [**?**], which includes all the device software (Section **??**). After downloading the app, the device owner is informed about the testbed's general usage policy in a consent form and must give consent before participating. Any device owner, regardless of age, country, or background, can opt into our testbed in this manner, and can opt out just by uninstalling the app. The app will also display a list of experiments and each experiment's policies,

so the owner can opt out of individual experiments.

To conduct an experiment, a researcher will first provide his institution's IRB policies to the clearinghouse and will sign up for his experiment. [Yanyan: need a screenshot] The clearinghouse server helps him acquire devices, and codifies policies specified by the researcher's IRB. After obtaining remote devices, the researcher can perform experiments directly on the devices through the experiment manager, using the credentials assigned by the clearinghouse.

Two usage scenarios further illustrate how the clearinghouse protocol in operation, are given below. In this case, Alice, a device owner, participates in the testbed. A researcher, Bob, runs code on Sensibility Testbed using Alice's smartphone, discovered among other devices. Specifically, Bob wants to know the cellular service quality in major cities. As such, he needs location information of individual devices, their cellular service provider, network type (3G, 4G, LTE, etc.), and signal strength.

3.2.1 *Clearinghouse tracks devices*

Once the app is started, Alice's device can be discovered by the clearinghouse. The resource manager in her app periodically contacts a lookup service to advertise the device sandbox to the rest of the testbed, such as the clearinghouse or experiment manager. [Yanyan: resource manager and lookup service have not been introduced yet.] A lookup service is a distributed key-value store, such as a distributed hash table (DHT) [**?**], that allows one to associate keys with values and retrieve values associated with keys. In this case, Alice's resource manager uses an *identification key*, `key.alice`, generated during the app installation, and associates this key with Alice's device IP address and port number, by storing this key-value entry in the lookup service. This key is not associated with Alice or her device's identity, but only with the app's installation on the device. [Yanyan: I think the device first puts key.sensibility in the lookup service. is this key's value key.alice? so when the clearinghouse looks up key.sensibility, it finds out key.alice; and then it uses key.alice to lookup IP:port. But is it necessary to go into such detail?]

To keep track of Alice's device, the clearinghouse periodically queries the lookup service to discover any new device installs. Once Alice's device is discovered, the clearinghouse obtains `key.alice`, and thus can retrieve Alice's IP address and port number by looking up her key in the lookup service. This allows the clearinghouse to communicate with Alice's device over the network. If Alice uninstalls the Sensibility Testbed app, `key.alice` is deleted at the clearinghouse, which effectively unlinks her device from any metadata stored on the clearinghouse. The key will also be removed from the lookup service after a period of inactivity. An experiment manager controlled by a researcher can lookup available device installs in a similar way. Instead of finding all devices available to the testbed, the experiment manager will only be able to lookup devices assigned to the researcher by the clearinghouse.

3.2.2 *Researcher registers experiment and provides IRB policies*

Researcher Bob's first step to access the testbed begins when he fills out an experiment registration form at the clearinghouse. The clearinghouse registration page shows a list of sensors accessible to testbed users, and each of their possible accuracy and access frequency limits. In this particular case, Bob specifies that his experiment can (1) read location information from devices at the granularity of a city, (2) read accurate cellular signal strength and network type, as well as cell IDs, and (3) get location and cellular network data updates every 10 minutes. After filling out this form, Bob downloads the experiment description he provided, the detailed information about Sensibility Testbed and several relevant forms, such as those addressing consent, terms of participation (for device owners), terms of usage (for the researcher), and so on. [Yanyan: cite our docs] Bob then uses these forms as a template to complete the IRB application with his institution. These forms serve as a set of reference documents to make it easier for researchers like Bob to file the necessary IRB paperwork with their institutions.

After the application is submitted, Bob's IRB may disagree with his initial experiment requirements. For example, Bob's IRB will not permit his experiment to access cell IDs in cellular networks, but approves the other policies. In this case, Bob will revise the experiment registration form, refile the paperwork, and obtain IRB approval. Bob then submits the revised registration form and his IRB approval to the clearinghouse.

Finally, the clearinghouse parses Bob's registration form, extracts each data accuracy and access frequency limits approved by Bob's IRB policies (to blur an exact location to a city center, disallow access to cell IDs, and allow cellular network and location query once every 10 minutes), and assigns an experiment account to Bob. Once his account is activated, Bob obtains his public/private key pair, and can request a number of devices from the clearinghouse (Section **??**). The clearinghouse then looks up available devices like in Section **??**. If the clearinghouse discovers that Alice's device is available, it assigns her device to Bob's experiment account by placing Bob's public key in Alice's sandbox. At this point, Bob can access Alice's device through his experiment manager, just like using `ssh`.

As already described, the clearinghouse has a default set of blurring layers for accuracy and access frequency levels for each sensor. The clearinghouse first instructs the sandbox on Alice's device to add Bob's policies by preloading a set of blurring layer code. It then supplies the extracted data from Bob's registration form as input parameters to the blurring layers. When Bob runs his experiment, the policies are transparently applied to the experiment code. The policies are easily customizable. Details about how policies are implemented will be introduced in Section **??**.

4. SENSIBILITY TESTBED ARCHITECTURE

The architecture of Sensibility Testbed is comprised of three main components that are critical to its operation, as shown in Figure **??**. In this section, we discuss each of these components in greater detail, describing their functions and introducing several key techniques that make the testbed's enhanced security possible.

4.1 Device Software

The device software of Sensibility Testbed is the only one of the three components that runs on end users' devices. It

is divided into three parts: a secure sandbox called Repy (Restricted Python), a resource manager that facilitates the interaction between researchers and the sandbox, and a device manager that allows the device owner to control if the sandbox and resource manager can run on the device.

4.1.1 Secure Sandbox

The Repy sandbox is the most important in the device software. This security-reviewed sandbox [**?**] mitigates the impact of bugs in experimenter code by providing security isolation and performance isolation[3]. Researchers use a Python-like programming interface [**?**] to write experiment code, upload the code and execute it in the sandbox on remote devices. The programming interface includes functions for networking, file system access, threading, locking, logging, and so on. To access sensors, the sandbox also has a set of sensor functions [**?**]. Another important feature of the Repy sandbox allows us to change the behavior of its programming interface, and control the data gathered from the device to adapt to any IRB proscribed limits. For example, the sandbox can anonymize the IP address of a device, constrain the frequency of access to GPS locations, and disable access to cameras. The details of policy implementation are presented in Section **??** and **??**.

Security and performance isolation. The sandbox is the execution environment of all experiment code. It isolates experimenters' programs from one another, and prevents any of the code from harming the device through its strong isolation techniques. More importantly, the sandbox provides a flexible system call interposition technique where system call behavior can be modified. This allows us to define and implement different data access policies.

Repy has a small, self-contained kernel as its trusted computing base (TCB). Every system call in the TCB is strictly sanitized to preserve consistent behavior across different OSes, and to avoid uncommon corner cases that can be exploited. Furthermore, since the TCB is small (about 8,000 lines of code), it is less likely to have a kernel exploit than other more complex kernels. As a result, any vulnerability in an experiment cannot escape the sandbox and perform malicious actions to the device system. The Repy sandbox exposes its system calls for accessing resources on a device through a Python-like programming interface [**?**] isolating experiment code from the kernel. This same sandbox design has been successfully running experiments as part of a network research testbed for more than six years, without significant operation faults or security breaches [**?**]. [Yanyan: Justin thinks this statement is weak.] [Lois: I don't know if this answers his concerns since the meaning is essentially the same]

In addition to security isolation, the sandbox provides performance isolation by interposition on system calls [**?**] that use resources, such as network and disk I/O, and preventing or delaying the execution of these calls if they exceed their configured quota. This isolation means the sandbox can set access limits. For example, reading from and writing to the file system can only occur in a per-experiment directory, sending and receiving data via the network interface cannot exceed a configured rate, and CPU, memory and battery consumption cannot exceed a set level. Therefore,

[3]This sandbox has been used successfully for more than six years in our prior work, the Seattle testbed [**?**].

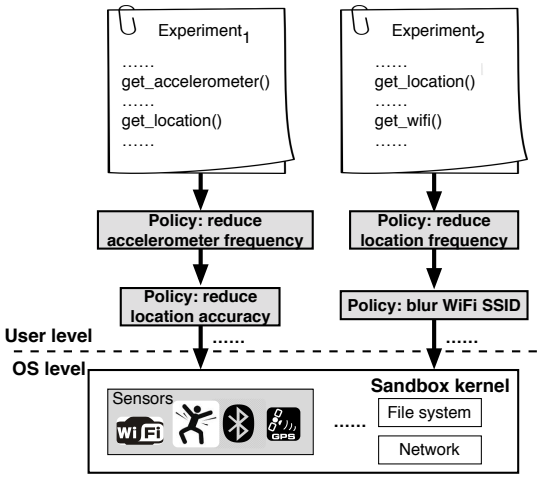


Figure 2: Policy stack demonstrating how Sensibility Testbed implements blur policies.

the sandbox isolates the experiment program from the rest of the device. Due to this isolation, different researchers can run experiments on different sandboxes on the same device, without any interference between the experiments.

Policy stack. In Sensibility Testbed, privacy policies are implemented as blurring layers, with each layer acting as a reference monitor in the sandbox [**?**] to enforce an access control policy over each sensor. Using the sandboxing technique in our prior work [**?**], we are able to interject code to control the behavior of sandbox functions, or system calls. A sensor access policy can (1) reduce the precision of the raw sensor data returned from a device, such as returning the location of a nearest city rather than the device's exact location, and (2) restrict the frequency of access to a sensor, such as the polling rate of a gyroscope or an accelerometer, to avoid password interference [**?**].

Each policy is a blurring layer, which defines either of the above two categories of policies. Different sets of policies can be customized according to the regulations set by IRB, through loading individual blurring layers in order, as a policy stack. In each stack, a lower layer is the ancestor of a higher layer. Every layer inherits the policy defined by its ancestor layers, with the exception of the lowest layer. The lowest blurring layer with no ancestors is the Sensibility Testbed's sandbox kernel. The experiment program runs at the top of the policy stack, thereby inheriting all the policies defined by the lower layers, as shown in Figure **??**. Each policy stack acts as a set of filters for different sensors, through which a call must pass before a sandboxed program can access the sensor data.

4.1.2 Resource Manager

The second part of the device software, called a resource manager, establishes a trust relationship between a device and a researcher. It manages a researcher's access to the sandboxes on a device, and controls the start and stop of the sandboxes on the device. The resource manager controls which researchers can access the sandbox on any given device, and communicates with the clearinghouse or experiment manager. If multiple researchers have access to the

same device, the resource manager splits the resources in the sandbox among the researchers, and partitions the one sandbox into several smaller ones. In order for the clearinghouse or a researcher's experiment manager to discover the sandbox on a device, the resource manager also contacts a lookup service that announces the existence of the available devices. An example is given in Section **??**.

4.1.3 Device Manager

The last part of the device software is a device manager that allows device owners to manage the software running on their devices. This is the only part of device software that a device owner will interact with directly. With the device manager, a device owner can install and remove the device software, and start and stop the operations. The device manager also provides a user interface to allow a device owner to opt out of any experiment. Thus, the device owner has the freedom to choose what experiments will be allowed to run on the phone or tablet.

4.2 Clearinghouse

The clearinghouse [**?**] is a testbed server that has two responsibilities. First, it keeps track of devices and grants researchers access to available devices; and second, it sets up the relevant IRB policies for each individual experiment that must be enforced through the sandboxes on remote devices. Researchers register their experiments at the clearinghouse, and provide their institution's IRB policies. When an account is approved, the researcher is assigned a pair of public/private *authentication keys* by the clearinghouse, to authenticate himself with the clearinghouse. This researcher can then sign into his account and request a number of sandboxes for his experiment. The clearinghouse looks up available sandboxes on behalf of the researcher by querying the lookup service. Once a sandbox is discovered, the clearinghouse stores the sandbox's meta information, and assigns it to the researcher's experiment account.

Besides assigning devices, the clearinghouse also has the role of instructing the sandboxes assigned to this researcher to add the IRB policies specified during registration. The clearinghouse does so by communicating with the resource managers on those devices, which control the code executed in the sandboxes. The involvement of the clearinghouse in any given experiment ends after the researcher deploys his code to the devices. It does not store any data on the researcher's behalf. This Sensibility Testbed clearinghouse protocol for research with device owners has been approved by the IRB at New York University (IRB # 15-10751). An example of this protocol in operation is given in Section **??**.

4.3 Experiment Manager

The last component in the testbed is an experiment manager, which a researcher can download to his own computer and use to run code in the sandboxes on the remote devices. The researcher uses the experiment manager as a light-weight command line console [**?**] to directly access the remote devices, upload experiment code written in the Repy programming interface, and communicate with the resource manager on the device to start or stop the execution of the experiment. To authenticate himself with the remote sandbox, the researcher uses his public/private key pairs to establish a secure connection from his computer. The experiment manager can also be used to download data from the remote devices to the researcher's local computer, or the researcher can set up his own server to store the data[4]. If an experimenter stores data on his own server, he must use protective measures to ensure that data sent from the mobile devices is properly encrypted, and the server storage cannot be tampered with by any other parties. Researchers can also opt to use a data store service we provide (a service called Sensevis [**?**], not shown in Figure **??**). After the data is collected, the method of securely storing the data will be mandated by the researcher's IRB.

This Sensibility Testbed clearinghouse protocol for research plays a central role in easing the process of device recruitment and experiment setup for experimenters, and ensures the enforcement of privacy policies. Using Sensibility Testbed, device owners' privacy is protected from any inadvertent or malicious attempt, and researchers are able to go through a streamlined process of device recruitment and experiment setup.

5. IMPLEMENTATION

In last section, we introduced the Repy sandbox briefly. In this section, we describe how to control the precision of sensor data (Section **??**) and the frequency of sensor access (Section **??**).

5.1 Policy Enforcement

5.1.1 Reducing Data Precision

Policies are implemented by each blurring layer via a *virtual namespace* that provides function mapping to substitute raw data access with restricted data access. To ensure the runtime behavior of each function mapping (for example, a function mapping of `get_location()` still returns location data), every blurring layer uses a *contract* to verify the interface semantics between the blurring layers.

Virtual namespace. The virtual namespace of each blurring layer executes the code in descendant layers with the corresponding layer's function mapping. By our convention, the virtual namespace of a blurring layer does not contain functions from the sandbox kernel or the namespace of its parent layer, unless explicitly specified by the mapping. [Yanyan: why is this statement needed?] For example, if a layer `foo` with functions `get_battery()`, `get_accelerometer()`, and `restricted_get_accelerometer()` were to instantiate a descendant layer `bar` with a function mapping

```
{
'get_battery': get_battery,
'get_accelerometer': restricted_get_accelerometer
}
```

then the module `bar` would have access to `foo.get_battery` via the name `get_battery`, and to `foo.restricted_get_accelerometer` via the name `get_accelerometer`.

[4] If an experimenter stores data on his own server, he must use protective measures to ensure that data sent from the mobile devices is properly encrypted, and the server storage cannot be tampered with by any other parties. This is enforced by requiring the experimenter to register his server by providing its certificate and URL to our clearinghouse. Following receipt of this data, the clearinghouse will instruct the devices accessible to the experimenter that all the sensor data collected should be sent to this server. The sandboxes on these devices will issue `HTTPS POST` using the server's certificate, and send encrypted data to the experimenter's server.

That is, the function on the left of the colon : *replaces* the function on the right via the virtual namespace. In this case, layer `bar` would not be able to access `foo.get_accelerometer`.

Contract. Each function that can be called by descendant blurring layers needs to have its arguments, return value, and other runtime behavior verified. This concept is similar to system call filtering mechanisms [**?**, **?**] that mediate access to a sensitive function interface. In our case, the functions are the calls to smartphone sensors that can potentially reveal a device owner's private information. Our verification process uses a contract. For each function that has a mapping, the contract lists the number and types of its arguments, the exceptions that can be raised by the function, and the return type of the function[5]

A contract is represented as a Python dictionary in Repy. As an example, if the blurring layer `foo` wanted to create a contract that would map `get_battery()` and `restricted_get_accelerometer()` into a new namespace, the contract would be:

```
{'get_battery': {
  'type': 'func',
  'args': None,
  'exceptions': (BatteryNotFoundError),
  'return': dict,
  'target': get_battery
  },
'get_accelerometer': {
  'type': 'func',
  'args': str,
  'exceptions': (ValueError),
  'return': list,
  'target': restricted_get_accelerometer
  }
}
```

Note that the symbols in the contract come from `foo`'s namespace. Thus the target for the `get_battery` in the contract is the `foo.get_battery` function. Similarly, the target for the `get_accelerometer` in the contract is the `foo.restricted_get_accelerometer`.

Whenever an experiment calls a function, such as `get_battery` or `get_accelerometer`, a verification process must perform type-checking using the contract. If the verification process detects a semantic violation, [Yanyan: add an example] the entire experiment program is terminated. This guarantees that to a sandboxed program, the implemented policies do not change a function's semantics.

Each blurring layer provides a contract to instantiate the next blurring layer by substituting a version of the function that enforces a given policy. All descendant layers (layers loaded after this layer) will have access to the version of the function that enforces this new policy. The final blurring layer starts the experiment program with the appropriate set of functions. Therefore, the experimenter's program is ensured to use all the new policies. An example of blurring layer implementation is given in [Yanyan: our TR.]

5.1.2 Restricting Data Access Frequency

Reducing data precision partially protects a device owner's privacy. However, frequent data access can be another channel to snoop on personal information. If sensor queries are made on a sufficiently frequent basis, they can be used to track an individual's activity. The goal of reducing sensor access frequency is to prevent an accumulation of identifiable information, such as location [**?**], wireless network identifiers, and even accelerometer data. (Note that specific details as to how frequently a certain sensor can be accessed without risking such an invasion is out of the scope of this work). Furthermore, the battery power of a device can be drained faster if sensors are accessed with unnecessary frequency. Therefore, restricting the data access rate mandates that experiments collect data only as often as is necessary. To achieve this, the Repy sandbox provides a second mechanism called `nanny` that mediates and restricts access frequency to sensors.

`nanny` treats all sensors as *resources*, and the function calls to access the sensors as the *usage* of resources. When an experiment program's use of a resource is above a given threshold, `nanny` pauses the program for as long as required to bound it, on average, below the threshold. Therefore, if an experiment program attempts to use a resource at a rate faster than is allowed by a policy, the function call is blocked until sufficient time has passed to average out the overall access rate. To monitor and control the usage of resources, `nanny` keeps a table of resource assignments that tracks and updates requests and releases. Once resource caps are set, an experiment program can never call a function to access a sensor more frequently than the cap.

An example implementation is given in [Yanyan: our TR.]

5.2 Resource Manager and Device Manager

[Yanyan: Albert, please add]

5.3 Clearinghouse

[Yanyan: Albert, please add]

5.4 Experiment Manager

[Yanyan: Albert, please add]

6. EXAMPLE SCENARIO

[Yanyan: may work better as a subsection of the previous section.]

From a researcher's specified IRB policies to running experiment code, the process goes as follows. The clearinghouse creates a list of access policies for a researcher's experiment, according to the specified IRB policies. The sandbox on a mobile device, under the control of this experimenter, obtains a list of command-line arguments from the clearinghouse, which includes all the blurring layers and parameters for each layer (Section **??**). The pre-set blurring layers determine the type of data required, such as accelerometer access rate, and the IRB parameters customize the specific policy, such as accessing an accelerometer at a rate of 50 times/second. The sandbox then instantiates the first blurring layer according to its contract, i.e., the function mapping that contains the kernel's exported functions. The newly instantiated blurring layer repeats this process to instantiate the next security layer with a potentially updated contract and function mapping. Eventually, the experimenter's program is instantiated in a separate layer with the functions provided through the stack of blurring layers that preceded it. The experimenter's program will then be subject to all the policies defined in the preceding layers, or the policy stack.

[5] Since Python is a dynamically-typed language, it is useful to type check a function's arguments, exceptions, and return values.

The mechanisms in this section are all transparent to the experimenters and device owners, as the implementation of policies is controlled by the clearinghouse on behalf of the experimenters. An experimenter is aware of certain policies in place, but does not need to implement or explicitly enforce such policies.

7. EVALUATION

In this section we evaluate the effectiveness of Sensibility Testbed's privacy mechanisms, its usability as a mobile testbed, and its performance. To investigate whether the policies provided by Sensibility Testbed are representative, we surveyed the projects in the past few years and identified their security and privacy mechanisms. We analyzed if Sensibility Testbed can provide policies to protect the corresponding data in Section **??**. We then show two examples by applying Sensibility Testbed's policies to two experiments in Section **??**. We demonstrate that with the policies in place, we can protect the device owner's privacy, while the data is sufficient for answering research questions. We show the performance overhead of Sensibility Testbed in Section **??**, and our deployment experience in Section **??**.

7.1 Sensibility Testbed's Privacy Policies

Privacy and security concerns. In order to identify the current privacy and security concerns on mobile devices, we surveyed 31 projects in the past few years. We then analyzed these projects by reading their corresponding publications, identified their needs or their provided mechanism for privacy protection. These projects, listed in Table **??** and **??**, range from social network applications [**?**] to facial recognition algorithms [**?**]. If their privacy need or privacy protection can be supported by equivalent *default* policies in Sensibility Testbed, a ✓ is marked in the default column. These sensors belong to low to moderate risk, as defined in Table **??**. Similarly, if the sensors are of high risk, a ✓ is marked in the specialized column. This means that a different IRB procedure could be followed to extend default policies (Section **??**). If a sensor cannot be supported by a default or specialized policy, a ✗ is marked in the N/A column.

Out of these projects, 19 of them proposed privacy protection about location information, 16 of them considered motion sensors such as accelerometer and gyroscope risky, and 9 had concerns about wireless network such as WiFi and Bluetooth (connection/pairing history, MAC addresses, etc.). As expected, location (mainly GPS) and motion sensors (mainly accelerometer) have the most privacy and security concerns [**?**].

Are Sensibility Testbed's policies sufficient? As shown in Table **??** and **??**, nearly 86% of the security and privacy issues in prior projects can be addressed in Sensibility Testbed using default policies, and 13% issues can be resolved by extending default policies (specialized policies). Only about 1% of them cannot be protected by the policies in Sensibility Testbed, [Yanyan: I'm not sure yet how to explain this. I don't know what experiment would need this.]

In particular, prior work shows that Android users' touch inputs can be revealed through a few attack techniques like keyloggers and fingerprinting. User generated data thus can be informative enough for malware to infer the key the user

Project	Sensor	Equivalent Sensibility Testbed policy		
		Default	Specialized	N/A
EnCore [**?**]	Bluetooth	✓		
[**?**]*	Accelerometer	✓		
	Camera		✓	
ProtectMyPrivacy [**?**]	Device ID		✓	
	WiFi	✓		
	Bluetooth	✓		
	Address book		✓	
	Location	✓		
CQue [**?**]	Location	✓		
	Accelerometer	✓		
	Gyroscope	✓		
	Bluetooth	✓		
MaskIt [**?**]	Location	✓		
	Cellular	✓		
	Bluetooth	✓		
Jigsaw [**?**]	Accelerometer	✓		
	Microphone		✓	
	Location	✓		
Caché [**?**]	Location	✓		
[**?**]*	Cellular	✓		
TapPrints [**?**]	Accelerometer	✓		
	Gyroscope	✓		
FindingMiMo [**?**]	WiFi	✓		
	Location	✓		
	Accelerometer	✓		
	Magnetometer	✓		
ACCessory [**?**]	Accelerometer	✓		
TouchLogger [**?**]	Gyroscope	✓		
MockDroid [**?**]	Location	✓		
	Internet†	✓		
	Cellular	✓		
	Address book		✓	
	Device ID		✓	
	Broadcast intent		✓	
Guardian [**?**]	Bluetooth	✓		
	Internet†	✓		
	Microphone		✓	
	Cellular	✓		
	Motion sensors	✓		
	CPU usage‡	✓		
[**?**]*	Accelerometer	✓		
[**?**]*	Accelerometer	✓		
	Gyroscope	✓		
AccelPrint [**?**]	Accelerometer	✓		
[**?**]*	Microphone		✓	
	Accelerometer	✓		
	Gyroscope	✓		
	Magnetometer	✓		
	Ambient light	✓		
	Location (GPS)	✓		
	Touch screen			✗
	Camera		✓	
Gyrophone [**?**]	Gyroscope	✓		
[**?**]*	Location	✓		
Where's Wally [**?**]	Location	✓		
AnonySense [**?**]	Location	✓		
	Network interface addresses	✓		
[**?**]*	Accelerometer	✓		
	Microphone		✓	
LP-Guardian [**?**]	Location	✓		
[**?**]*	Location	✓		
PrivStats [**?**]	Location	✓		
ipShield [**?**]	Location (GPS)	✓		
	Accelerometer	✓		
	Gyroscope	✓		
	WiFi	✓		
	Cellular	✓		

*These projects do not have a project name.

‡CPU usage can be obtained through reading the files in `/proc/stat`.

Table 2: Sensibility Testbed's support for privacy and security policies. A default policy is supported by Sensibility Testbed without extra effort. A specialized policy can be supported by extending default policies. A policy is marked as N/A if it is not possible to provide support.

Project	Sensor	Equivalent Sensibility Testbed policy		
		Default	Specialized	N/A
TaintDroid [?]	Location	✓		
	Accelerometer	✓		
	Internet†	✓		
Koi [?]	Location	✓		
Cloaking [?]	Location	✓		
Locaccino [?]	Location	✓		
Accomplice [?]	Location	✓		
	Accelerometer	✓		
TapLogger [?]	Gyroscope	✓		
	Accelerometer	✓		
Total	**77**	**65/77 (84.42%)**	**11/77 (14.29%)**	**1/76 (1.29%)**

†Internet connectivity policies can be implemented by interposing socket calls.

Table 3: Table **??** continued — Sensibility Testbed's support for privacy and security policies.

enters [**?**, **?**], or to fingerprint and identify individual devices [**?**, **?**]. Because these motion sensors are accessible without requesting any permissions or notifying the device owner, these attack techniques are much less detectable. However, the chance for these techniques to succeed depends on their sampling rate. Therefore, the policies to restrict the access rates to motion sensors are effective when the allowed rate is lower than the best keylogger would require to identify a key, or the best tracker to fingerprint a device. Section **??** shows such an example.

Another line of attack is on location privacy. [Yanyan: need to wirte more here. should we do an experiment for location?]

7.2 Privacy Protection and Experiment Functionality

[TODO: Q2: is ST useful for answering research questions? A: compare the results of an algorithm with varying levels of data precision (Seth's algorithm?), and show a diagram suggested by Justin.]

[Yanyan: with the privacy protection in place, is the data we provide sufficient for experiments to function?]

Research has shown that using sensors on a modern smartphone, such as a speaker [**?**], microphone [**?**] or accelerometer [**?**, **?**], one can build a robust device fingerprint that is independent of the software state and survives a hard reset. The more recent accelerometer based fingerprinting is particular interesting because an accelerometer is accessible in mobile apps without requesting any permissions from the device owner. Specifically, the technique in [**?**] is based on the device-specific errors in accelerometer calibration, which result in unique scaling and translation of the measured values. In this section, we show that by using data blurring in Sensibility Testbed, it is possible to prevent such fingerprinting, while producing data that is still sufficient for benign experiments to function.

Background on accelerometer calibration. An accelerometer measures the acceleration force to a device along three physical axes. However, during the manufacturing and assmebly process, imperfections or variations are introduced for each individual hardware instance, resulting in biases in the sampled data that are unique to a specific accelerometer [**?**]. The value of a measurement by an accelerometer, denoted by v_m, can be approximated as a linear funciton of its true value, denoted by v_t, as $v_m = v_t S + O$. S and O are the sensitivity and offset of the accelerometer, respectively. If an accelerometer has no calibration bias, then $S = 1$ and $O = 0$. Note that this linear bias is an approximation of the real bias. In practice, S and O manifest randomness.

Estimating biases along one axis. Like in [**?**], we take advantage of the Earth's gravity g, and analyze the biases along the Z-axis, S_z and O_z. To estimate two unknows, S_z and O_z, we need two measurements: an accelerometer's measurement value along the Z-axis when the device is facing up (z_m^+) and facing down (z_m^-). Therefore, we have

$$\begin{cases} S_z = (z_m^+ - z_m^-)/2g \\ O_z = (z_m^+ + z_m^-)/2 \end{cases} . \quad (1)$$

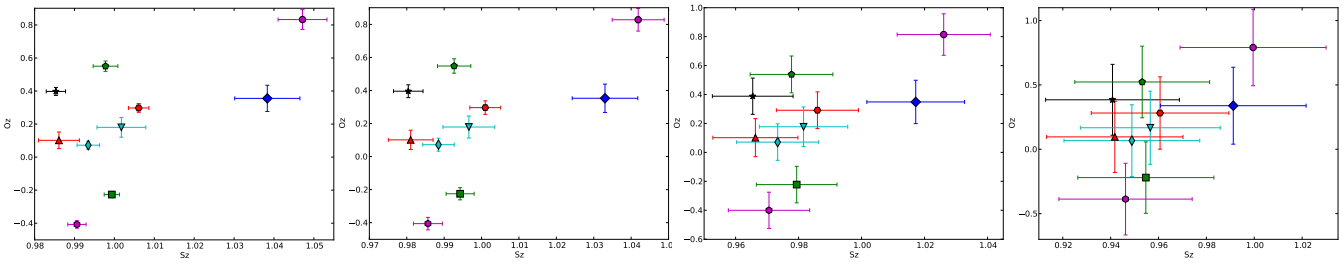
The authors of [**?**] showed that this method produces very satisfactory results in a range of settings even if the surface on which the phone lies is not perfectly level.

Data blurring and fingerprinting. We carried out an accelerometer fingerprinting experiment on a group of X Android devices, with $10,000$ measurement samples from each. Figure **??**(a) shows the estimated S_z and O_z based on unfiltered accelerometer data. The fingerprinting method is very effective. Among the X devices, only one pair is near-collision. In Figure **??**(b)-(d), we apply a blurring layer to the accelerometer that rotates the device by a small angle θ. In Figure **??**(b), θ is a random variable that is uniformly distributed between 0 and $10°$. Similarly, θ in Figure **??**(c) and (d) is uniformly distributed between 0 and $20/30°$. As shown, the more we rotate the device, the less distinct the estimation of S_z and O_z are for each device. The results in Figure **??**(c) are almost undistinguishable. Therefore, using data blurring in Sensibility Testbed, we are able to prevent fingerprinting and keep the anonymity of mobile devices in such a case.

7.3 Microbenchmarks

We measured the overhead incurred by running blurring layers with experiment code in Sensibility Testbed, to demonstrate that it is feasible to protect sensor data privacy on end-user devices without affecting user experience.

Figure **??** shows the runtime overhead of using varying number of blurring layers, i.e., policies, to protect sensor data. The experiment code uses calls `get_battery()` in the Repy sandbox. Each point in the figure is averaged over 100 iterations. Blue points show the execution time of us-



(a) Data without blurring. (b) $[0, 10°]$ rotation. (c) $[0, 20°]$ rotation. (d) $[0, 30°]$ rotation.

Figure 3: Accelerometer fingerprinting without blurring, and with different levels of blurring that partially randomize the accelerometer data. Each device's S_z-O_z pair is represented by the median and standard deviation of $10,000$ data samples. [Yanyan: it's better to draw a scatter plot with the result of a clustering alg overlaid on it.]

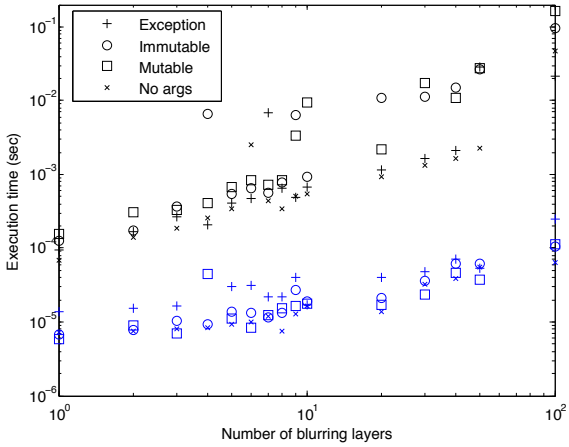


Figure 4: Overhead incurred by running varying number of blurring layers with experiment code. Blue points show the execution time of an experiment using blurring layers without verifying the runtime behavior. Black points show the execution time with the runtime behavior verified.

ing blurring layers without verifying the runtime behavior. Black points show the execution time with the runtime behavior verified, i.e., using a contract to check arguments, return values and exceptions. Note that a contract checks for both mutable and immutable data types [**?**] for arguments and return values. When the number of layers is below 10, the runtime is slightly increased from $10^{-5}s$ to $10^{-4}s$. Although the runtime overhead is higher with more layers, as Table **??** and **??** show, most projects need less than 10 policies.

Table **??**(a) shows the total runtime overhead when an experiment calls `get_accelerometer()`, without any blurring layer (no layer), with blurring layers but without verifying the runtime behavior (no-op), and with blurring layers that verifies the runtime behavior and rounds up accelerometer to two decimal points (round-up). As shown in the table, the total runtime is *linear* with an increasing number of layers/policies. The overhead per policy is almost constant, with no-op round 0.12 - 0.14 ms and round-up 0.14 - 0.21 ms. Therefore, the runtime overhead will not affect conducting an experiment or user experience.

Table **??**(b) shows the memory overhead without any blurring layer (no layer), and with blurring layers that verifies the runtime behavior and rounds up decimal points (round-up). As the results show, the memory overhead is almost constant, around 0.48 - 0.55 MB, until the number of blurring layers reaches 100. [Yanyan: is there an explanation?] Nevertheless, the overhead never exceeded 1 MB in all cases.

Operations		**Number of blurring layers**			
		3	**10**	**50**	**100**
No layer (baseline)		6.276 ms			
No-op	Total	6.654 ms	7.756 ms	12.75 ms	18.94 ms
	Per-layer	0.1259 ms	0.1479 ms	0.1295 ms	0.1266 ms
Round-up	Total	6.907 ms	7.991 ms	13.86 ms	20.64 ms
	Per-layer	0.2104 ms	0.1715 ms	0.1518 ms	0.1436 ms

(a) Runtime overhead.

Operations	**Number of blurring layers**			
	3	**10**	**50**	**100**
No layer (baseline)	6.246 MB			
Round-up	6.742 MB	6.801 MB	6.727 MB	7.230 MB
Overhead	0.492 MB	0.555 MB	0.480 MB	0.984 MB

(b) Memory overhead.

Table 4: Overhead with varying number of blurring layers.

7.4 Experience Using Sensibility

Our early prototype of Sensibility has been used by dozens of researchers. This has come in two forms: researchers performing extended practical deployments and researchers experimenting with Sensibility Testbed at a hack-a-thon. As

our IRB was recently approved, the experiments to date have focused on the usability of Sensibility Testbed as a platform for experimenters. As a result, this section primarily answers the question: How easy is it to use Sensibility Testbed?

Research deployment. Sensibility Testbed has been used by several research projects where the researchers installed Sensibility Testbed on their own devices and then conducted an experiment. A research group in Austria created the Open3G [**?**] project to investigate cellular technology coverage in Vienna. The researchers who created this project estimated that the effort to write the code to collect the sensor data took about an hour.

Sensibility Testbed was also used by a high school student as part of a vehicle data collection project [**?**]. The student connected their phone to the OBD2 port in their car and used Sensibility Testbed to capture data. The student then drove around the greater New York area and used this information to derive information about driving habits and patterns.

Hack-a-thons. In the past two years, we have hosted In 2014, we hosted a Hack-a-thon styled, one-day workshop co-located with the IEEE Sensors Applications Symposium (SAS) [**?**]. About twenty workshop participants with various backgrounds worked in teams for six hours and built four functional applications using Sensibility Testbed. None of the participants had any prior experience with this testbed, and many of them had no background in Computer Science. With this success, we hosted a second workshop with SAS in 2015, and will continue in 2016. Interesting experiments developed by the participants include a device network monitor: when the battery level of a device is low, WiFi or Bluetooth that requires high power is turned off.

[Yanyan: how much time a student spent doing some measurements (like Thomas's motion detection alg).]

[Yanyan: compare to other projects, developers can save thousands of lines of code, and reuse the same set of devices.]

7.5 Experience Implementing Policies

Experiments are not the only thing that needs to be implemented in Sensibility Testbed. It is important that security policies can be implemented for Sensibility Testbed.

[Justin: Add content about Richard, Sara, and my experience with security policies in the classroom.]

8. PRACTICAL CHALLENGES AND LIMITATIONS

Below we discuss the limitations of Sensibility Testbed when designing experiments to run on the testbed, and consider various challenges we met while building the system.

8.1 Limitations

In Sensibility Testbed, experiment code runs in a sandbox, and can access sensors only through privacy-preserving blur layers. In contrast to a native app (in our case, an Android app), this starkly curtails the experiment's access to the mobile device. For example, Sensibility Testbed does not provide a way to interact with the device owner through unsolicited on-screen messages or notifications that a native app could send[6].

Similarly, Sensibility Testbed limits access to other parts of the native API. For example, there is no way for an experiment to signal that it wants to handle an `Intent` [**?**], e.g. act as an MP3 player to open a file of this type, or handle phone calls, etc. [**?**]. [Albert: Do you think we should explain the concept of Intents more?]

Note that the operations above could well be added to the sandbox API – there is no technical obstacle for this. Rather, it was our fundamental security design choice for Sensibility Testbed to not allow these.

8.2 Challenges

We discuss the existing challenges of building Sensibility Testbed. We hope to resolve these challenges as we gain more practical experience with Sensibility Testbed. We approach questions of use, usefulness, technical design, and general architecture.

Use. [Yanyan: This doesn't sound like a challenge?] From the perspective of a researcher designing an experiment to run on Sensibility Testbed, there is a certain learning phase to go through in order to familiarize with the test and deployment tools, the sandbox API, operational aspects of a distributed system, and so on. Since the sandbox uses a different programming language than Android usually does, and the patterns to access sensor values are different as well, no previous experience with Android is required to program for Sensibility Testbed. Indeed, from our experience in educational contexts, an experimenter who brings along any programming knowledge will quickly learn how to use the sandbox, while specific Android experience does not necessarily provide an advantage.

Usefulness. Sensibility Testbed's current feature set makes it most useful as a passive, distributed, large-scale sensor network. It does not map well to interactive crowd sensing, crowd sourcing, and (geo) tagging tasks, for studies looking at personal data stored on the phone (such as performing research using the device owner's phone book), or introspection into the Android OS (which, e.g., PhoneLab provides). In short, we expect other approaches to remain valid and interesting besides Sensibility Testbed, but hope that its scale and distribution are attractive properties nevertheless.

[Albert: Could add questions regarding scale, too: "Is S.T. a meaningful way to scale out smartphone research?", "Shall we scale it out at all, or let Google / the mobile operators do it".]

Another interesting aspect we hope to understand better as we open Sensibility Testbed to the public is what motivates device owners to altruistically "donate" resources to researchers. We hypothesize that altruism is an incenctive, especially since Sensibility Testbed currently provides no native (GUI-based, notification) way for experiments to interface with the device owner. We note that there are projects that keep records of resource donations (like a high-score table), let contributors form teams, and so on [**?**]. This rewards contributors with some form of acknowledgment and publicity. Other projects appear to have utilized pure altruism to great effect. For example, the authors of [**?**] claims hundreds of installs from the Android app store, suggesting that there exists a sizable population of device owners that will install an app that gives them neither an obvious, direct

[6]Interactivity is still possible. An experiment may serve a web page on the local device through the web browser, and interact with the device owner as a web app.

benefit from interactivity, nor a form of publicity like the high-score table.

Technical challenges. Sensibility Testbed is designed to minimize the privacy repercussions of smartphone research by blurring the sensor values that experiment code sees. For this, we provide blurring layers that transform the sensor values, round them or otherwise limit their accuracy, replace them by random data, limit the access rate, and so on (Section **??**). In Section **??** we show that blurring is a meaningful strategy to counterbalance a researcher's desire for sensor access and a device owner's desire for privacy, and that our implementation of blurring can express complex interrelations between sensor values and accuracies. However, this does not guarantee that the existing blurring layers and configurations will always be able to address adequately all possible privacy concerns. Also, bugs in the sandbox or blur code might expose the device owner's privacy. To meet these objections, Sensibility Testbed includes a software updater so that we can fix any security issues and improve the coverage provided by the blurring layers.

[Albert: Another possible point is that interface-compatble, transpartent, drop-in blur functions might confuse more than they are useful, because there is no clear indication of the accuracy and rate that you get (cf. Leon's student's presentation on 2015-11-15). This is more criticism of drop-in blur rather than blur as such.]

Choosing and parametrizing blurring layers also is a usability challenge we currently try to resolve. As in Section **??**, researchers request access to sensors at certain accuracy levels and rates through their IRB, and Sensibility Testbed provides technical means to instantiate blurring layers that meet the approved specifications. For the device owner, intervention is currently limited to a binary choice, opt-in or opt-out. This provides a streamlined yet little expressive way for device owners to decide on participation in an experiment. In our prior work [**?**, **?**], we evaluated finer-grained sensor access settings, but found it difficult to balance expressiveness (when and how accurately which sensor or group of sensors may be accessed) and usability (i.e., the number and description of user interface widgets that enable the device owner to express their desired policy). Expressiveness is great for concerned device owners that want to make informed decisions, and the sandbox and blurring architecture allows for additional, prioritized policies to be instantiated before the researcher's IRB's and Sensibility Testbed's defaults. However, we need to understand better whether our existing interface satisfies the device owners.

Architectural challenges. Are blurring layers the right way to solve the problem? E.g., collect enough data to make use of temporal autocorrelation; secure enough. Also, deviow's sign away their rights with app installation anyway, don't they? Why bother with blur then?

Are IRBs the right way to solve the problem? E.g., rogue researchers that deliberately obfuscate what they will do with sensor readings, so as to breach privacy. E.g., ignorant IRBs that greenlight everything and put the burden on our default IRB (????).

9. RELATED WORK

Several techniques for protecting the privacy of mobile users have been proposed in recent literature. One means for enforcing privacy is to employ a third-party anonymizing agent that acts as a proxy between the data source and the service using the anonymized data [**?**, **?**]. This technique has several drawbacks. First, the data is accessible to the agent before it is anonymized, so the agent must be trusted to handle the unfiltered data. Second, if the agent is compromised, the privacy of all users is also compromised. Finally, frequent readings need to be sent to the agent, since it needs access to all raw sensor data before it can apply privacy algorithms. This can result in an unnecessary bottleneck, as the service utilizing the data may only need access to very infrequent readings. Sensibility Testbed overcomes these drawbacks, since it performs its privacy preservation techniques on the device itself without the need for a third-party agent. The only data leaving the device is that which goes directly to the researcher. Peer to peer techniques can also be employed as a method to protect privacy [**?**]. In this case, at least k peers participate to ensure k anonymity. When a user wants to be k-anonymous, he finds k - 1 peers to form a group with him and then cloaks his exact location into a spatial region covering this group. This technique relies on participation of multiple users, which is not always possible.

Data blurring has been suggested in [**?**] for privacy preservation during context reporting. The authors demonstrate location and time blurring of reports, also known as spatial and temporal cloaking. [Jill: can we say something about how we handle multiple sensors & privacy to make this a strong argument? Need to think about this one a bit more]

There have also been many works dedicated to detecting privacy violation from apps on mobile devices [**?**, **?**, **?**]. These approaches alert when sensitive data is exfiltrated from the device, either at runtime [**?**, **?**] or install time [**?**]. [Jill: add more references here] Although these systems notify the user when there is a potential privacy breach, they leave the mitigation decision up to the user. Sensibility Testbed, on the other hand, protects the user directly from exfiltration of sensitive data without requiring manual intervention at a critical time.

Some mobile data sets have been collected in order to provide researchers with more diverse data than they would typically have access to [**?**, **?**]. These data sets do not have the flexibility of dynamic collection, as offered by Sensibility Testbed, and quickly become out of date as new technologies develop. In addition, they typically do not enforce privacy of the user, since they simply require their subjects to sign consent forms, and often contain a limited amount of sensor data, if any at all. An important aspect of Sensibility Testbed is that it collects only as much data as is necessary for a researcher's experiment. In comparison to data sets that collect as much diverse and frequent data as possible from devices, our approach is much more scalable and usable, preserving battery and resource consumption. Furthermore, Sensibility Testbed makes tailored data collection easy for researchers by streamlining the implementation of IRB policies and broadening the pool of research subjects. This promotes researchers to use customized data collection on the most up-to-date technology and user behavior, without being forced to resort to outdated datasets. PhoneLab, [**?**], is a smartphone testbed for public experiment that was also developed for this purpose. However, it does not focus on security and privacy nor does it automate the IRB pol-

icy application process, which drastically reduces time and complexity for researchers.

10. CONCLUSION